TEXAS INSTRUMENTS

Digital Control Applications with the TMS320 Family

Application Book

Selected Application Notes

Digital Control Applications with the TMS320 Family

1991

1991

Digital Signal Processing Products

Digital Control Applications with the TMS320 Family

Edited by
Irfan Ahmed
Digital Signal Processing—Semiconductor Group
Texas Instruments Incorporated

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to or to discontinue any semiconductor product or service identified in this publication without notice. TI advises its customers to obtain the latest version of the relevant information to verify, before placing orders, that the information being relied upon is current.

TI warrants performance of its semiconductor products to current specifications in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Unless mandated by government requirements, specific testing of all parameters of each device is not necessarily performed.

TI assumes no liability for TI applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Texas Instruments products are not intended for use in life-support appliances, devices, or systems. Use of a TI product in such applications without the written consent of the appropriate TI officer is prohibited.

TRADEMARKS

Apollo is a trademark of Apollo Computer, Inc.
Apple and *Macintosh* are trademarks of Apple Computer Corp.
CROSSTALK is a trademark of Microstuf, Inc.
DEC, *VAX*, and *VMS* are trademarks of Digital Equipment Corp.
IBM, *OS/2*, *PC*, *PC–DOS*, *PC/XT*, and *PS/2* are trademarks of IBM Corp.
Intel is a trademark of Intel Corporation
MS–DOS and *MS OS/2* are registered trademarks of Microsoft Corp.
NEC is a trademark of NEC Corp.
Power–14 and *Power–Source* are trademarks of Teknic, Inc.
Sun is a trademark of Sun Microsystems, Inc.
UNIX is a registered trademark of AT&T Bell Laboratories, Inc.
VMEbus is a trademark of Motorola, Inc.
XDS is a trademark of Texas Instruments, Inc.

CONTENTS

PART III Implementation of Digital Controllers

PART IV Applications of Digital Controllers with the TMS320

Computer Peripherals

Motion Control and Robotics

Power Electronics

Automotive

Specialized Applications

BIBLIOGRAPHY

Preface

Using digital methods for controlling motors, robotic arms, or disk drives is not new. But technical advances in digital signal processing and high-performance digital signal processors (DSPs) such as the TMS320 family are rapidly moving digital control from the laboratory to the market place. Personal computers, automated manufacturing equipment, automobiles, military weapons, toys, and games are examples of products that are enhanced by the application of digital control technology.

This book introduces the reader to the concepts of signal processing and DSPs as they apply to digital control theory. It also presents a collection of published articles that review selected applications within the broad spectrum of digital control. The book is divided into four parts and a bibliography:

PART I Introduction to Digital Controllers

PART II Design of Digital Controllers

PART III Implementation of Digital Controllers

PART IV Applications of Digital Controllers with the TMS320

BIBLIOGRAPHY

Each part is introduced by the editor so that readers can gain insight into its purpose. The bibliography is furnished for those who wish to seek additional studies in the areas of automotive, control, and industrial applications.

Opportunities to design digital control systems have grown enormously over the past few years. This book is being published to aid practicing control engineers in becoming familiar and comfortable with digital control theory. It can also be a valuable tool for teaching at the undergraduate and graduate levels. The book brings together the latest concepts and applications in digital control theory to meet the needs of both new and experienced designers.

The editor, authors, and I hope that you enjoy this application book and gain valuable information to assist you in designing new digital control systems as well as modifying current systems.

Gene A. Frantz
Applications Manager
Digital Signal Processing
Texas Instruments Incorporated

PART I
Introduction to Digital Controllers

DSP-Based Control Systems

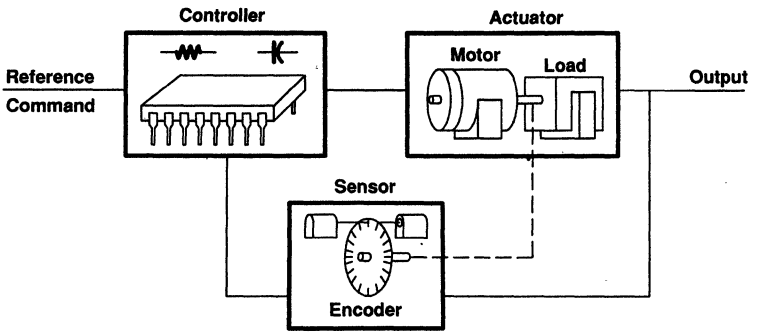
Digital signal processors (DSPs) are making digital control more practical. The special architecture and high performance of DSPs make it possible to implement a wide variety of digital control algorithms previously reserved for research work and simulation studies in laboratories. This general introduction discusses these aspects and uses of DSPs in digital control systems. It is followed by papers that discuss the suitability of DSPs for implementing digital controllers.

Control Systems

A control system commands or regulates a process in order to achieve a desired output from the process. As shown in Figure 1, a simple control system consists of three main components: sensors, actuators, and a controller. Sensors measure the behavior of the system or the process and provide feedback to the controller. Some of the sensors used in control systems are resolvers, shaft encoders, and current sensors. Actuators supply the driving and corrective forces to achieve a desired output. Typical actuators are AC/DC motors and valves.

The controller generates actuator commands in response to the commands received from the operator and to the feedback provided by the sensors. The controller consists of computation elements that process these signals to achieve a desired response from the entire system. The function of the controller is to ensure that the actuator responds to the commands as quickly as possible and at the same time to ensure that the system remains stable under all operating conditions. Typically, a controller will modify the frequency response of the system. The computational elements of the controller are implemented with either analog or digital components.

Figure 1. Control System

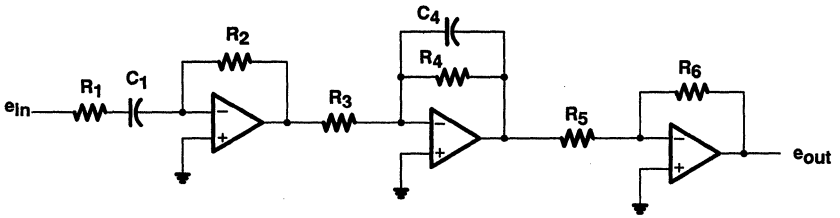


Analog Control Systems: Control systems have traditionally been implemented with analog components like operational amplifiers, resistors, and capacitors. Figure 2 shows a simple analog controller. These elements are used to implement filter-like structures that modify the frequency response of the system. Al-

though more powerful analog processing elements like multipliers are available, they are generally not used because of their high cost. In spite of the simpler processing elements, analog controllers can be used to implement high-performance systems.

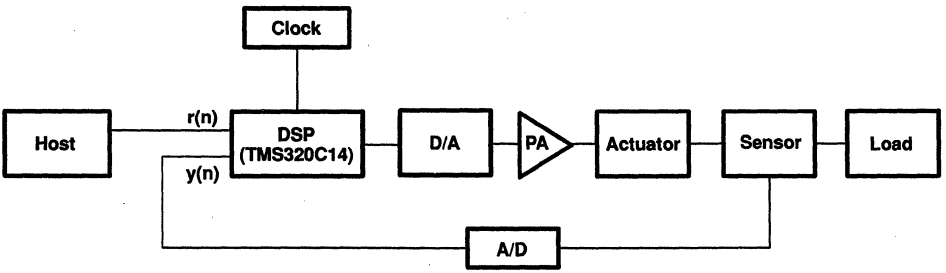
Most analog systems use single-purpose characteristics of an error signal like P (proportional), I (integral), D (derivative), or a combination of these characteristics. This limits most analog systems to designs based on classical control theory.

Figure 2. Analog Controller



Digital Control Systems: With the high performance and increasing reliability of microprocessors, digital controllers are taking over many applications from analog controllers. In the digital control system shown in Figure 3, a DSP (TMS320C14) processes the feedback/error signal [y(n)] in relation to the input/reference signal [r(n)]. A digital-to-analog converter (D/A) changes the digital output of the processor into an analog signal to drive the power amplifier (PA) and actuator. The D/A is typically represented by a ZOH (zero order hold). Similarly, on the input side, an analog-to-digital converter (A/D) interfaces the sensor's signal to the DSP. In addition, memory is required to store the commands necessary for the operation of the system; the TMS320C14 uses its on-chip memory for that purpose.

Figure 3. Digital Control System



Analog Versus Digital Controllers: Several tradeoffs have to be made in selecting a controller. Analog controllers continuously process a signal and can be used for very high bandwidth systems. They also give very high resolution of a measured signal and thus provide precise control. Analog controllers have

been around for a long time, are well understood, and are easy to design. They can be implemented with relatively inexpensive components.

On the negative side, analog controllers suffer from component aging and temperature drift. Even a perfectly designed controller will exhibit undesired characteristics after a while. Analog controllers are hard-wired solutions, making modifications or upgrades in the design difficult. Analog controllers are also limited to simpler algorithms from classical control theory, like PID and compensation techniques.

Most processes are analog in nature. Digital systems can only attempt to approximate them. The accuracy of this approximation determines the performance of the digital system. Digital controllers sample the signal at discrete time intervals. This limits the bandwidth that can be handled by the controller. The accuracy of the signal and coefficients that can be represented is limited by the resolution or the word length of the processor. Digital controllers require additional components like A/Ds and D/As, although newer processors include these components on the same chip. Digital controllers are relatively new, and their behavior is not thoroughly understood. Thus, designing high-performance digital controllers can be challenging.

However, digital controllers have some major advantages. They are not affected by component aging or temperature drift, and they provide stable performance. Designing in the z-domain helps to control their behavior more precisely. Digital controllers can be used to implement more sophisticated techniques from modern control theory, such as state controllers, optimal control, and adaptive control. They can also handle nonlinear systems. Digital controllers are programmable and make it easy to upgrade and maintain design investment. They can be time-shared to implement additional functions like notch filters and system control to reduce system cost. If digital controllers are designed properly, their advantages greatly outweigh their disadvantages. Table 1 compares analog and digital controllers.

Table 1. Analog Versus Digital Controllers

	Analog Controller	Digital Controller
Advantages	High bandwidth High resolution Ease of design	Programmable solution Insensitive to environment Shows precise behavior Implements advanced algorithms Capable of additional functions
Disadvantages	Component aging Temperature drift Hard-wired design Good only for simpler design	Creates numerical problems Must use high-performance processor Difficult to design

Processor Requirements for Digital Controllers

The choice of processor is critical in determining the performance and behavior of the digital controller. The poor performance of a digital system can generally be traced to selection of the wrong type of processor. Available choices are microcontrollers, general-purpose microprocessors, and DSPs. In addition, reduced instruction set computer (RISC) processors and bit-slice processors can be used, although their usage is not practical in most cases because of high cost. The following factors must be considered when selecting a processor:

- Architecture
- Performance
- Peripheral Integration

Architecture: Processor architecture is probably the most important factor. A control system is a demanding, realtime signal processing system. Control theory essentially deals with proper techniques for processing control signals. Processing signals in realtime raises numerical issues that must be resolved correctly, to ensure that performance from a digital controller is acceptable. Some of the problems resulting from inadequate processor architectures are quantization noise, truncation noise, limit cycles, and overflow-handling.

Quantization noise results from representing a signal in discrete or quantized magnitude levels. The signals and gain coefficients must be represented accurately without any loss of resolution for the smallest and largest magnitudes. A processor should support a large word length and scaling shifters to provide the resolution and dynamic range needed. This allows the signals and coefficients to be scaled to the full resolution of the processor. In some cases, floating-point support may be necessary if gain coefficients and signals are time-varying variables and have large dynamic ranges.

Truncation noise results from the processing of signals in realtime. Either a higher resolution or larger word length is needed for interim results. For example, the result of a 16×16 multiplication is 32 bits. If only a 16-bit storage capacity is available to the 32-bit resultant, the loss of the lower 16 bits is known as truncation error. A processor should be able to support a larger intermediate word length for interim results.

Limit cycles usually result from quantization and truncation errors. Insufficient resolution of the output causes the output to oscillate around the actual value without being able to reach it. Minimization of quantization and truncation errors reduces limit cycles.

Realtime processing requires a large number of mathematical operations. Sometimes the results will exceed the range handled by registers. When registers overflow, they may make a positive number turn negative. A processor should be able to handle this overflow situation without significant change in the value of the result.

Performance: Performance is another important criterion in selecting a processor for a digital controller. Sampling the signal at discrete time intervals requires certain performance requirements from the processor. The sampling rate should be at least 10 to 20 times the system bandwidth. The processor must finish processing the signal before the arrival of the next sample, or information will be lost. The processing requirement is also dependent upon the controller structure and the algorithm.

Another aspect of performance is the computational delay. The processor should finish processing the signal as soon as possible. Too much delay in calculation will add phase delay and will affect the phase margin and stability of the system. The processor should have fast instruction cycle time. It should also have a very fast multiplying time because multiplication is the basic element in discrete representation of all signal processing control algorithms.

Peripheral Integration: The final consideration is the amount of peripheral integration on the system. Peripheral integration is important from a system cost, ease of design/interface, and board space point of view. Typical peripherals are on-chip timers for sample rate selection, D/A or PWM (pulse-width modulation) circuitries to drive the actuators, either an A/D converter or an interface to optical encoders, or other sensors. In addition, bit I/O pins are required to look at system flags and other conditions.

Digital controllers have not been widely used, because most processors lack appropriate architectures for signal processing. Microcontrollers have been designed primarily to replace hard-wired logic, to handle data acquisition, and to implement logical decisions. On the other hand, microprocessors have been designed primarily to act as computing elements in computer systems. Thus, both types of architecture have failed to meet the requirements of signal processing; nevertheless, they have been used for it. Only DSP architectures can solve the fundamental problems encountered in control and other signal processing applications.

DSP Architectures

The TMS320 DSP architecture has been optimized for signal processing systems. Figure 4 shows the typical architecture of a basic DSP. Some of the key elements are multiple buses, 16-bit architecture, 32-bit registers, and hard-wired implementation of various functions. It minimizes numerical problems in signal processing and meets the bandwidth requirements of high-performance systems using sophisticated techniques. The features and benefits of TMS320 architecture are shown in Table 2.

Figure 4. DSP Architecture

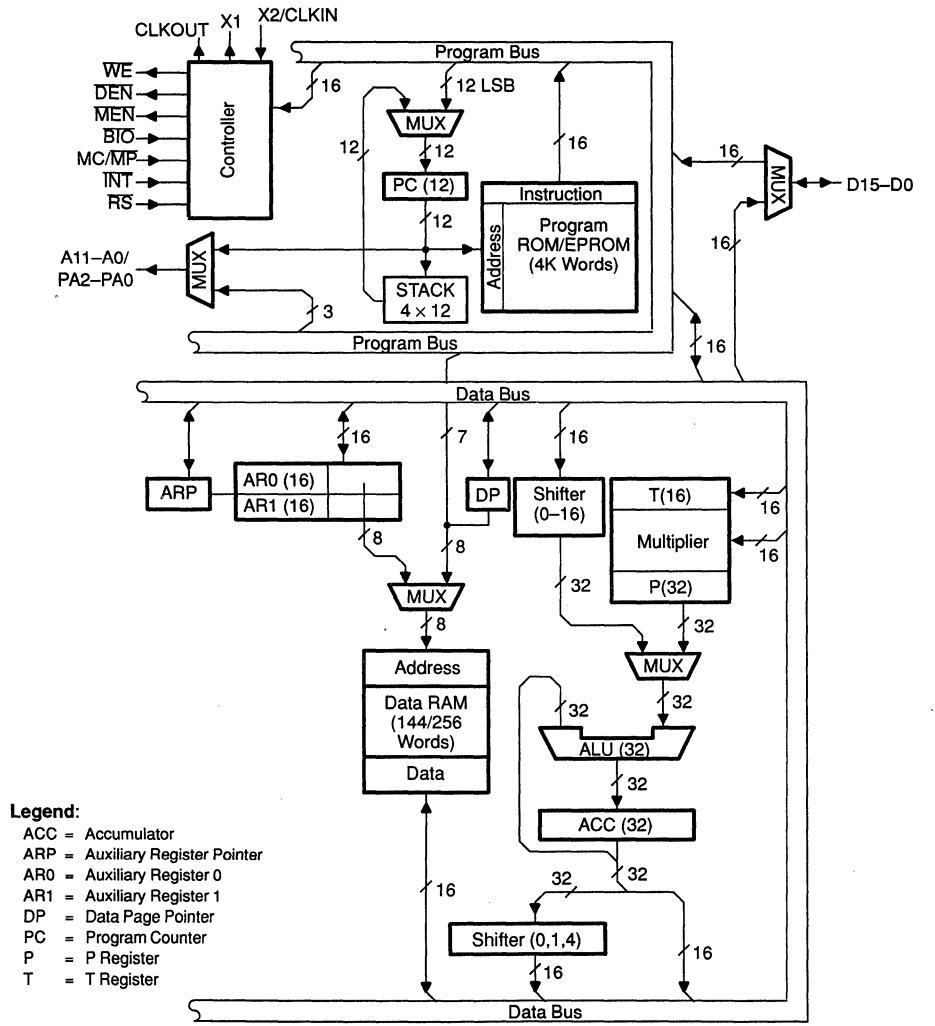


Table 2. TMS320 Architectural Features

Feature	Benefit
Single-cycle instructions	Execute advanced control algorithms in realtime
Pipelined architecture	Controls high-bandwidth systems
Harvard architecture	Simultaneously accesses data and instructions
Hardware multiplier	Minimizes computational delays
Hardware shifters	Have larger dynamic range
16-bit word length	Minimizes quantization errors
32-bit registers	Minimizes truncation errors
Hardware stack	Supports fast interrupt processing
Saturation mode	Prevents wrap-around of accumulator

To minimize numerical problems, the fixed-point TMS320 architecture has a 16-bit word length with 32-bit accumulator and other registers. The TMS320 DSPs include hardware shifters, which allow scaling, prevent overflows, and keep the required precision. These shifters allow shifting to take place simultaneously with other operations and without additional execution time.

Also, the instruction set has been optimized for signal processing. The DMOV instruction implements the z^{-1} operator. The MACD instruction implements four operations simultaneously: multiplies two values, moves data, accumulates previous result, and loads T register. To handle overflow during arithmetic operations, an overflow mode is included. This allows the accumulator to saturate at most positive or least negative values (similar to analog circuits), instead of rolling over and varying between positive and negative values.

Several features of DSP architecture provide the performance necessary to implement digital controllers. All functions are performed internally in hard-wired logic so that it takes a single cycle to execute most functions. Processors not optimized for signal processing usually perform functions in microcode and require numerous cycles to do so. The TMS320 devices employ an internal multiple-bus architecture that allows simultaneous fetching of instructions and data operands.

The TMS320 DSPs contain a hardware multiplier that performs a 16×16 multiplication in a single cycle. This minimizes the computation delay time and allows very fast sampling rates to be implemented for high bandwidth systems. An on-chip hardware stack reduces interrupt response time and minimizes stack pointer manipulations. Table 3 compares the architectural features of a DSP and a microprocessor/microcontroller (μP/μC).

Table 3. DSP Versus Microprocessor/Microcontroller

	DSP	Microprocessor/Microcontroller
Advantages	Signal processing architecture High performance Advanced control techniques Additional functions	On-chip peripherals Supervisory functions Familiar architecture
Disadvantages	Limited peripherals	Low performance Computation delay Numerical problems

Table 4. Feature Comparison

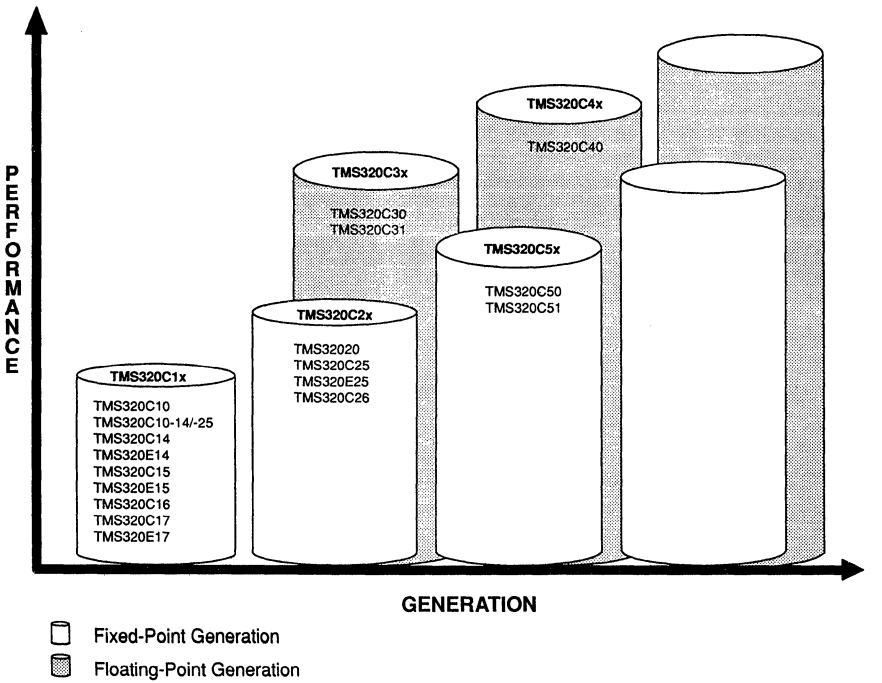
FEATURE	320C14	320C25	80C196	68000	68020	UNIT
Instruction cycle time	160	100	333	400	120	ns
Frequency	25	40	12	10	24	MHz
Multiply (16 × 16) → 32	0.16	0.1	2.2	7.0	1.0	ns
PID loop	2.2	1.3	27.0	25.0	4.8	µs
Matrix multiply (3 × 3) (3 × 1)	4.3	2.7	24.3	65.2	9.5	µs

Many on-chip DSP features enhance system integration; peripherals include RAM, ROM/EPROM, serial ports, timers, PWM, encoder interface, and parallel I/O. Table 4 compares performance characteristics of the TMS320C14, TMS320C25, and several µCs and µPs.

TMS320 Digital Signal Processors

The TMS320 family consists of five generations of fixed-point devices and floating-point devices (see Figure 5), offering different performance ranges. Members of each generation are object code and, in some cases, pin compatible.

Figure 5. TMS320 Family Roadmap



TMS320 Fixed-Point DSPs: There are three generations of TMS320 fixed-point DSPs: TMS320C1x, TMS320C2x, and TMS320C5x. All fixed-point DSPs have a 16-bit architecture with 32-bit ALU and accumulator. They are based upon a Harvard architecture with separate buses for program and data, allowing instructions and operands to be fetched simultaneously. They also feature a $16 \times 16 = 32$ hardware multiplier for single-cycle multiply operations, and a hardware stack for fast context-save operations. An overflow saturation mode prevents wrap-around. All instructions (except branches) are executed in a single cycle. Performance ranges from 5 MIPS (million of instructions per second) to 28.5 MIPS.

The TMS320C1x generation is based on the first DSP, the TMS32010, introduced in 1982. It includes 144/256 words of on-chip RAM and 4K words of address space. Instruction cycle time is 160 ns. Members of this generation include the TMS320C10, TMS320C14 and its EPROM version TMS320E14, TMS320C15/E15, and TMS320C17/ E17. All these devices have expanded memory of 256 words of on-chip RAM and 4K words of on-chip ROM/EPROM. The TMS320C14/E14 has been optimized for digital control applications. An additional member, TMS320C16, has an expanded memory address space of 64K words. Low-power versions are also available for 3-V systems.

The TMS320C2x generation is based on the TMS320C25, featuring 544 words of on-chip RAM and 4K words of on-chip ROM. Total address space is expanded to 64K words for both data and program. The instruction set has been considerably enhanced over the TMS320C1x instruction set, reducing the instruction cycle time to 100/80 ns. Other members include the TMS320E25 (an EPROM version of TMS320C25), TMS32020, and TMS320C26.

The TMS320C5x generation includes the TMS320C50 with 10K words of on-chip RAM and 2K words of on-chip ROM and the TMS320C51 with 2K words of on-chip RAM and 8K words of on-chip ROM. With an instruction set even more enhanced than the TMS320C2x instruction set, a TMS320C5x device is designed to execute an instruction in 35 ns. New features include a separate PLU, shadow registers for fast context save, JTAG serial scan emulation, and software wait states.

TMS320 Floating-Point DSPs: There are two generations of TMS320 floating-point DSPs: TMS320C3x and TMS320C4x (the first DSP designed for parallel processing). All floating-point devices have a 32-bit architecture with 40-bit extended precision registers and are based on a Von Neuman architecture. Multiple buses have been added for even faster throughput than the traditional Harvard architecture (program and data memory in separate spaces). Features include a hardware floating-point multiplier and a floating-point ALU.

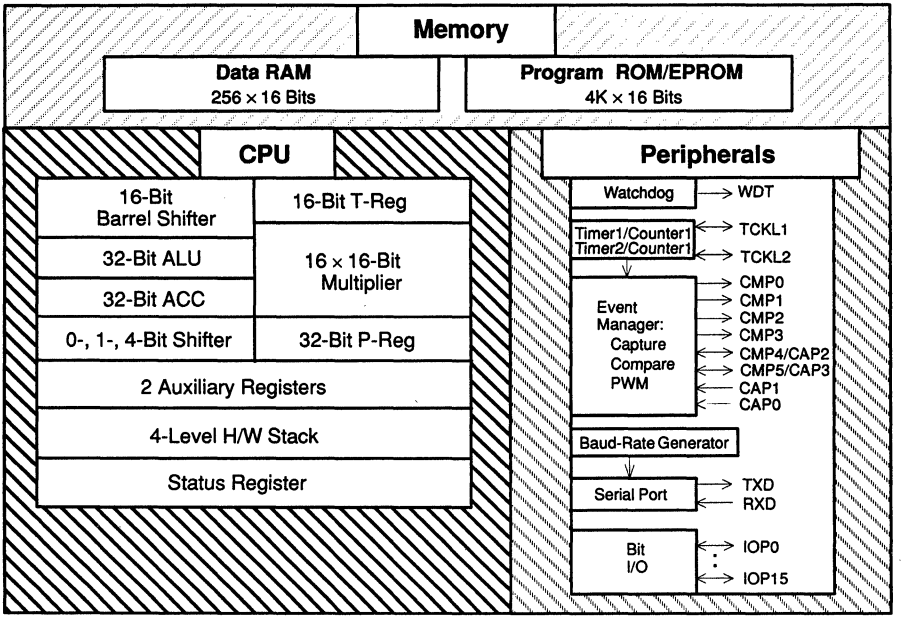
The TMS320C3x generation is based on the TMS320C30, featuring $2K \times 32$ words of on-chip RAM, $4K \times 32$ words of on-chip ROM, and a 64-word instruction cache. Other features include a separate DMA, two serial ports, two timers, two external 32-bit data buses, and a 16 M-word address space. Instruction cycle time is 60 ns, and the device is capable of performing up to 33 MFLOPS (million floating-point operations per second). Another member of the TMS320C3x generation is the TMS320C31.

The TMS320C4x generation includes the TMS320C40, a parallel digital signal processer. It includes six communication ports, a self-programmable/six-channel DMA coprocessor, a developing/debugging analysis module, two independent 32-bit memory interfaces, a 16G-byte addressing space, and two timers. Other features include two 4K-byte RAM blocks, one 16K-byte ROM block, and a 512-byte instruction cache. This generation is designed to execute an instruction in 40 ns, perform up to 275 MOPS (million operations per second), and provide a 320-Mbyte/sec throughput.

TMS320C14 – An Optimal Solution

The TMS320C14 is the first device that provides an optimal solution for implementing digital controllers on a single chip. Its TMS320C15 CPU meets the architectural and processing requirements for controllers, and it incorporates all the I/O peripherals needed in controllers and typically found in 16-bit microcontrol-

Figure 6. TMS320C14/E14 Key Features



- 160-ns instruction cycle
- 100% object code compatible with TMS320C15
- 4 16-bit timers
  - 2 general-purpose timers
  - 1 watchdog timer
  - 1 baud-rate generator
- 16 individual bit-selectable I/O pins
- Serial port – UART
- Event manager with 6-channel PWM D/A capability
- CMOS technology
- 68-pin PLCC and CLCC packages

lers. These peripherals include 16 pins of bit I/O, four timers, six channels of PWM, four capture inputs for optical encoder interface, a serial port with UART mode, and 15 interrupts. Figure 6 shows the key features of the TMS320C14.

The TMS320C14 can address 4K words of on-chip ROM or EPROM or off-chip memory, and 256 words of on-chip RAM. It has an on-chip hardware multiplier that performs a 16 x 16 = 32 multiplication in 160 ns. The TMS320C14 has a 32-bit ALU and 32-bit accumulator. It contains two hardware shifters and a four-deep on-chip hardware stack. Two auxiliary registers provide indirect and autoincrement addressing modes. The TMS320C14 has a general-purpose and DSP-specific instruction set and is 100% object code compatible with the TMS320C15 and other members of the TMS320C1x generation. The TMS320C14 has 16 pins of bit I/O that can be individually selected as inputs or outputs. In addition, each bit can be individually controlled without affecting the others. The 16-bit I/O port has the capability to detect and match patterns on the input pins and generate an interrupt when a specific pattern is detected.

The TMS320C14 contains four 16-bit timers. Two of the timers can be used as event counters with internal or external clocks. A third timer can be used as a watchdog timer and can also give a pulse output to drive external circuitry to indicate a time-out. The fourth timer can be used as a baud-rate generator for the serial port. Each timer is associated with a 16-bit period register and can also generate a separate maskable interrupt to the CPU.

The TMS320C14 has an event manager that consists of a compare subsystem and a capture subsystem. The compare subsystem has six compare registers that are constantly comparing their outputs with one of the timers. Associated with each compare register is an action register that controls all of the six output pins and two interrupt pins. The action registers determine an action that takes place on output pins in case of a match between the timer and a compare register. The compare subsystem can also be configured to generate six channels of high-precision PWM using a high-speed timer mode. In this mode, the compare subsystem can generate a PWM output that can be varied from 8 bits of resolution at 100 kHz to 14 bits of resolution at 1.6 kHz.

The event manager also contains four capture inputs that capture the value of a timer in a four-deep FIFO when a certain transition is detected on a capture input pin. Each capture input can detect pulses as narrow as 160 ns and can also generate a maskable interrupt to the CPU.

The TMS320C14 serial port is capable of full-duplex asynchronous operation with a transmission/reception rate of up to 400K bps. The serial port has a separate dedicated timer for generation of baud rates. The serial port also supports two industry standard protocols for interprocessor communication.

Finally, the TMS320C14 has a total of 15 internal/external interrupts, which can be individually masked. All the interrupts trigger a master interrupt that is controlled by the INTM bit in the status register.

Summary

The TMS320 family of DSPs solves many of the fundamental problems of signal processing in digital servo control systems. With their processing power, it is now possible to implement advanced concepts from modern control theory in cost-effective control systems. DSPs provide the precision and bandwidth of analog systems and at the same time provide the reliability of digital systems. Newer DSPs like the TMS320C14 provide a single-chip solution for the majority of servo control applications.

References

1. Texas Instruments, *TMS320C1x User's Guide*, 1989.
2. Texas Instruments, *TMS320C14/E14 User's Guide*, 1988.
3. Texas Instruments, *TMS320C2x User's Guide*, 1990.
4. Texas Instruments, *TMS320C3x User's Guide*, 1990.
5. Texas Instruments, *TMS320C4x User's Guide*, 1991.
6. Texas Instruments, *TMS320C5x User's Guide*, 1990.
7. Texas Instruments, *Digital Signal Processing Applications with the TMS320 Family*, 1986.
8. Texas Instruments, *Digital Signal Processing Applications with the TMS320 Family*, Vol. 2, 1990.
9. Texas Instruments, *Digital Signal Processing Applications with the TMS320 Family*, Vol. 3, 1990.

DIGITAL SIGNAL PROCESSORS

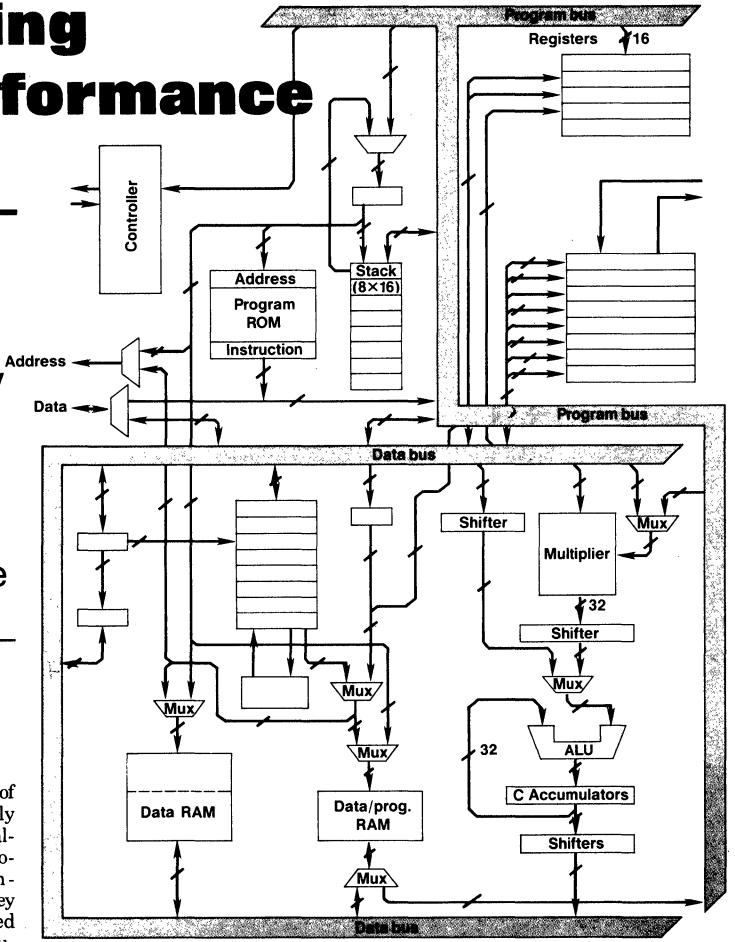
Simplifying high-performance control

Modern control algorithms often demand real-time speed that ordinary microcontrollers cannot provide. Digital signal processors are optimized to handle such tasks.

IRFAN AHMED
STEVEN LINDQUIST
Texas Instruments Inc.
Houston, TX

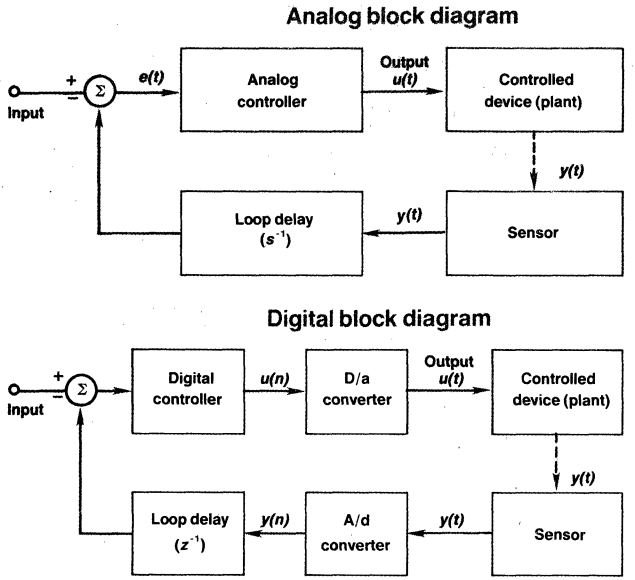
Electronic control systems of few years ago were frequently designed around a general-purpose microprocessor or microcontroller. But though conventional micros are versatile, they sometimes fall short when applied to high-speed tasks in telecommunications and computers, and in electromechanical tasks such as automotive engine control.

The problem is that advanced control algorithms, as used in digital filtering and discrete Fourier transforms, demand numerous multiplications and additions. When done in software on an ordi-



Many digital signal processors are built with a Harvard architecture, where data and instructions occupy separate memories and travel over separate buses to speed program execution. The two buses are evident in this simplified block diagram of a TMS320C25, a second generation CMOS processor. Other features of note on the 68-pin chip include eight auxiliary registers and a hardware multiplier specially designed to handle complex arithmetic.

Reprinted, with permission, from *Machine Design*, Sept. 10, 1987.



When reduced to a block diagram, traditional analog control systems resemble the digital counterpart. But analog controller qualities are determined by circuit elements, while those of digital counterparts are programmed in a few lines of code.

nary processor, these operations can consume too much time to provide high-speed control.

Most new classes of control algorithms, along with other algorithms such as state modeling, state estimation, Kalman filtering, and optimal control can be implemented with analog circuitry. In practice, however, it is difficult to design analog hardware that offers the precise and often nonlinear behavior required in such approaches. In addition, it is often expensive to build in the needed stability and temperature range.

The modification of a control algorithm implemented in hardware can also be complicated. Changes may sometimes be made simply by substituting a simple component, but can also involve redesigning part of the control system.

An approach to solving the speed requirements associated with modern control algorithms is to use a special kind of processor chip. Digital signal processors (DSPs) are constructed to speedily perform the

A DEADBEAT CONTROLLER

Some advantages of a DSP become clear when implementing functions that are difficult or impossible to realize in analog controllers. A deadbeat controller serves as an example.

In principle, analog controllers require an infinite time to settle to a reference input signal. In practice, they usually approach the reference quickly enough for most purposes. But when extremely fast settling is needed, digital deadbeat controllers may be preferred.

As a review, deadbeat controllers are those that settle to a steady state in as few samples as possible. If n is the order of the controller, deadbeat controllers reach steady state in $n+1$ samples. They are constructed by selecting the proper elements for the feedback loop. Control theory says that this behavior is obtained by cancelling all of the poles of the plant and replacing them by poles at the z-plane origin.

Obviously, because controller poles must have a one-for-one relationship with those of the controlled device, a deadbeat controller will always have the same order as the plant. Such controllers can be implemented with just a few DSP instructions. The resulting program executes quickly and, thus, is suitable for adaptive control systems where the control algorithm must be calculated repeatedly.

Deadbeat controllers should be avoided in cases where the plant poles lie outside or close to the unit circle in the z-plane. In such applications, imperfect pole cancellation can lead to serious system instability.

The basic deadbeat control algorithm is given by

$$G_{db} = \frac{p_0 + p_1 z^{-1} + p_2 z^{-2} + \ldots}{q_0 + q_1 z^{-1} + q_2 z^{-2} + \ldots}$$

If the application transfer function is given by

$$G_{plant} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots}{a_0 + a_1 z^{-1} + a_2 z^{-2} + \ldots}$$

then the deadbeat coefficients are defined in terms of the plant coefficients by

$$p_0 = r/(b_0 + b_1 + b_2) \qquad q_0 = r - p_0 b_0$$
$$p_1 = a_1 p_0 \qquad q_1 = b_1 p_0$$
$$p_2 = a_2 p_0 \qquad q_2 = -b_2 p_0$$

Here, r is the reference input or desired state of the controlled device. Most systems can be analyzed by assuming an arbitrary value, such as unity, for r. The accompanying 15-step program for the 32010 DSP implements the algorithm.

Program

```
DBEAT   IN      X0, PA0     READ INPUT SAMPLE
        ZAC                 ACC = 0
        LT      Y2          T = y(n-2)
        MPY     Q2
        LTD     YN          ACC = -q2 * y(n-2)
                            y(n-1) → y(n-2)
        MPY     Q1
        LTA     X2          ACC = -q1*y(n-1) - q2*y(n-2)
        MPY     P2
        LTD     X1          ACC = p2*x(n-2) - q1*y(n-1)
                              - q2*y(n-2)    x(n-1) → x(n-2)
        MPY     P1
        LTD     X0          ACC = p1*x(n-1) - q1*y(n-1)
                              - q2*y(n-2)
        MPY     P0
        APAC                ACC = p0*x(n) + p1*x(n-1)
                              + p1*x(n-2) - q1*y(n-1)
                              - q2*y(n-2)
        SACH    YN          STORE OUTPUT
        OUT     YN, PA1     SEND IT TO PORT 1
```

kinds of arithmetic operations associated with digital filtering and processing. Most DSPs are built with what is called a Harvard architecture. This configuration is unlike conventional computer architectures in that it employs separate data and instruction memories that are accessed by separate buses. The benefit of this arrangement is increased speed because instructions and data can move in parallel instead of sequentially.

In addition, these ICs generally carry high-speed hardware multipliers and fast on-chip memories that eliminate delays associated with shuttling information on and off chip to peripheral devices. This promotes fast program execution.

For example, a DSP can fetch an

APPLYING DSPs IN SIMPLE CONTROL

A PID loop provides a simple example of how DSPs can be applied to common control problems. A basic analog PID (proportional-integral-differential) control algorithm is frequently defined by

$$u(t) = K_p e(t) + K_i \int e(t)\,dt + K_d\, de/dt$$

where e = some input voltage that varies over time. U = output voltage and K_p, K_i, and K_d are constants. This equation indicates that output voltage is proportional to the sum of an input error voltage, the time integral of the error voltage, and the time rate of change of the error voltage.

For the sake of review, PID control functions as follows. The integral term is added to the basic proportional term to reduce the steady-state error to zero. It makes possible a nonzero control output even when the error signal (controller input) is zero. In this manner, it serves to anticipate increasing error and apply a correction faster than would normally be the case.

The derivative term is added to improve the stability of the feedback loop. It allows the system to provide more correction for a faster rate of change of error. The proportional K constants are usually chosen using standard s-plane techniques such as root-locus diagrams, Routh-Hurwitz criterion, Bode plots, and state variable techniques.

A typical approach to implementing a digital control algorithm is to write the analog transfer function in the usual way using Laplace transforms, and then convert the equation into a sampled data version through use of z transforms. Next, the digital transfer function is converted to a difference equation in the time domain. A program is then written for a DSP that implements this time domain difference equation.

The two most widely used analog/digital transformation methods are the matched pole-zero (also called matched z-transform) and the bilinear transformation. Though the former method is simpler, it is somewhat heuristic and does not always produce a suitable controller. The bilinear transformation is more complex but mimics analog functions more closely. This is because it uses the trapezoidal rule instead of rectangular areas to solve the differential equation specifying the transfer characteristic.

The bilinear transformation converts expressions in Laplace transforms into corresponding equations in z using the identity

$$s = \frac{2(z-1)}{T(z+1)}$$

where T = sample period.

Under the bilinear transformation, parallel or cascaded control elements retain their respective structures. Overall frequency response is treated less faithfully, however. Low frequencies map accurately, but high frequencies do not.

For that reason, a frequency prewarping scheme is usually employed with this technique. Here a single critical frequency is matched in the analog and digital domains by replacing each s in the analog transfer function with $(\omega_o/\omega_p)s$, where ω_o is the frequency (in rad/s) to be matched in the digital transfer functions and

$$\omega_p = (2/T)\tan(\omega_o T/2)$$

To summarize, the design of any digital control function usually begins with the specification of a few critical frequencies (ωs) and magnitude requirements (Ks). These are prewarped into a set of analog specifications by plugging each ω into the prewarping formula. The resulting frequencies are then used in deriving the Laplace transform version of the transfer function. This function in s is derived in the usual way, and then is converted to a digital transfer function in z, generally by means of the bilinear transformation. Finally, an inverse z transformation applied to this expression yields a difference equation that is expressed in terms of sample times. This equation can then be coded into a DSP.

The procedure can be readily applied to the equations

Program

```
PID     IN      E0 PA0      GET NEW SAMPLE
        MPYK    0           CLEAR P REGISTER
        LAC     Y2          ACC = y(n-2)
        DMOV    YN          y(n-1) → y(n-2)
        LT      E2
        MPY     K2
        LTD     E1          ACC = y(n-2) + K2•e(n-2)
        MPY     K1
        LTD     E0          ACC = y(n-2) + K1•e(n-1) + K2•e(n-2)
        MPY     K0
        APAC                ACC = y(n-1) + K0•e(n) + K1•e(n-1)
                                  + K2•e(n-2)
        SACH    YN
        OUT     YN, PA1
```

defining a PID loop. The exact sequence of operations is too lengthy to be given here, but the resulting difference equation is

$$u(n) = u(n-2) + K_1 e(n) + K_2 e(n-1) + K_3 e(n-2)$$

$$K_1 = K_p + 2K_d/T + K_i T/2$$
$$K_2 = K_i T - 4K_p/T$$
$$K_3 = 2K_d/T - K_p + K_i T/2$$

Here $e(n)$ is the nth input sample of the controller, the nth sample of the error voltage; $u(n)$ is the nth output sample of the controller, $u(n-1)$ is the $n-1$ sample, and so forth.

Because this equation represents quantities in terms of sample number rather than as functions of time, it can be easily implemented in software for a DSP. The accompanying 13-instruction program for the 32010 processor executes the above PID difference equation in about 2.6 μs when the processor runs at 20 MHz. In contrast, a similar program running on a general purpose processor such as a 10-MHz 68000 would consume 25.4 μs, or 26.1 μs on a 12-MHz 8096 processor.

BASICS OF DIGITAL FILTERING

One of the most common applications for DSPs is in digital filters. Although the subject is complex, the basic process of obtaining filtering action from a DSP can be understood in terms of the simple control concepts outlined previously.

Remember that a control function can be converted to an equivalent difference equation by first prewarping critical frequencies, transforming the s-plane equation into the z plane, and taking the inverse z-transform to yield a difference equation expressed in terms of sample times. The same process applies when synthesizing an equation for a digital filter.

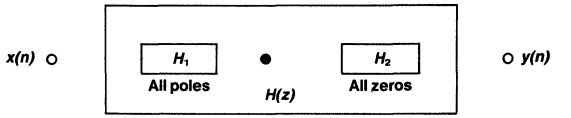
The usual way of expressing the digital filter difference equation is as a product of two transfer functions where the first (H_1) contains only the denominator or poles and the second contains only the numerator or zeros. The output of the filter is obtained by calculating an intermediate result. The result comes from operating on the input with H_1 and then operating on the result with H_2.

The resulting equation is usually in terms of the accompanying block diagram which is known as a direct form II realization. The delay blocks are a form of storage and delay used to generate the n-k samples, the $\times$ represents multiplication between a sample and the corresponding coefficient, and the Σ is a summing operation.

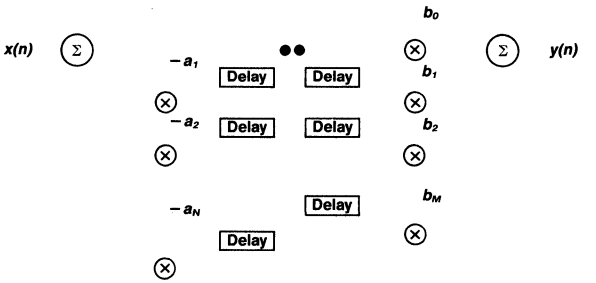
Analysis shows that the two branches of delay elements can be combined because they both refer to delayed versions of the intermediate result. The resulting simplified diagram is often expressed in shorthand fashion as shown.

One benefit of direct form diagrams is that they show clearly why DSPs contain certain special instructions in their instruction set. For example, the DMOV instruction for the 32010 moves data from one data memory location to the next higher location to implement the z^{-1} delay element. The LTA instruction loads a T register with a multiplicand while also accumulating the previous multiplication results. Such instructions allow a sum-of-products expression to be calculated at the rate of 400 ns per element.

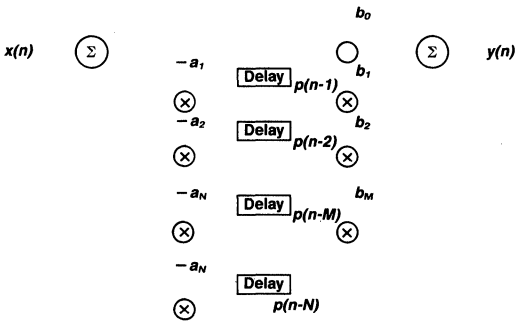
Equivalent transfer function model



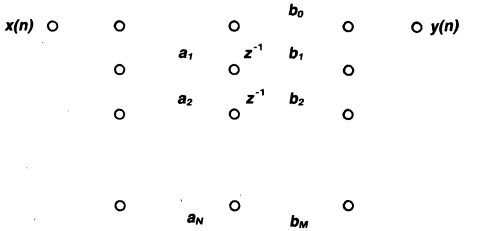
Equivalent block diagram



Form II realization

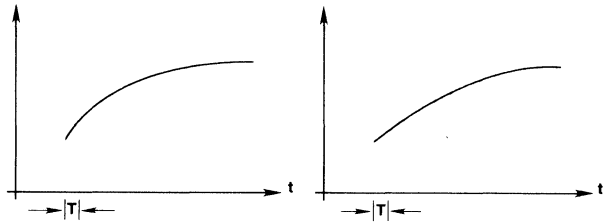


Shorthand form II diagram



instruction while loading two numbers into its multiplier. An ordinary processor such as a 68000 might chew up as many as 80 clock cycles to multiply two numbers and add the result to an existing sum. A DSP chip such as TI's TMS320C25 can do the same operation in a single clock cycle covering about 100 ns.

DSPs take the form of single-chip ICs, specialized board-level computers, and bit-slice chips optimized for signal processing operations. Of these, single-chip versions are the most widely used because their low cost makes digital signal processing practical in a variety of applications, ranging from consumer electronics to automotive engine control.



The bilinear transformation maps analog transfer functions into the digital domain through the representation of error signals as a series of trapezoids. The simpler matched pole-zero transformation is less precise because it employs a more crude rectangular approximation.

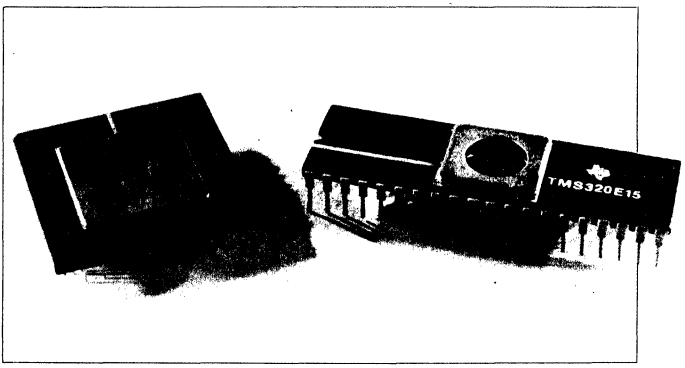
Simple approach

DSP architecture arises from the calculation sequence used to synthesize digital filters and discrete Fourier transforms. These two functions form the basis for much of the digital signal processing now used in industry. The calculation sequence, in general terms, is one of a linear constant coefficient difference equation:

$$y(n) = -\sum_{k=1}^{N} a_k y(n-k) + \sum_{k=0}^{M} b_k X(n-k)$$

This equation basically says that any output y can be expressed as a weighted sum of the input x at the present time n, past inputs $x(n - k)$ for some number of past samples k, and past outputs $y(n - k)$. Terms a_k and b_k are the weighting factors. A computer optimized to quickly synthesize this equation must be able to store an input, multiply it by a weighting factor, and sum it with previous inputs.

DSP architecture provides these functions by incorporating a large degree of parallelism, carrying out multiple operations per machine cycle. The ability to perform parallel fetches from two registers and store the contents in two memory locations is an example. In addition, the memory on chip is extremely fast and constructed in ways designed to facilitate data transfers. For example, the Harvard architecture on the TMS320 DSP family contains provisions for transferring information between data and instruction memories.



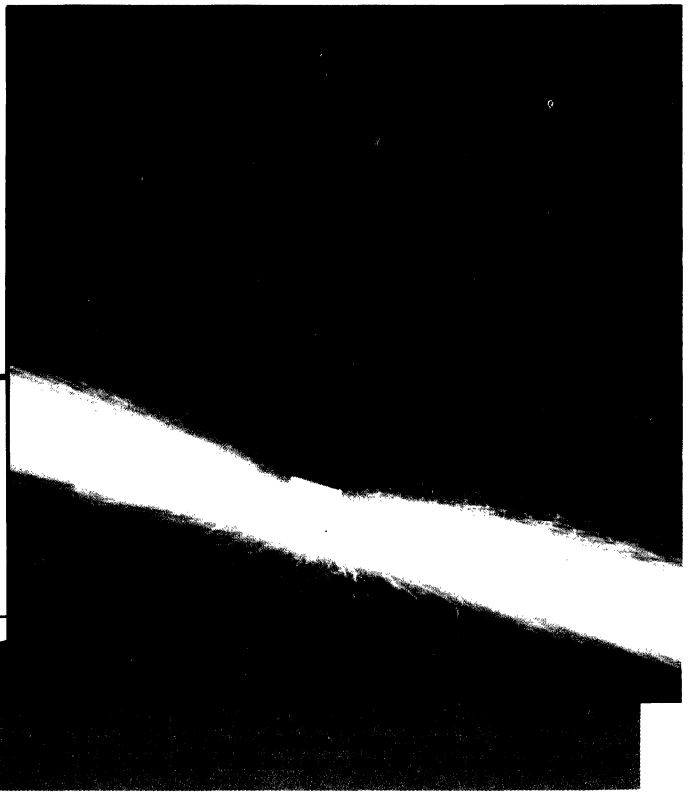
Compared to first generation DSPs such as the EPROM-version 320E15, second generation devices sport higher speed and more on-chip features. The cmos 320C25, for example, provides a 100 ns cycle time and 544 bytes of on-chip RAM.

Because DSPs typically do not need to store large programs or blocks of data, they usually lack the extensive memory-management circuitry found in general-purpose microprocessors. Nevertheless, DSPs have become very powerful. The first such chips had only limited instruction sets and memory, and were limited to fixed-point (integer) calculations.

In contrast, DSP chips today are second and third-generation devices that eliminate such problems. They typically use clock rates of 20 MHz, and 40 MHz clocks are not unheard of. Newer DSPs also provide on-board functions such as serial ports, analog/digital and digital/analog converters, EPROM, bit I/O timers, and similar functions that enhance capability.

The cost of single-chip DSPs is on the order of a few dollars, comparable to that of conventional microprocessors used in control applications. Recently developed DSPs tend to provide sophisticated functions that enable them to operate with video and radar-frequency signals. Examples of such functions can be found in the TMS320C30, a third-generation chip. The device provides floating-point math capability, facilities for handling off-chip memory as well as on-chip RAM and ROM, a more extensive instruction set, and clock cycle times of about 60 ns. ■

Taking Control with DSPs



New DSP microcontrollers offer many improvements over current analog and digital control systems.

TOM BUCELLA
Teknic Inc.
Rochester, NY

IRFAN AHMED
Texas Instruments
Houston, TX

In many control systems, digital signal processors (DSPs) are relegated to computational chores that bog down conventional processors. But their limited role is expected to increase because new DSPs can manage I/O tasks as well.

These revolutionary ICs are basically microcontrollers with on-chip digital signal-processing hardware. They make possible single-chip control for real-time multiaxis systems. In addition, software and hardware support tools simplify their use in motion applications.

Analog to digital

Digital signal processors have enabled control systems to advance from analog to full-digital implementations. Microprocessor-based systems are only a halfway point. They are an improvement over analog controllers, but lack processing speed to totally displace older technology. DSPs, on the other hand, have powerful arithmetic logic units (ALUs) capable of high-speed processing.

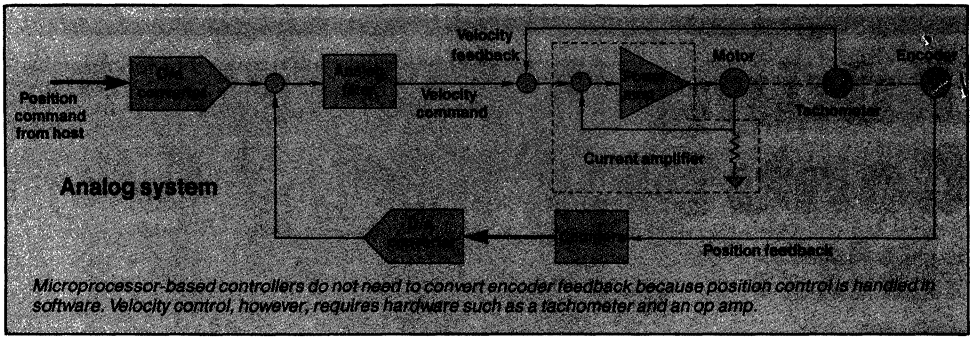
Early solid-state controls consisted of hard-wired analog networks built around operational amplifiers. Analog controls offer two distinct advantages over digital systems. First, they provide higher speed control by processing input data in real time. They also have higher resolution over wider bandwidths because of infinite sampling rates. However, they have several drawbacks.

Analog component values vary with age and temperature, necessitating periodic adjustments to maintain consistent operation. For example, high-gain amplifier parameters such as offset and gain can drift by as much as 20% in their lifetime. Such fluctuations can cause major changes in the frequency response of band-pass and band-reject filters.

Other weaknesses stem from the construction of analog hardware. Reliability can be a problem because analog systems typically have high part counts. Also, component lot tolerances frequently complicate design and may introduce error. And field upgrades are nearly impossible, often requiring redesign and repackaging of the hard-wired circuits.

In contrast, microprocessor-based motion systems offer many improvements over their analog counterparts. Drift is eliminated because most functions are performed digitally. Upgrading or modifying a digital system usually involves rewriting the software: hardware does not need to be replaced. And single-chip solutions for simple applications are possible with microcontrollers that have on-

Reprinted, with permission, from *Machine Design*, Oct. 12, 1989.



Microprocessor-based controllers do not need to convert encoder feedback because position control is handled in software. Velocity control, however, requires hardware such as a tachometer and an op amp.

chip hardware for I/O operations.

Even the best microcontrollers, however, have limitations. In many applications, they are too slow. Processor time is largely spent managing system I/O, leaving little time for data manipulation. Also, microcontroller ALUs are not suited for high-speed processing. Only simple control algorithms can be supported. Real-time, adaptive, or multiaxis control is inefficient and often impossible because computations overload the processor.

Most processor-based systems employ lookup tables to avoid calculations. But interpolation and round-off errors reduce precision. Also, lookup tables can consume vast memory space, often limiting algorithms only one variable.

To reduce table size, data word lengths are sometimes shortened. But this approach may introduce limit cycling. Cycling occurs when output commands have fewer significant digits than the required operating point. For example, a set point of 7.42 cannot be achieved with two-digit word. In that case, the output would cycle continuously between 7.5 and 7.4.

There are several reasons why standard microcontrollers are slow and inefficient in complex applications. One is that they have only a single bus for both program commands and data. Another reason is that a conventional ALU multiplies numbers by repetitive addition. These hardware limitations slow the processor and ultimately reduce sampling rates.

DSP microcontrollers, on the other hand, are geared for high-speed control applications. A dual-bus (Harvard) architecture allows simultaneous processing of program instructions and data. The ALU features hardware multipliers that handle multiply/accumulate operations in a single instruction cycle. This is particularly important for motion-control applications because control algorithms are dominated by multiply and accumulate instructions.

While general-purpose processors take from 5 to 20 μs to multiply two 16-bit numbers, DSPs need only 60 to 150 ns, about 100 times faster. Such speed improvements make possible sampling rates of over 20 kHz. They also allow controllers to extract more information from feedback data during the time between sampling periods. For instance, DSPs can provide speed control by calculating velocity from encoder position data. Microprocessor-based systems, on the other hand, are too slow to estimate velocity and typically use tachometers for feedback.

Other hardware enhancements include barrel registers. Barrel registers allow DSPs to scale numbers in a single instruction cycle. Scaling pushes all insignificant zeros to the right side of the number field by shifting the data string to the left. These maneuvers increase precision by making room for less significant bits during calculations. They also minimize truncation errors. Conventional processors scale numbers in software, shifting them one bit at a time. A one-bit word in a 16-bit field may eat up 15 clock

THE BASICS OF CONVERSION

A basic DSP controller consists of an analog-to-digital (a/d) converter or quantizer on the front end to sample analog input signals. A high-speed processor operates on the data according to a control algorithm in memory. The processor provides digital outputs that may be tapped directly or converted to an analog format through a digital-to-analog (d/a) converter.

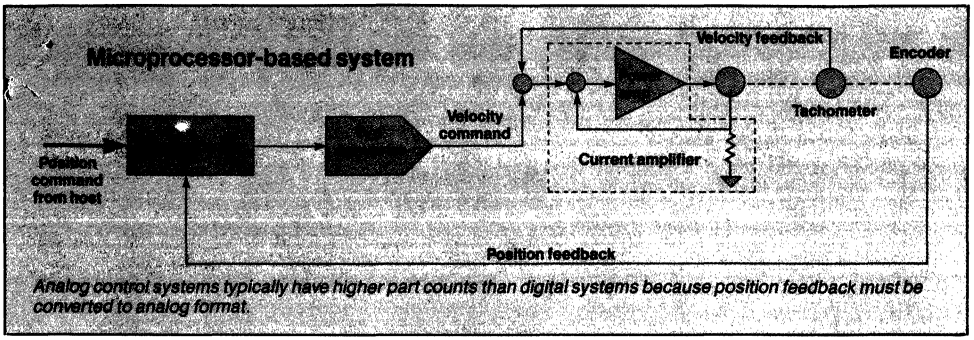
DSP systems depend on a/d converters to obtain accurate measurements of analog signals. A/d converters sample continuous-data (analog) signals by capturing small slices at periodic intervals. The sampled signal is reconstructed and the DSP sees a succession of amplitude-modulated, zero-width pulses whose envelope conforms to the analog signal.

Accuracy of digitized information is a function of the number of data points sampled per second. The higher this number, the better the resolution of the reconstructed signal. Minimum sampling rate that provides distortion-free data is determined by Nyquist sampling theory. For an analog signal whose highest frequency component is f_c, the minimum sampling rate is $2f_c$. Typically, a sampling frequency of 6 to $10f_c$ is used.

If sampling frequency is too low, the DSP sees a so-called alias signal at a frequency substantially different from f_c. Once aliasing occurs, it is impossible to recover the original signal. Filtering or any other technique cannot bring it back.

A simple method to prevent aliasing is to increase the sampling rate. Using a filter that limits the analog signal's bandwidth is another. But aliasing cannot be totally prevented because of the filter's nonideal qualities and high-frequency noise components in the analog signal.

Another concern with the a/d input stage is aperture time, the length of time the sampling pulse takes to get a slice of

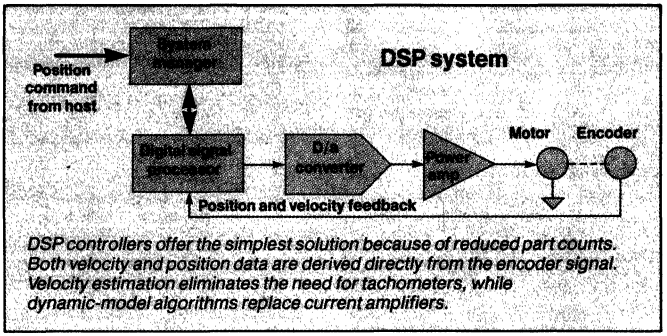


Analog control systems typically have higher part counts than digital systems because position feedback must be converted to analog format.

cycles.

Improvements also result from reduced instruction sets oriented toward signal processing. For example, a single DSP command called MACD multiplies two numbers, adds the product to an accumulator, and shifts the data to an adjacent register. This sequence of operations synthesizes a digital filter pole or zero. Commands such as MACD simplify software development by reducing the number of code lines.

DSPs, furthermore, allow controllers to provide functions impossible with analog or microprocessor systems. For instance, they can produce sharp-cutoff notch filters that eliminate narrow-band mechanical resonances. In motion systems, mechanical vibrations may occur from about 1 to 100 Hz, with some as high as 10 kHz. Notch filters remove energy that would otherwise excite resonant modes and possibly make the system unstable.



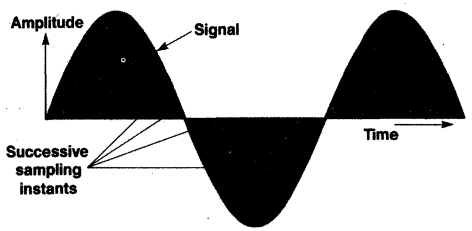
DSP controllers offer the simplest solution because of reduced part counts. Both velocity and position data are derived directly from the encoder signal. Velocity estimation eliminates the need for tachometers, while dynamic-model algorithms replace current amplifiers.

In addition to control functions, the analog input. Its maximum value depends on required accuracy and analog signal slew rate. Signals with high slew rates need shorter aperture times to maintain accuracy.

After sampling, the quantizer (a/d converter) changes the data to a digital format. Rounding signal magnitude up or down to the nearest threshold level introduces quantization error. Threshold levels are discrete values that digital strings can assume. Quantization error is the difference between the actual analog signal and the nearest threshold value. Maximum quantization error for a linear ramp signal, for instance, is one-half the separation between adjacent threshold levels.

As threshold levels move closer together, resolution increases and the discrepancy between the analog input and the quantized output decreases. Quantization error can only be reduced by increasing the number of discrete conversion steps or threshold levels.

Another key component in DSP systems is the digital-to-analog (d/a) converter. The d/a converter accepts digital inputs and produces analog outputs.



A digitally reconstructed signal is a succession of amplitude-modulated, zero-width pulses whose envelope conforms to the original analog signal. Analog-to-digital converters record signal amplitude at periodic intervals between which all control algorithm computations must be completed.

One type of converter employs electronic switches that turn on a voltage or current in response to the input. Another form converts digital values into variable duty-cycle pulses. Such pulse-width-modulation (PWM) converters can directly drive electromechanical loads through switching amplifiers.

An important d/a converter property is its linearity. Linearity measures the converter's ability to produce the same analog output change for equivalent digital input changes. Thus, a digital transition from 1 to 10 and another from 41 to 50 should cause relatively the same effects on the output. For instance, both transitions should raise the output 18 mV.

Accuracy, a static comparison of input and output values, is also important in d/a conversion. Another concern, so-called a "glitch", is an undesired excursion of the output voltage when a change at the input is registered. Glitches can occur while the input goes from one value (switch configuration) to another. It is caused by the indeterminate nature of switches between states.

DSPs also offer other services to the system such as diagnostic monitoring. Diagnostic monitoring is achieved with FFT (Fast Fourier Transform) or spectrum analysis. By observing the frequency spectrum of mechanical vibrations, failure modes can be predicted and corrected in early stages.

Perhaps the most powerful DSP capability is adaptive control. The technique is possible because DSPs have the speed to concurrently monitor the system and control it. A dynamic-control algorithm adapts itself in real time to variations in system behavior. For instance, FFT data can be used to tune notch filters to track and eliminate vibrational modes as they vary with system speed, weight, balance, or other parameter.

DSPs in motion

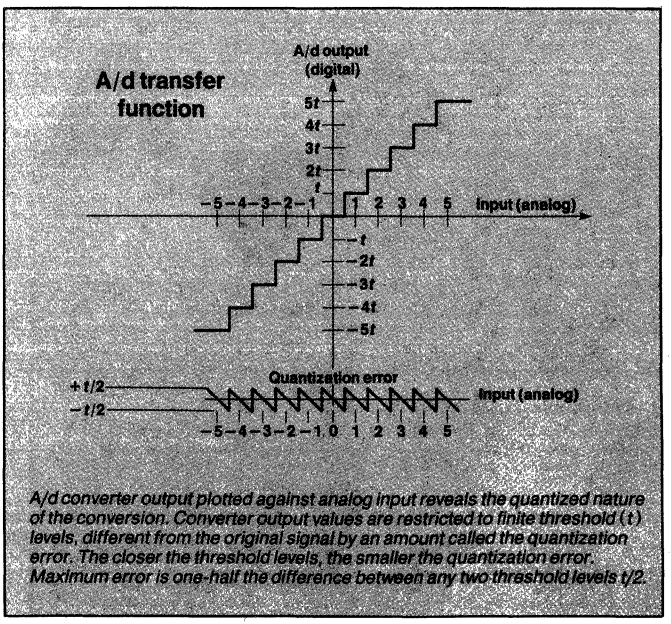
Digital signal processors were originally designed for audio/video applications such as speech coding and image recognition. But new applications in motion control demand hardware and software features not included on most commercially available DSPs.

A new breed of DSP microcontrollers, led by the TMS320C14, combines both signal-processing and system-management functions on a single IC. The signal-processing section samples inputs and runs control algorithms, while the system manager handles interrupts and schedules tasks, I/O, and other events that require interpretation.

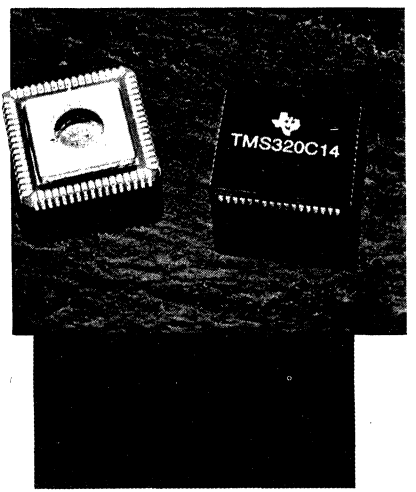
For a particular application, minimum processing speed is determined by the required sampling rate. Sampling rate depends on the bandwidth of the system under control. According to Nyquist's theory, an analog signal must be

Comparing motion controllers

Controller	Advantages	Disadvantages
Analog	High bandwidth	Aging and temperature variations
	High resolution	Hard-wired system complicates upgrade and modifications
	Real-time processing	Large component count
	Infinite sampling rate	Limited to single-variable control
Micro-processor	Insensitive to aging and temperature variations	Speed limited by single-bus architecture
	Software control makes modifications and upgrade easy	Lookup tables reduce precision and speed
	Single-chip solution is possible	Repetitive-addition multiplies reduce speed
	Limited multiaxis coordination	Low sampling rates reduce precision
Digitial signal processor	Insensitive to aging and temperature variations	Requires expert knowledge of system
	Dual-bus (Harvard) architecture boosts speed	Currently high costs (will decline)
	Software control simplifies modifications and upgrade	No compiler for TMS320C14
	High sampling rates improve precision	
	Single-chip solution is possible	
	Handles multiaxis systems	
	Implements complex algorithms such as adaptive control	
	Provides special filtering impossible with other techniques	



A/d converter output plotted against analog input reveals the quantized nature of the conversion. Converter output values are restricted to finite threshold (t) levels, different from the original signal by an amount called the quantization error. The closer the threshold levels, the smaller the quantization error. Maximum error is one-half the difference between any two threshold levels t/2.



sampled at more than twice the frequency of its highest frequency component. In practice, however, controllers typically sample at rates six to ten times above the highest frequency.

All processing must be completed between sampling periods. A controller with a sampling rate of 10 kHz, for instance, has 100 μs to sample the input and calculate the output. In many cases, multiply and accumulate procedures account for the majority of calculations. Some algorithms may call up to 50 multiplies per sample. Thus, high-speed multiply and accumulate hardware is necessary for DSP controllers.

Such hardware is available in the 320C14. In one instruction cycle, it multiplies two 16-bit numbers and stores the result in a 32-bit accumulator. Instruction cycle time for the 25-MHz processor is only 160 ns, for a throughput of 6.4 Mips (million instructions per second).

It is important that the product of two 16-bit numbers be stored in a 32-bit accumulator. But not all 16-bit processors have 32-bit accumulators. If only a 16-bit register is available, 16 bits are lost with each multiply. Truncation of this sort reduces precision and may show up as random fluctuations or noise in the system variables. Attempts to control noise often degrade system operation.

Processing capability is also a function of internal data format. For instance, floating-point processors are suited for applications with wide dynamic range because their data registers contain large exponential fields. This type of data representation frees designers from concerns over signal magnitude. The drawback with floating-point processors, however, is their high price.

Fixed-point processors, on the other hand, cost much less. They also provide greater accuracy because their data registers contain larger mantissa fields. The tradeoff is a lower dynamic range. Dynamic range can be expanded by doing floating-point calculations in software. But this approach reduces speed. For example, the 320C14 executes 16-bit floating-point multiplies in 6.5 μs.

Overflow protection is another

DSP BUILDING BLOCKS

The building blocks of analog control systems are operational amplifiers, while in digital signal-processing (DSP) systems they are multipliers. Multipliers are key hardware for executing digital filters and Fast Fourier Transforms (FFTs) in software. Originally, multiplier ICs were available only in individual packages. Now, they are integrated into most DSPs chips.

Digital filters are capable of higher speeds and sharper cutoffs than analog filters. In addition, they provide better stability with less drift. They also require no adjustments and can have nearly an unlimited signal-to-noise ratio (SNR). The SNR of a digital filter is proportional to its analog-to-digital (a/d) resolution.

Digital filters are usually based on a linear constant-coefficient difference equation such as

$$y(n) = \sum_{k=0}^{N} a_k y(n-k) + \sum_{k=0}^{M} b_k x(n-k) \qquad (1)$$

where, $x(n)$ = filter input sequence, $y(n)$ = filter output sequence, a_k = output coefficients, and b_k = input coefficient.

When the input coefficients are all zero, equation 1 reduces to

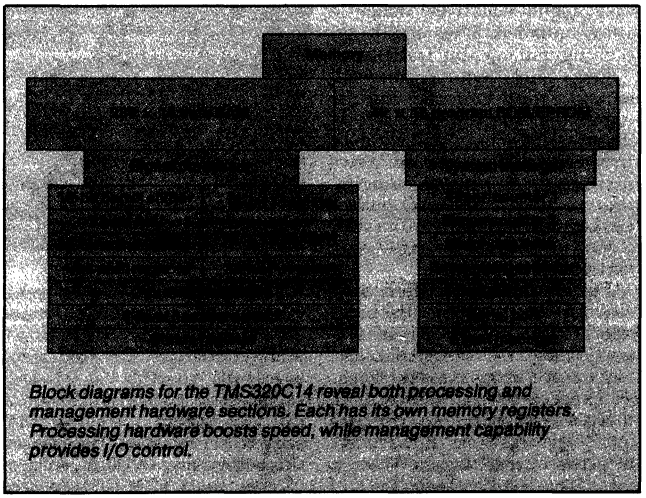
$$y(n) = \sum_{k=0}^{M} b_k x(n-k) \qquad (2)$$

This is called a finite impulse response (FIR) filter of length $M+1$. Such a filter consists of a tapped delay line with a series of M digitally summed multiplies. It has no feedback, making it unconditionally stable.

Another digital filter type, the infinite impulse response (IIR) filter, is defined when at least one a_k term is nonzero. An IIR filter has both feedforward and feedback terms like some op-amp-based analog filters. It is simpler than the FIR in terms of hardware and software. But it is also potentially unstable and susceptible to offsets and nonlinear response.

Multipliers and accumulators also play an important role in implementing FFTs. The FFT is a hardware-efficient version of the Fourier transform. It decomposes a time function into its frequency components, providing frequency analysis of the signal.

DSP system analysis is simplified by techniques such as the z transform. The z transform does for sampled-data systems what the Laplace transform does for continuous-data systems: it describes system output for a specified transfer function and input. Like the Laplace transform, the z transform permits algebraic techniques instead of differential equations.



Block diagrams for the TMS320C14 reveal both processing and management hardware sections. Each has its own memory registers. Processing hardware boosts speed, while management capability provides I/O control.

FLOATING VS. FIXED POINT

A DSP control system's word size determines resolution and dynamic range. With fixed-point processors, there is a linear relationship between word size and dynamic range. Each additional bit adds a higher power of the base 2. Resolution in such processors is fixed by the value of the least-significant bit (LSB).

In contrast, floating-point processors increase their dynamic range much more drastically with word size. Each additional bit allows the exponential term to grow by a factor of two. For a given word size, however, floating-point resolution is not as good as fixed-point because less digits are assigned to the mantissa. But in most DSP systems, dynamic range is usually more important than resolution.

How the TMS 340C14 stacks up

Feature	320C14	80C196	68000
Frequency (MHz)	25.6	12.0	10.0
Instruction cycle (ns)	160	340	400
Throughput (Mips)	6.4	1.0	0.6
Multiply (16 × 16) (μs)	0.16	2.4	7.0
PID loop (μs)	2.2	27.0	25.0
Matrix multiply (μs) (3 × 3) (1 × 3)	4.3	23.3	65.2

TMS320C14 features

Signal processor/system manager requirement	TMS320C14
Speed > 5 Mips	6.4 MIPS
Single-cycle instructions	Yes
Multiply time < 200 ns	160 ns
Word length > 32 bits	32 bits
Multiplication registers >32 bits	32 bits
Scalers (Shifters)	Yes
Overflow protection	Yes
Servo update time < 50μs (i.e. sampling rate > 20 kHz)	Yes
Control-oriented instruction set	Yes
On-chip memory	256 RAM, 4k ROM
Bit I/O	16 bits
Timers/counters	4
Serial communications	Yes
Hardware interrupts	15 levels
Software development support	Yes
Hardware development supports	Yes
Floating-point arithmetic	No

concern. Control algorithms, with many successive multiply and addition operations, can easily overflow registers. In an overflow, data registers on many processors recycle from their most positive to their most negative number. But polarity changes at the output of a motor controller, for instance, can reverse motor direction. Accumulation registers on the 320C14, on the other hand, latch at the most negative or most positive value. This feature eliminates the need to protect against polarity (motor) reversal.

To function as system manager, DSPs must have on-chip I/O and other peripherals. For starters, the 320C14 has 16 bit-selectable digital I/O lines that can be configured in any combination of inputs and outputs. The I/O lines can be used, for example, to scan keyboards or drive external devices. A special input feature sets an interrupt flag when inputs collectively match a stored number. This facilitates counting or timing applications.

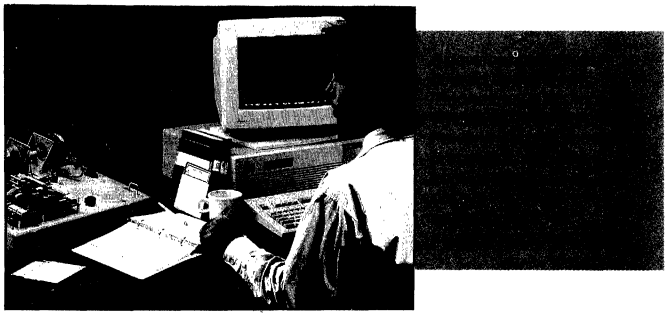
The IC also features an event manager that controls capture (input) and compare (output) subsystems. The capture section is equipped with hardware optimized for timing applications. For instance, encoder feedback pulses can be timed with up to 160-ns resolution to provide accurate position and speed data.

The compare subsystem is basically the chip's output. Its hardware is optimized for driving motion systems. One operating mode, for instance, allows it to function as a 6-channel PWM controller with up to 40-ns resolution. Six compare (CMP) registers work in harmony with two internal timers. When a match is detected between the CMP register and its specified timer, the event manager changes the state of the CMP output pin. Two internal interrupts are also generated.

Other on-chip peripherals include an array of 16-bit timers. Two timer/counters are intended for clocking external events and serving the event manager. Another, the watchdog timer, prevents internal software hang-ups. When the watchdog times out, a maskable interrupt is set and a pulse is generated on an output pin. The pulse may reset external hardware or the processor.

Development systems

Although DSP control systems offer numerous benefits, designing them can be difficult. For the most part, familiar analog design tools such as breadboards and scopes offer little help. And because compilers are not yet available for the 320C14, code must be written in assembly language. Limited stack space with room for only 4-level calling poses another challenge. A call is a branch statement that jumps to a subroutine. Because few subroutines may be called, sections of code must often be repeated throughout programs.



Despite these obstacles, makers of development systems for the 320C14 have found ways to simplify the design process. For one, their products compensate hardware shortcomings by supplementing memory space. Secondly, they allow engineers to prototype DSP control systems on PC plat-

forms, using methods similar to those for microprocessors or microcontrollers. Teknic Inc., for example, makes a development board called the Power-14.

Power-14 is designed for motion control applications. On-board switching servoamplifiers deliver a total of 750 W for driving various types of motors, linear actuators, and proportional valves. At the input, eight channels of 12-bit analog-to-digital conversion accept tachometer, potentiometer, and other sensor signals. The 20-μs converter provides extremely high (0.02%) resolution. In many applications, 0.4% resolution from 8-bit conversion is sufficient.

The Power-14 board adapts to a wide range of motion-control applications because its I/O sections can be configured by the user. Configuring is done in software. On-board configuration logic interfaces the I/O hardware with the 320C14's event manager (CAP and CMP lines). CMP lines connect to servoamplifiers, while encoder inputs are fed to CAP lines.

Four input modes accommodate a variety of feedback schemes. Input logic, in addition to routing signals, converts quadrature encoder feedback into count-up and count-down pulses. It also includes edge-detection circuits for index signals. The extra hardware reduces software overhead and improves system speed.

Likewise, there are four output modes. Output logic controls drivers independently or in pairs through nonoverlap hardware. The board can be adapted for brush or brushless dc motors, single or three-phase ac drives, stepper motors, variable reluctance motors, and other ac or dc loads.

Output hardware consists of six power transistors. Each has a no-load current-sensing device. Current feedback signals (12 bit) warn of overcurrent conditions and provide information such as torque or force to control algorithms. In dc brush motors, for instance, torque is directly proportional to the amount of current into the windings. But torque calculations for dc brushless motors require an additional term equal to the angle between the windings and the rotor.

High-speed MOS transistors form three half-bridge (totem-pole) amplifiers. Switching speed is determined by the number of bits in the output command. Up to 16 bits of digital-to-analog resolution are possible. A larger number of bits gives higher resolution, but slower amplifier switching rates. For example, 10 bits of resolution, sufficient for most systems, are available at a 25-kHz switching rate. Adding only one bit halves switching frequency to about 12 kHz.

Another Power-14 feature is a 40-pin connector that makes the 320C14's digital I/O directly available. Configuration logic and on-board peripherals are bypassed. This allows designers to develop custom nonmotion applications. Off-board connections, for instance, can be made to CODECs and external clock sources. CODECs code and decode signals for pulse-code-modulation (PCM) transmissions. On-board connections can be made to capture, compare, and digital I/O lines.

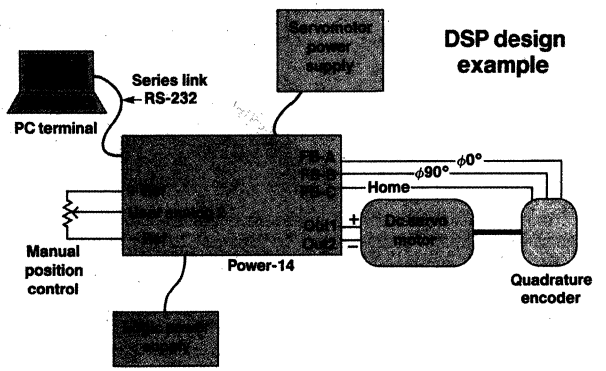
DSPs AT WORK

An example servo/indexer system configuration demonstrates the Power-14 evaluation board. The system employs a brush-type dc motor, two-phase optical encoder, dc supply with regulated 5-V and ±12-V signals, and servomotor power supply. A PC running a terminal emulator program provides system control from the keyboard. A potentiometer is used for manual position control. Also required is a 3201X assembler/linker.

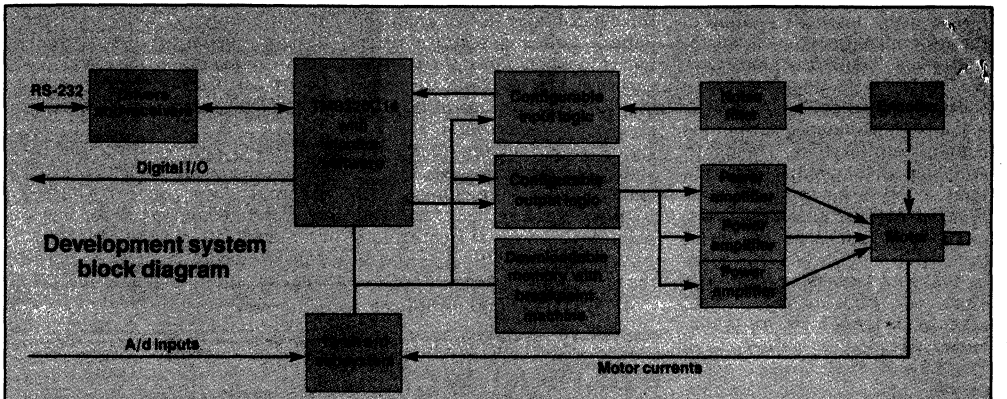
The software package provides full control over the system, allowing users to vary PID coefficients and observe the effect on system operation. The potentiometer can be used to adjust servo position. Encoder feedback position may be displayed to reveal steady-state error.

Users may also generate custom code. Developers begin by writing program modules in 320C14 assembly language on the monitor/debugger. Programs may be tested with various emulator and simulator software. To be applied to the live system, the code must be assembled and linked into a TI TAG file. A communications program such as Crosstalk or Procom downloads the code to the board. The monitor provides utilities to run and debug the program.

Development of 320C14 applications can be accelerated with Teknic's Power-Source software package. The modular library of calls and routines makes code writing faster and easier. It can be used, for instance, as the basis of a custom DSP control design. The PID loop can be taken out and replaced with the user's algorithm. Supporting commands that initialize and run the chip may not need to be modified.



A TMS320C14 indexing servosystem can be configured with the Power-14 board and controlled by the demonstration software. Encoder phase signals are fed through up/ down counters (input mode 2), while a PWM drive (output mode 2) powers the motor.



Development system block diagram

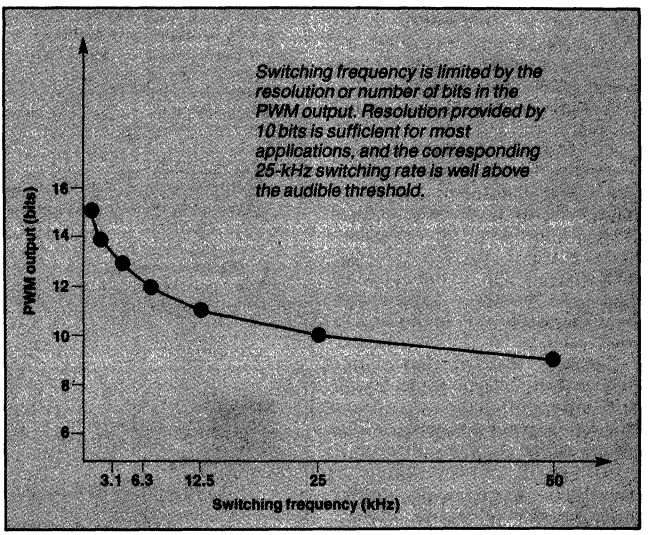
The Power-14 evaluation board includes input and output configuration logic, three-stage amplifier, 12-bit a/d converter, and an RS-232 port. In a typical motion application, the board allows the TMS320C14 to drive a motor using encoder feedback.

A small on-board prototyping area provides a place to fabricate special signal-conditioning circuits. For instance, encoder inputs may be fed through decoder/prescalers to reduce software overhead. Sensor signals may be amplified, filtered, or isolated. Outputs may likewise be modified with digital-to-analog or frequency-to-voltage converters.

Other hardware on the Power-14 includes a TMS320E14 processor and 8k words of on-board downloadable program memory. The 320E14 is the EPROM version of the 320C14. Suppression circuits are added to reduce conducted electromagnetic interference (EMI) from the amplifiers. An RS-232 serial port is also available. The port allows the board to communicate with the development platform.

Users can generate code, test software, and control motion systems from the host's monitor/debugger screen. Typically, the host is a PC running a terminal emulator program. A command-driven monitor interface provides access to all debugging facilities. Code is entered on screen, assembled, and downloaded through the RS-232 port for execution. A demonstration program based on a simple PID algorithm allows users to experiment with DSP control systems.

The demonstration/test program is supplied on a 5¼-in. floppy disk. It is specifically written for a DSP control system consisting of a dc motor and an optical encoder. From the monitor, users adjust PWM rates and duty cycles, dump current analog-to-digital converter values, display and set I/O ports and memory contents, and vary PID coefficients.



Users may also develop their own code. The monitor/debugger allows them to download code, set breakpoints, and display and modify memory register contents. Breakpoints allow users to see exactly what is going on (register contents) at specific points in the program. If software is causing a problem or hanging up during a certain task, a breakpoint can be used to obtain a snapshot of the register states at the point of interest.

Breakpoints also allow users to develop code in stages. One part is written and tested, while the remainder of the program is artificially simulated with values plugged into registers. External hardware provides one breakpoint, while multiple software breakpoints are possible. ■

USING DIGITAL SIGNAL PROCESSORS FOR CONTROL

Herbert Hanselmann

University of Paderborn, Department of Automatic Control in Mechanical Engineering
4970 Paderborn, Federal Republic of Germany

ABSTRACT

Digital single-chip signal processors solve speed problems arising with the implementation of measurement and control algorithms. After a discussion of processing power and applications an outline is given of an advanced CAE support system for the implementation of complex control and related systems.

1. INTRODUCTION

Digital single-chip signal processors (DSP) are very attractive means for the implementation of measurement and control algorithms, mainly because of their computing speed, which is more than an order of magnitude higher than with fast modern 16/32 bit microprocessors or microcontrollers, using fixed point arithmetic (Hanselmann, 1986a, 1986b, 1987).

A list of present devices that seem to be useful for control implementation and are available to the public is given in Tab. I.

DSP make implementation of nontrivial controllers with high sampling rate feasible at reasonable cost; the TMS 32010 in particular has already been used in many control applications, as described for example by Slivinski and Borninski (1985), Kanade and Schmitz (1985), Hanselmann (1986b).

In the following some speed benchmarks are presented, then some applications are discussed, followed by a brief discussion of DSP limitations and how they will develop with future DSP. The last sections are on Computer Aided Control Engineering (CACE) for the implementation of fast and complex control systems, particularly on DSP.

2. SPEED BENCHMARKS

Even older DSP deliver impressive speed in measurement and control applications as shown by the following benchmark data for the Texas Instruments TMS 32010: infinite-impulse response filter biquad section 2.2 μs, finite-impulse response filter 0.4 μs per tap, complex 64 point FFT 0.6 ms, 1024 points 43 ms (Burrus and Parks, 1985), table-lookup with linear interpolation 8 μs, generation of maximum sequence PRBS noise out of a 32 bit register 5.4 μs per clock, sine function generation 6.6 μs per point (Mehrgardt, 1984), 9th order controller at 31 kHz sampling rate, including overflow management code, 15th order multivariable controller with 13 inputs and 3 outputs, with some nonlinearities, at 10 kHz sampling rate.

Some future DSP from Tab. 1 even promise to be significantly faster. The Motorola 56000 for instance will be about 4 times faster with IIR or FIR filters due to shorter cycle time, and almost 10 times faster in the 1024 point FFT application where the TMS 32010 is slowed down due to RAM limitations.

For the 9th order controller mentioned above a speed comparison has been made against 16/32 bit microprocessors. This single input single output controller arose in an industrial application with a very fast electromechanical positioning system (Hanselmann, 1986b; Hanselmann and Moritz, 1986). Since with general microprocessors the multiply operation mainly determines the execution time, an upper bound for the achievable sampling rate can be given based only on the total number of multiplications. This upper bound

signal processors		type	available
NEC	μPD 7720	U	1982
Texas Instr.	TMS 32010	U	1983
Fujitsu	MB 8764	U	1984
STC	DSP 128	U	1985
Texas Instr.	TMS 32020	U	1985
Texas Instr.	TMS 320C25	U	
Nat. Semi.	LM 32900	U C	
Analog Dev.	ADSP 2100	U C	
Phillips	PCB 5011	U	
Thomson	TS 68930	U	
Motorola	DSP 56000	U	
Nat. Semi.	LM 628	A	
NEC	μPD 77220	U	
NEC	μPD 77230	U F	

U universal
C processor core (external memory)
A algorithm-specific
F floating-point arithmetic

Tab. 1 present and future DSP

is given in the rightmost column of Tab. II. The controller had 33 nonzero and non-one coefficients, i.e. 33 16 x 16 bit multiply operations had to be performed per sampling interval. Since there are also additions and data transfer operations to be performed the sampling frequency actually achievable would be somewhat lower. A comparison of the estimate with actual experimental results was carried out on a filter (from Phillips and Nagle, 1984), and on the controller which Table II is based on. The target was a 68000 system running at 10 MHz, programmed in assembly language. Actual sampling rates turned out to be about 50% of the upper bound estimate in the filter case, where subroutines and loops were used, and about 70% in the controller case with fast subroutine- and loop-less code.

microprocessor	clock	f_s
8086	8 MHz	< 2 kHz
Z8000	5 MHz	< 2 kHz
68000	10 MHz	< 4 kHz
32016	10 MHz	< 5 kHz
TMS32010 signal processor		31 kHz

Tab. 2 achievable sampling frequencies

The same controller was also implemented on a TMS 32010 signal processor and ran at 31 kHz sampling frequency, with overflow management code included for the control variable output computation. Thus the signal processor is an order of magnitude faster. The main reason for this is that with the microprocessors the fixed-point multiplications are too time-consuming, a typical execution time being 6 μs for a 10 MHz 32016 processor (operands in memory). Due to hardware multipliers and efficient routing of operands and results through various buses the execution times of add / subtract as well as multiply operations of DSP are in the range of 100 ns to 300 ns. Multiplication is no longer the most time-consuming operation.

3. APPLICATIONS

Typical control applications of DSP are found in the field of controlling fast mechanical devices, using fast servohydraulic or electromechanical actuators. This is because the required control bandwidth can well be from 100 Hz up to several kHz, and sampling frequencies considerably higher than this are necessary.

Furthermore it is precisely with the control of mechanical devices that detailed models can and should be obtained, with many degrees of freedom and high system order. So the controllers frequently are of higher order too, particularly when standard techniques such as LQG control (i.e. state variable control with Kalman-filtering) are applied.

But even with more classical control structures the control algorithms frequently have to go beyond simple PID-type control. A good example is the platform control system described by Slivinski and Borninski (1985), where a bulk of structural notch filters pushes the total controller order up to 19. Structural notch filters are used to cope with resonances in the mechanical structure by making them approximately "invisible" in the control loop. In the magnetic disc drive head positioning application described by Hanselmann (1986b) and Hanselmann and Moritz (1986) this approach has also been used (one of the controllers studied was that mentioned in section 2 and Tab II). The computational power required is particularly high there because of very high control bandwidth.

We use an experimental lab system with the TMS 32010 for implementation of such controllers. It is also used by some R&D departments in industry. This system accommodates up to 15 inputs and outputs each (analog or digital) and is equipped with a Z80 single board computer for host-target communication (RS 232), sampling-rate programming, TMS program download, and program storage in nonvolatile RAM. Along with appropriate CACE software (see section 5) this experimental system forms a very powerful tool for control system realization and evaluation.

Examples of applications apart from the magnetic disc drive are the active or semiactive suspension of vehicles and multiaxis robot control.

Vehicle suspension using a fully active hydraulic actuator instead of the passive spring / damper system has been realized for a single wheel test bed using the Intel 2920 DSP (Lückel and Kasper, 1984; Kasper, 1985), and will be realized for a Volkswagen Golf car in the future using the TMS system. A semiactive suspension (single wheel in the first stage) is under study in an industrial company using our TMS system. A 4 wheel suspension is expected to require controllers of order 10 ... 20, with more than 10 inputs from sensors, and 4 outputs to the actuators. Controllers are to a large extent linear, but there are some nonlinear compensations of nonlinear plant behaviour to perform. The sampling frequencies will range from some hundred Hz up to about 5 kHz, the higher sampling frequencies being required for the fast hydraulic subsystems.

The objective of the robot control application is damping and stiffening of an elastic robot by control (Moritz *et al.*, 1985; Moritz and Henrichfreise, 1986). A three axis robot with electric servomotors, Harmonic Drive gearboxes, and light aluminum arms has recently been successfully controlled by a multivariable controller using the TMS system. Oscillations visible by the naked eye when conventional cascade control was used were completely damped in all relevant degrees of freedom with the multivariable controller. Without the feedforward inputs there were 4 inputs from strain gage sensors, 3 inputs from angle encoders, 3 inputs from tachogenerators, 3 angle reference inputs to the controller, and 3 outputs to the motors. The order of this multivariable controller was only 6 in this first development stage, and the sampling frequency of 23 kHz was higher than required. In the next stage the controller will be augmented by friction observers and compensators, and the workload for the DSP will increase.

4. LIMITATIONS

The computing speed of DSP is impressive, but there are also several limitations.

Compared to microcontrollers such as the Intel 8096 or the NEC 78312 a DSP system usually requires far more hardware surrounding the processor chip. These microcontrollers include sophisticated i/o function blocks, right up to AD-converters, decoders for incremental angle sensors, and serial communication circuitry, whereas current DSP are only computing machines (with the exception of so-called algorithm-specific DSP such as the one listed in Tab. I, which is however very special purpose). DSP often also require very fast static RAM for program and data storage.

Another drawback with some DSP is their limited addressing capability, which is most severe with data RAM. The NEC 7720 and TMS 32010 have 128 and 144 16 bit words of on-chip data RAM, without the possibility of extending data RAM externally at full speed.

If the application requires service of interrupts from various sources, the next problem with DSP is encountered. Of the 'on-the-market' DSP from Tab. I, only the TMS 32020 allows for more than one interrupt source (i.e. 3 external plus some internal ones), whereas the MB 8764 and the DSP 128 have no interrupt mechanism at all. One reason for this may be that some hardware precautions are necessary when pipelined instruction execution is interrupted.

A common restriction with all present DSP is that they are only fast with fixed-point arithmetic, see for instance Blasco (1983) for the TMS 32010 and Crowell (1985) for the TMS 32020. Because standard operand wordlength is 16 bit, and accumulation (think of scalar product computations) is in most cases performed with extended precision (up to 35 bit) at no extra cost, the accuracy and dynamic range will usually be sufficient for control purposes, provided the control algorithm has been prepared appropriately (see section 5). The desire to have floating-point arithmetic is often caused by lack of know-how and tools for precisely this preparation of a controller for fixed-point implementation.

A last drawback to be mentioned is due to lack of programming support. With the exception of the TMS 32010 (see section 6) only assembly language programming is supported commercially. For runtime efficiency most users also tend towards assembly level programming. However, because of 'exotic' architectures and instruction sets compared to general microprocessors, programming easily gets tedious and error-prone. This applies particularly to those DSP which have a 'microcode-like' instruction set, such as the NEC 7720, the MB 8764 and some of the announced DSP.

Additionally, memory restrictions may require tailored coding for every version of a controller, and efficient code constructs may be dependent on the actual numerical values of operands, leading to frequent reprogramming when a controller is in its development stage where numerical values are not yet fixed. For an example see the TMS 32010 code of Fig. 1. It checks the result of a downscaled scalar product computation (Hanselmann, 1986b, 1987)

$$r = c_s^T x \qquad (1)$$

(where all coefficients in c_s have been downscaled from the original coefficient vector by a common factor 2^{ν} to fit them into the fractional number range) whether the rescaled true result is overflowing, in which case saturation is performed. Version a) is valid for all reasonable ν, whereas the much faster version b) is only valid for two values of ν because of restrictions of the processor. And for some other values of ν there is even another optimal version (not shown) in between. Fig. 1 also shows another problem of DSP: quite complicated

```
NUE       EQU ...    ; scale-factor
ALL1      EQU 1111111111111111B
ALL1MSB0  EQU 0111111111111111B
MAX       EQU  32767D
MIN       EQU -32768D
            .
            .
;downscaled result in accumulator
          SACH HI,0       ; save acc
          SACL LO
          BGEZ POS
;negative value
          LAC HI, NUE+2
          SACH Z,0
          ZALS Z
          XOR ALL1
          BZ NOOF
  ;saturate
          ZALS MIN
          SACL RESULT
          B ENDOF
;positive value
POS:      LAC HI, NUE+2
          SACH Z,0
          ZALS Z
          BZ NOOF
  ;saturate
          ZALS MAX
          SACL RESULT
          B ENDOF
;no overflow
NOOF:     LAC LO,15
          SACH LO,0
          ZALS LO
          AND ALL1MSB0
          SACL LO
          LAC HI, NUE+1
          SACL HI
          ZALH HI
          ADD LO, NUE+2
          SACH RESULT
ENDOF:
```

a)

```
NUE       EQU ...    ; scale-factor
MAX       EQU  32767D
MIN       EQU -32768D
            .
            .
;downscaled result in accumulator
          SACH RESULT, NUE+1
          BLZ NEG
;positive value
          SUB MAX, 15-NUE
          BLEZ ENDOF
  ;saturate
          LAC MAX, 15-NUE
          SACH RESULT, NUE+1
          B ENDOF
;negative value
NEG:      SUB MIN, 15-NUE
          BGEZ ENDOF
  ;saturate
          LAC MIN, 15-NUE
          SACH RESULT, NUE+1
ENDOF:
```

b) (ν = 0 or 3 only)

Fig. 1 overflow-handling code (TMS 32010)

run-time and memory consuming code constructs may be required to do rather simple things, quite unlike the computing of scalar products which DSP are designed for.

Some of the problems discussed will disappear with some future DSP, which will not only be faster, but also allow for more program and data memory, incorporate more hardware for miscellaneous tasks (timers, communication ports), and have more flexible instruction sets. The dynamic range of fixed-point arithmetic will also be extended (although rarely really needed in control applications), by longer accumulators and in some cases (Motorola 56000 and NEC 77220) by a larger basic operand wordlength of 24 bit. Floating-point arithmetic DSP are also appearing. There is already one (proprietary) DSP at Bell Labs performing full 32 bit floating-point arithmetic at 150 ns per operation, and one DSP for the public (by NEC) has been announced (see Tab. I). It can also be expected that more programming tools such as general or special purpose language compilers will emerge.

5. CACE-TOOLS

Efficient use of DSP for control implementation requires some CACE-tools to assist in the preparation of the controller before programming it, and also it is desirable to circumvent processor specific assembly level programming.

In the pre-programming phase there are decisions to be made and checks to be performed which are mainly related to discretization, quantization, and timing. All this is not specific to DSP used in control, but would also apply to application of general microprocessors or microcontrollers. Only the peculiarities and problems of fixed-point arithmetic become irrelevant when floating-point arithmetic can be used with microprocessors or microcontrollers.

The CACE-tools we developed and still use comprise software-modules which perform, or at least assist, in performing the following tasks:

- discretization of continuous designs via a selection of methods,
- choice of realization structures for multivariable systems with respect to finite wordlength restrictions,
- scaling for fixed point (fractional) arithmetic, scale factors supplied by user from for example simulation, or found automatically,
- checking for differences in frequency or for example step response due to discretization and due to fixed point coefficient representation,
- checking for effects of AD- or DA- signal quantization, arithmetic, overflow, and nonsimultaneous sampling by nonlinear control system simulation,
- automatic code generation (formerly for Intel 2920, now for TMS 32010) from a description of a linear controller in state-space, plus optional nonlinear extensions.

In the early stages of design only a few assumptions such as on the future AD-converter resolution and a rough estimate of sampling rate may be involved In later stages discretization and timing effects are taken into account first, then an accurate abstract model of the target DSP program , already involving finite wordlength effects, comes into play. It depends on the user's experience and on the controller whether the CACE-tools for implementation have to be used to the full. It is not uncommon for only discretization, selection of a standard realization structure, and automatic scaling to have to be performed. A more detailed discussion of these steps to be taken in the pre-programming phase can be found in Hanselmann (1987).

The last step, i.e. programming, is performed fully automatically for the linear part of a controller by an automatic code generator (Fig. 2) for the TMS 32010 (Loges, 1985). The controller is assumed to have been translated into a single state space difference equation of the form

$$\begin{aligned} x_{k+1} &= A\,x_k + B\,u_k + f_x(x_k,u_k,y_k,k), \\ y_k &= C\,x_k + D\,u_k + f_y(x_k,u_k,k) \end{aligned} \qquad (2)$$

Code for the nonlinear parts is not generated but linked to the generated code. The code generator provides overflow management code, so-called scalar product scaling, and extended precision arithmetic on demand, and copes with the data RAM limitations of the TMS 32010 by gradually moving to memory saving code if necessary in a run-time optimal way. The code generator concept has also been used by workers in the general signal processing (filtering etc.) field, for references see Hanselmann (1987).

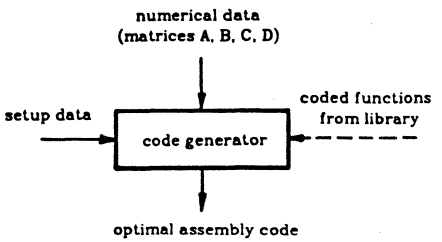


Fig. 2 automatic code generator

Experience shows that, using the abovementioned tools, in routine cases a control design can be brought to experimental evaluation in less than an hour, and virtually no knowledge of the target DSP is necessary for the control engineer who is only interested in getting his control system working. Our previous work has, however, been restricted to a certain class of controllers (single state-space description, single-rate, nonlinear terms supported but not integrated in the CACE-software data structure). More complex controllers now demand a more advanced concept.

6. FUTURE CACE-CONCEPT

The main restrictions of our previous CACE-tools have been: (i) assumption of the controller in the form of (2), (ii) separation of information belonging together logically, (iii) task dependency, (iv) target processor dependency in code generation tools.

We are going to remove these restrictions now by developing tools based on models of complex controllers and by layering the code generation procedure.

The goal is to close the gap between sophisticated control system design and realization of a designed controller by means of mostly automatic tools working on a model of the controller. This model may be for an analog version of the controller initially, and will subsequently be transformed step by step into a full digital controller model via the stages of sampling rate selection, discretization, structure selection, scaling etc..

Designing a modeling concept on which such implementation tools can be based is a nontrivial task for complex controllers. The usual collection of a few discrete state-space models or z-transfer-functions is far from sufficient to make up a model.

It should for instance be possible to represent complex controllers constructed from submodels in a hierarchical way. In the vehicle suspension application mentioned in section 3 there are controllers for the individual wheel hydraulics on a medium hierarchical level, the subsystems encapsulated in these controllers are on the lowest level, and on the highest hierarchical level is the total 4 wheel controller (including pitch and roll control etc.).

The same example also shows the need to account for multi-rate systems because of high sampling rates for servohydraulic control, and lower rates for car body attitude control.

It is also important to accommodate timing information, i.e. information about when input signals are sampled, when output signals are available and what the sequence of execution of subsystem algorithms is.

Information regarding data formats and arithmetic in the target processor should also be representable in a form sufficiently abstract to be processor independent, but close enough to the hardware and architecture of target processors for running a control system simulation for instance to yield 'real-world' results.

In order to manage all this information it is advisable to follow the lines of modern software engineering. The approach we are investigating is to define a model language which works in the user's technical terms as much as possible, and represents the information in a readable, consistent, and logical way. A model description given in this language is then to be used throughout the design and implementation process, up to simulation and final code generation. The conventional data structures such as collections of matrices are only a part of a model, possibly a small one. Even there, several distinctions must be made and types of controller submodels such as standard state space models, FIR-filters, FSVD-type state space models (Hanselmann, 1987) should be introduced.

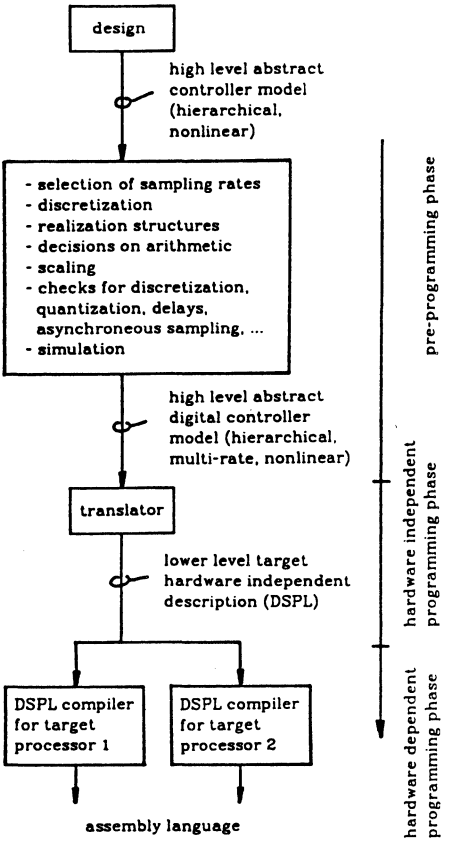


Fig. 3 future code generation

The final stage of code generation will now also be based on the controller model. Because of the possible complexity of such models, and in order to get more target processor independency, code generation will be performed in at least two stages (Fig. 3), with an intermediate control task representation in a specific medium level language (DSPL, digital system programming language) program which will be derived from the abstract controller model by means of a translator. We have designed a first version of DSPL (Hanselmann and Schwarte, 1987), and we expect to have a preliminary DSPL compiler for the TMS 32010 by the end of 1986.

We prefer to have a language which is tailored to signal processing and control tasks rather than using general purpose programming languages such as C or Pascal. Compilers for these two languages emerged recently for the TMS 32010 processor (Marrin, 1985). Our goal is to create an instrument which generates code as efficient (but more efficiently) as that which a human programmer would for the tasks we deal with, and to keep the target hardware and processor dependent parts as small as possible. This will be achieved by concentrating such dependencies into the compiler for the rather basic DSPL language, which can be modified for new processors (even custom-designed ones) with reasonable effort.

7. CONCLUSIONS

As shown by benchmarks and applications, digital signal processors are attractive for control implementation due to their computing speed. Compared to some other types of processors there are some limitations however, which will partly be removed with the new signal procesors expected in the near future. It has been stressed that efficient use of DSP for control implementation requires some CACE-tools to assist in the preparation of the controller before programming it, and in programming itself.

REFERENCES

Blasco, R. W. (1983). Floating-Point Digital Signal Processing Using a Fixed-Point Processor. Presented at Southcon; also in *Signal Processing Products and Technology*, Texas Instruments.

Burrus, C. S. and T. W. Parks (1985). *DFT/FFT and Convolution Algorithms*, Wiley & Sons.

Crowell, C. D. (1985). Floating-Point Arithmetic with the TMS 32020. *Texas Instruments Application Report.*

Hanselmann, H. (1986a). Einsatz Digitaler Ein-Chip-Signalprozessoren in der Mess- und Regelungstechnik. *Bulletin Schweizer Elektrotechnischer Verein*, 11, 632.

Hanselmann, H. (1986b). Digitale Ein-Chip-Signalprozessoren in der Mess- und Regelungstechnik. *5. Wiss. Konferenz "Anlagenautomatisierung"*, Leipzig, GDR.

Hanselmann, H. (1987). Implementation of digital controllers. *Automatica*, survey paper, accepted for publication.

Hanselmann, H. and A. Schwarte (1987). Generation of fast target processor code from high level controller descriptions. *To be published.*

Hanselmann, H. and W. Loges (1984). Implementation of very fast state-space controllers using digital signal processors. *Proc. 9th IFAC World Congress*, Pergamon Press, New York.

Hanselmann, H. and W. Moritz (1986). High bandwidth Control of the head positioning mechanism in a Winchester disc drive. *Proc. IECON'86*, Milwaukee, Wisconsin.

Kanade, T. and D. Schmitz (1985). Development of CMU Direct-Drive Arm II. *Proc. 1985 American Control Conference*, Boston, 703.

Kasper, R. (1985). Entwicklung und Erprobung eines instrumentellen Verfahrens zum Entwurf von Mehrgrössenregelungen. Doctoral dissertation, University of Paderborn, appeared in series VDI Fortschrittsberichte, series 8, vol. 90, VDI Verlag, Düsseldorf.

Loges, W. (1983). Schneller digitaler Regler mit Signalprozessor. *Elektronik*, 19, 51.

Loges, W. (1985). *Realisierung schneller digitaler Regler hoher Ordnung mit Signalprozessoren.* Doctoral dissertation, University of Paderborn, appeared in series VDI Fortschrittsberichte, series 8, vol. 88, VDI Verlag, Düsseldorf.

Lückel, J. and R. Kasper (1984). Mehrgrössenregelung, Entwurf und Realisierung moderner Mehrgrössenregelungen am Beispiel eines hydraulischen Fahrzeugprüfstands. *Maschinenbau*, 3, 13 and 4, 27.

Marrin, K. E. (1985). VLSI and software move DSP techniques into mainstream *Computer Design*, Sept. 15, 69.

Mehrgardt, S. (1984). 32-Bit-Prozessor erzeugt analoge Signale. *Elektronik*, 7, 77.

Moritz, W., H. Henrichfreise and H. Siemensmeyer (1985). A contribution to the control of elastic robots. *Proc. IFAC Symp. Robot Control*, Barcelona.

Moritz, W., and H. Henrichfreise (1986). Regelung eines elastischen Knickarm-Roboters. *Proc. Workshop "Steuerung und Regelung von Robotern" of VDI/VDE-Gesellschaft Mess- und Regelungstechnik*, Langen, Germany.

Phillips, C. L. and H. T. Nagle (1984). *Digital Control Systems Analysis and Design.* Prentice-Hall, Englewood-Cliffs.

Slivinski, Ch. and J. Borninski (1985). Control System Compensation and Implementation with the TMS32010. *Texas Instruments Application Report.*

PART II
Design of Digital Controllers

Designing Control Systems

The design of a control system involves two major steps: (1) the process or plant must be put into a mathematical form so that its behavior can be analyzed and evaluated (i.e., a plant model must be derived), and (2) an appropriate controller must be designed so that the plant gives the desired response under the influence of the control system. Designing a controller requires selecting an appropriate structure and specifying performance requirements from the control systems. This introduction gives a brief overview of discrete systems, tells how to model a plant and convert it into a discrete mathematical form, and describes how to design different types of controllers. Most of the following information can be found in those textbooks appearing within the **Reference** section. The articles that follow this introductory material describe several of the commercially-available CAD packages that may be used for designing and simulating either the controller or the entire control system.

Discrete Systems

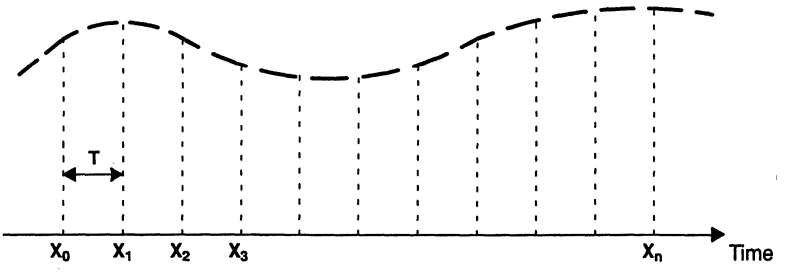
A system must be represented in its discrete form in order to be implemented on a DSP or a microprocessor. Discrete representation involves two elements. First, the signal is represented by its samples at discrete time intervals. These time intervals depend upon the sampling rate of the system. Second, the magnitude of the signal and its samples is also represented by discrete magnitude. The resolution of this magnitude depends upon the word length of the processing element. Here, only the sampling rate affects our treatment of this subject. However, in Part III's introduction, where we are concerned about the actual implementation, the effects of magnitude representation on a processor will greatly influence our treatment of that subject.

z-Transforms: In the continuous time domain, the system is represented with differential equations, and the analysis is carried out with Laplace transforms. Similarly, in the discrete time domain, a system is represented with difference equations, and the analysis is carried out with z-transforms. The z-transform of a signal is a representation of that signal as a sequence of samples as shown in Figure 1. Mathematically, it is given as a power series in z^{-n} with coefficients equal to the value of that signal or

$$X(z) = \mathbf{Z}(x(t)) = x_0 + x_1 z^{-1} + x_2 z^{-2} + \ldots + x_n z^{-n} \quad (1)$$

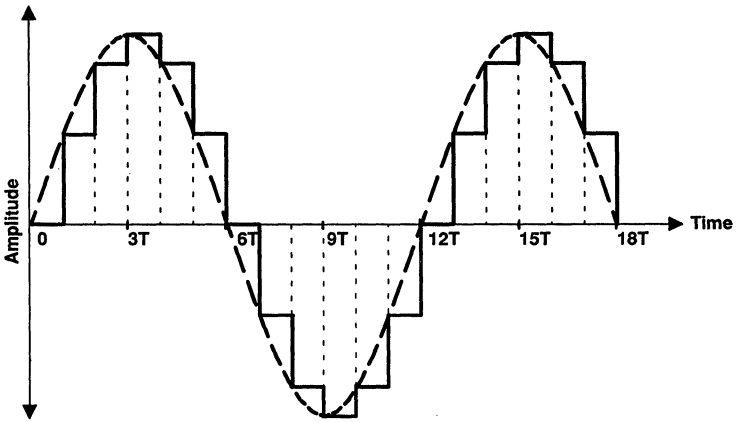
Z represents the z-transform; z^{-n} represents the delay of n samples, where n represents the position (0, 1, 2, $\cdots$, ∞) of time; x_0, x_1, x_2, $\cdots$, and x_n represent the magnitude of signal x(t) at that time.

Figure 1. z-Transform



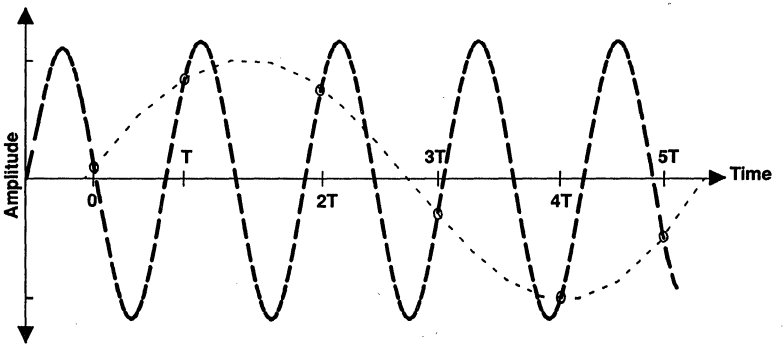
The z-transform represents the sampling process in a digital control systems. It converts a continuous signal to a discrete signal. The continuous signal can be recovered from the discrete signal as shown in Figure 2 by using a ZOH (zero order hold). The fact that both signals are equivalent allows us to do all our processing in the discrete time domain. Once the processing is complete, the signal can be converted back to continuous form.

Figure 2. A Continuous Signal Recovered from the Discrete Signal



One important consideration must be taken into account before the sampling is allowed to take place. According to Shannon's theorem, a signal must be sampled at a rate that is twice the highest frequency component of the signal. If this rule is not observed, the original signal cannot be recovered. Figure 3 shows a sine wave signal x(t) that is superimposed with a higher frequency sine wave. The higher frequency signal is giving the exact same samples as the signal x(t) and causing distortion. This effect is known as aliasing. To prevent this, low-pass filters known as antialiasing filters are used to filter out high-frequency components. Only the frequency of interest passes through. However, antialiasing filters should be used carefully in control systems because they introduce phase delay and affect the phase margins of the system.

Figure 3. A Sine Wave Signal



A general representation of any system in the z-domain can, by use of a transfer function, be given by the following equation where H(z) denotes the response of the system.

$$H(z) = \left[\frac{Y(z)}{X(z)}\right] = \left[\frac{b_0 + b_1z^{-1} + b_2z^{-2} + \ldots + b_nz^{-n}}{1 + a_1z^{-1} + a_2z^{-2} + \ldots + a_nz^{-n}}\right] \quad (2)$$

X(z) represents the z-transform of the input signal, Y(z) represents the z-transform of the output signal. $a_1 \cdots a_n$ and $b_0 \cdots b_n$ are coefficients that determine the response of the system. If both the denominator and numerator are factorized, the denominator represents the poles of the system and the numerator represents the zeros of the system. The output of the system is obtained by restating equation (2) as

$$Y(z) = -(a_1z^{-1} + a_2z^{-2} + \ldots + a_nz^{-n})[Y(z)] + (b_0 + b_1z^{-1} + b_2z^{-2} + \ldots + b_nz^{-n})[X(z)]$$

Since z^{-1} represents the delay of one sample time, the above equation can be restated in the time domain as a difference equation given by:

$$y(n) = -(a_1)[y(n-1)] - (a_2)[y(n-2)] + \ldots + b_0[x(n)] + b_1[x(n-1)] + \ldots \quad (3)$$

where y(n – 1), y(n – 2), x(n), and x(n – 1) represent samples of y(t) and x(t) at time intervals of n, n – 1, n – 2, etc.

Equation (3) is the standard form of representing systems in the discrete time domain, just as differential equations are the standard form of representing systems in the continuous time domain. Equation (3) also represents the standard form of implementation on a DSP.

In classical control, the analysis is frequently carried out with Laplace transforms. It is possible to convert directly from the s-domain to the z-domain. The relationship between s-domain and z-domain is given by the following equation:

$$z = e^{sT}$$

where T is the sampling period. However, in practice, several approximations are used to convert from one plane to another since an exact transformation is not possible. Table 1 shows the z-transform of some of the functions. Using these relationships, it is possible to carry out the analysis in the s-domain and transfer the results to the z-domain, or vice versa.

Table 1. z-Transform

FUNCTION	LAPLACE TRANSFORM	z-TRANSFORM
u(t)	$\frac{1}{s}$	$\frac{z}{(z-1)}$
t	$\frac{1}{s^2}$	$\frac{Tz}{(z-1)^2}$
e^{-at}	$\frac{1}{s+a}$	$\frac{z}{(z-e^{-aT})}$

Discretization Methods for Analog Systems: Different techniques can be used to convert continuous systems into discrete systems. However, a continuous system can only be approximated and can never be exactly equivalent. The conversion from the s-domain to the z-domain usually causes some distortion in the response and must be considered.

Step Invariant Method: This technique also known as ZOH (zero order hold) produces a discrete system whose step response is the same as the original continuous system at the sampling instants. It assumes that

the system is preceded by a ZOH (D/A converter) and followed by a sampler (A/D converter) so that both input and output of the resulting system are digital. Both the ZOH and sampler are included in the conversion scheme. The conversion is given by the following equation:

$$H(z) = (1 - z^{-1})\ Z\left[L^{-1}\frac{H(s)}{s}\right] \tag{4}$$

where Z represents the z-transform, and L^{-1} represents the inverse Laplace transform.

This transformation is usually what is required to convert a continuous plant to a discrete form; however, it gives unsatisfactory response with controllers and should be avoided when transforming continuous controllers. The ZOH introduces phase lag and distorts the frequency response of the controller. The Laplace transform can be split up by using partial fractions and z-transform tables.

Ramp Invariant Method: In this method, the step input described above is replaced by a ramp input, also called a first-order hold method. The ramp invariant conversion is given by the following equation:

$$H(z) = \left[\frac{(1 - z^{-1})^2}{Tz^{-1}}\right] Z\left[L^{-1}\frac{H(s)}{s^2}\right] \tag{5}$$

where T is the sampling period.

The ramp invariant usually gives good results and may be used when converting continuous controllers.

Matched Pole-Zero: In this technique, the poles of the s-domain are directly mapped into the z-domain according to the relationship $z = e^{Ts}$, where T is the sampling period. To equal the number of poles and zeros, additional zeros are added at $z = -1$. The gain of the two systems is matched at a critical frequency by choosing an arbitrary gain constant. This method does not take into consideration any aliasing effects.

Backward Difference: This technique replaces the derivative of a function by the difference between present and previous samples and is given by

$$\frac{dy}{dt} = \frac{y(n) - y(n-1)}{T}$$

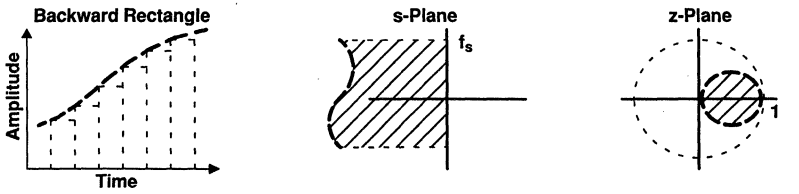
where T is the sampling period.

The transformation can also be done by using the following mapping:

$$s = \frac{1 - z^{-1}}{T}$$

This transformation maps the left half of the s-plane to a circle inside the unit circle of the z-plane. Hence, stable analog controllers also result in stable digital equivalents. In fact, some unstable analog systems give stable digital equivalents. The $j\omega$ axis in the s-plane does not map to the unit circle in the z-plane, thus degrading the frequency response. Using a higher sampling frequency gives a better approximation. Figure 4 shows the mapping from the s-plane to the z-plane for a backward difference approximation.

Figure 4. Mapping from s-Plane to z-Plane for Backward Difference Transformation



Bilinear Transformation: This technique, also called the Tustin transformation or the trapezoidal approximation, uses the relationship

$$s = \left(\frac{2}{T}\right)\left(\frac{z-1}{z+1}\right) \tag{6}$$

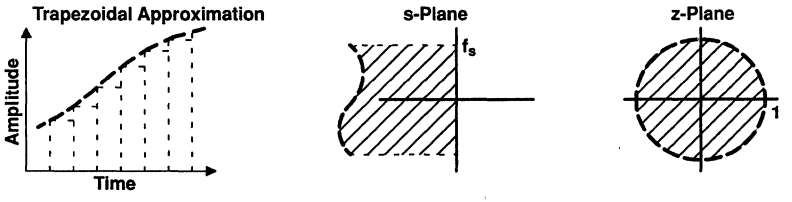
to transform an s-domain function into the z-domain. The left half of the s-plane band limited by the sampling frequency, f_s, is mapped into the unit circle in the z-plane. Thus, it is important to select as a high a sampling frequency as possible so that all poles are included. Although the frequency response of the continuous systems is replicated in the z-domain, it warps the frequency response at the critical frequencies of the system. To overcome the problem for systems like notch filters, the critical frequencies of the original s-domain are prewarped so that they end up in the z-domain system where they belong. The critical frequency ω_0 is prewarped to another frequency by the transformation,

$$\omega = \left(\frac{2}{T}\right)\left[\tan\left(\frac{\omega_0 T}{2}\right)\right]$$

where T is the sampling period.

This is the most commonly used method and always generates stable poles in the z-domain if the original s-plane poles are stable. Figure 5 shows the mapping from the s-plane to the z-domain for the bilinear transformation.

Figure 5. Bilinear Mapping from s-Plane to z-Plane for Bilinear Transformation



Other Methods: These are some other methods for transformation:

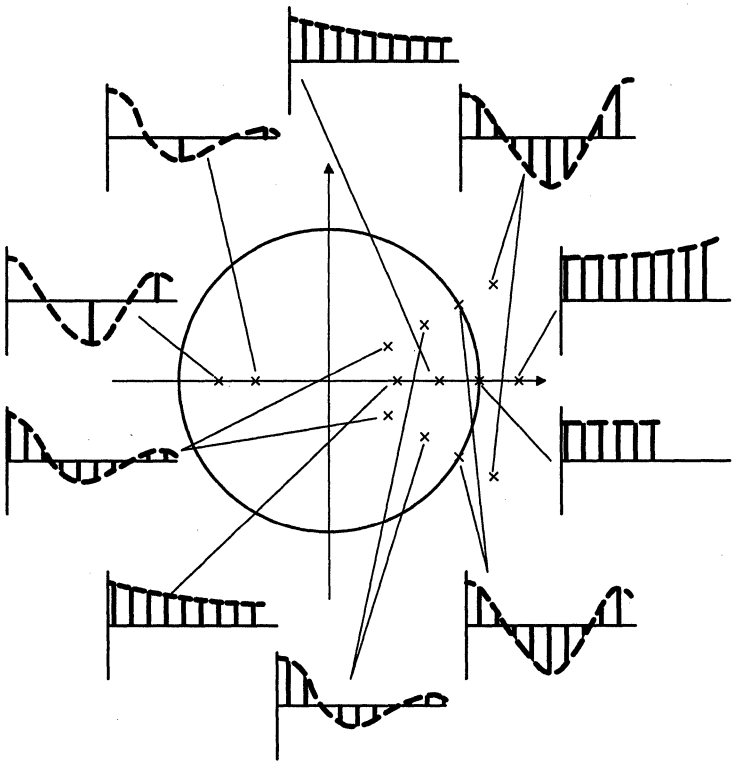
- Forward difference rectangular
- Matched pole-zero mapping
- Impulse invariant

They are less commonly used than the ones given previously and are not discussed here. However, different transformations result in different behavior and may be suitable for some structures.

Behavior of Poles in z-Domain: Conversion techniques change an existing analog design into a digital design. To ensure successful implementation of the control system design, some knowledge of the behavior of the poles in the z-domain is essential. As it is obvious from the mapping schemes above, the left half of the s-plane maps into the unit circle on the z-plane. This is the region of stability in the z-plane. Any poles (real or imaginary) located outside the unit circle are unstable and have an unbounded response. Poles located inside the unit circle give a stable response. Poles that lie on the unit circle provide oscillatory behavior. This corresponds to the $j\omega$ axis on the s-plane. As poles move toward the origin, their response decays at a faster rate. Zeroes may be located anywhere in the z-plane; however, as they move from the origin towards z = 1, they increase the overshoot of the system. If zeroes are located outside the unit circle, such a system is called a nonminimum phase system. Figure 6 shows the different pole locations and their

corresponding responses both inside and outside of the unit circle. One thing should be remembered; that unlike the s-plane, the mapping in z-plane is not unique. It is dependent upon the sampling frequency used for the discretization technique. A different sampling frequency gives a different mapping in the z-plane.

Figure 6. Response with Different Pole Locations in the z-Domain



Plant Modelling

The first part of designing any control system is to convert the plant into its mathematical form or to identify its parameters. The following example describes the derivation of a mathematical model for a plant.

A DC servo motor is used to represent the plant, and a model is developed for the motor. The motor is an analog device, and the given electrical and mechanical characteristics describe its behavior in the continuous time form. This model must be transferred into a discrete form or into the z-domain for use with a digital controller. The zero order hold method (ZOH) is used to transform the model into a discrete form.

In general, the electrical characteristics of a DC motor are given by

$$L\frac{di}{dt} + Ri = V - emf \tag{7}$$

where

L = inductance of motor

R = resistance

V = applied voltage

i = current

$\frac{di}{dt}$ = instantaneous current

emf = back emf = $K_e \dot{\theta}$

where K_e = emf constant

$\dot{\theta}$ = velocity

The mechanical characteristics are given by

$$J_m\ddot{\theta} + B\dot{\theta} + K\theta = T_L - J_L\ddot{\theta} \quad (8)$$

where

J_m = motor interia

θ = displacement

$\dot{\theta} = \frac{d\theta}{dt}$ = velocity

$\ddot{\theta} = \frac{d^2\theta}{dt^2}$ = acceleration

K = stiffness constant

B = damping constant

J_L = load inertia

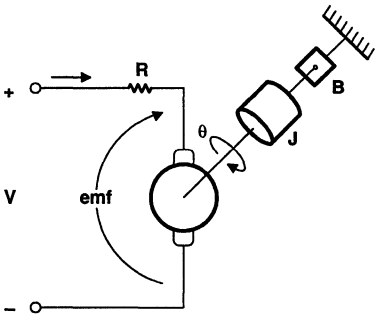
T_L = load torque = $K_t i$

K_t = torque constant

i = current

Figure 7 shows an equivalent electrical and mechanical model of the DC servo motor.

Figure 7. A Representation of a DC Servo Motor Model



The motor is a Pittman model 9412G316. It has the following parameters:

$R = 6.4$ ohm

$J_m = 1.54 \times 10^{-6}$ kg-m^2

$K_t = 0.0207$ N-m/A

$K_e = 0.0206$ volt/(rad/s)

The electrical time constant is given by $\frac{L}{R}$, and the mechanical time constant is given by $\frac{B}{J}$. In practice, $\frac{L}{R} << \frac{B}{J}$

Electrical steady-state conditions are reached quickly. Assuming steady-state current is reached, equation (7) is reduced to

$$Ri = V - emf = V - K_e\dot{\theta}$$

Combining (8) and the above equation results in

$$(J_m + J_L)\ddot{\theta} + B\dot{\theta} + K\theta = K_t\left(\frac{V - K_e\dot{\theta}}{R}\right)$$

Assuming both $J_m + J_L = J$ = system inertia and $K = 0$ = stiffness constant, the system equation becomes

$$\ddot{\theta} + \frac{1}{J}\left(B + \frac{K_t K_e}{R}\right)\dot{\theta} = \frac{1}{J}\left(\frac{K_t}{R}\right) V \qquad (9)$$

The Laplace transform of (9) is

$$(s^2 + as)[\theta(s)] = b[U(s)]$$

where

$$a = \frac{1}{J}\left(B + \frac{K_t K_e}{R}\right)$$

$$b = \frac{1}{J}\left(\frac{K_t}{R}\right)$$

If

$$U(s) = V(s)$$

then

$$\frac{\theta(s)}{V(s)} = \frac{b}{s(s+a)} \qquad (10)$$

Equation (10) is the final form of the transfer function of the motor in continuous form. This must be converted into a discrete form. The zero order hold (ZOH) transformation is used.

Zero order hold states that

$$G(z) = \left(1 - z^{-1}\right)\left\{Z\left[\frac{L^{-1}\, G(s)}{S}\right]\right\} \qquad (11)$$

Then,

$$\frac{G(s)}{s} = \frac{b}{s[s(s+a)]} = \frac{b}{s^2(s+a)}$$

Expanding as partial fractions, the above can be expressed as

$$\frac{G(s)}{s} = \frac{A1}{s} + \frac{A2}{s^2} + \frac{A3}{s+a}$$

Solving for A1, A2, and A3 gives

$$\frac{G(s)}{s} = \frac{\left(-\frac{b}{a^2}\right)}{s} + \frac{\left(\frac{b}{a}\right)}{s^2} + \frac{\left(\frac{b}{a^2}\right)}{s+a}$$

When multiplying by $(1 - z^{-1})$ and using tables to derive the z-transform,

$$G(z) = \frac{\left\{\frac{b}{a^2}[(e^{aT} - 1) + aT]z^{-1}\right\} + \left\{\frac{b}{a^2}[(1 - e^{aT}) - (aTe^{-aT})]z^{-2}\right\}}{1 - [(1 + e^{-aT})z^{-1}] + e^{-aT}z^{-2}} \tag{12}$$

where T = sampling period.

Substituting values for a, b, and T of

a = 1.116
b = 53.906
T = 0.001

the transfer function of the motor becomes

$$G_m(z) = \frac{\theta(z)}{V(s)} = \left(\frac{0.2694z^{-1} + 0.2693z^{-2}}{1 - 1.999z^{-1} + 0.999}\right)10^{-4}\ K_g$$

where K_g is a gain constant.

By introducing a numerator gain factor, the above equation can be rewritten

$$G_m(z) = \frac{\theta(z)}{V(s)} = \left(\frac{0.2694z^{-1} + 0.2693z^{-2}}{1 - 1.999z^{-1} + 0.999}\right)K_m \tag{13}$$

where K_m is a numerator gain factor.

Digital Controller Design

The next step in designing a digital control system is to design the controller. Before designing the controller, an appropriate structure for the controller must be selected. This will be influenced by the performance requirements of the system and the processing capability of the processor. The controller may be designed in the continuous domain or s-domain and then converted into discrete form by using one of the previously described discretization methods. Alternatively, the entire design may be carried out in the discrete domain or z-domain. It is assumed here that the design is carried out in the discrete domain. Here, an overview of different types of control algorithms is given and designing/implementing considerations for selected controllers are discussed.

Control Algorithms: The first step in designing the controller is to select an appropriate algorithm or controller structure. The processing burden imposed upon the controller is directly dependent upon the complexity and type of controller structure.

Compensation Techniques: Compensation techniques are one of the most commonly used control techniques. In this technique, the controller adds poles and zeros to get a desired system response. If the low-frequency response is modified, the controller is known as a lag compensator; if the high-frequency response is modified, it is known as a lead compensator. For a continuous control system, the controller is designed in the s-domain by implementing some of the well-known methods such as root locus, Bode plots, and Nyquist plots. The analog or s-domain design is then transferred into a discrete form or z-domain via transformation technique. Alternatively, the compensator can be designed directly in the z-domain by using z-domain frequency response methods or the z-domain root locus method. Compensation techniques allow for somewhat accurate modification to system behavior.

PID: The P (proportional), I (integral), and D (derivative) is a very commonly used analog control technique. In a PID controller, terms proportional to the error term, its integral, and its derivative are summed to achieve the controller output. A PID controller may be designed in the s-domain and then transferred into the z-domain by using one of the transformation methods. Alternatively, the PID algorithm is converted into a discrete form, and the design is carried out entirely in the z-domain. PID is probably the most commonly used algorithm. PID controllers are very robust, although the design of coefficients is somewhat arbitrary.

Deadbeat: A deadbeat algorithm is used when a quick settling time is required. Deadbeat design is carried out entirely in the z-domain. A deadbeat controller replaces the poles of the system with poles at the origin of z-domain.

State Space Model: In a state space model, a complete representation of the system is made in matrix form. This is accomplished by identifying and developing the relationship between the different states or variables of the plant. An appropriate feedback gain can be chosen to place the poles of the system at any desired location in the z-domain. State controllers are used to control multiple variables or states. These controllers are not implemented directly, because it may not be possible to measure all states. They are usually used in conjunction with observers. State space controllers allow precise control of system behavior.

Observer Model: Often in control systems, some of the states of the system are not available for measurement. An observer model or an estimator can be used to estimate the unknown states from the measurement of some of the known states. The estimated states along with an appropriate feedback gain can be used to complete the control loop and place the poles at any desired location. Observers are typically used in conjunction with state controllers when access to all state variables is not available.

Optimal Control: Optimal control synthesis is used when a specific performance or cost criterion (time and energy) must be minimized. Using the given criterion or function, an appropriate control law is derived, which is then implemented with a compensator (LQR – Linear Quadratic Regulator) or controller.

Kalman Filter: An observer model is used in a system where an exact measurement of some states is available. However, in stochastic systems, the presence of noise or uncertainty makes it impossible to make an exact measurement. A Kalman filter is an observer model in a noisy or stochastic system.

Adaptive Control: Adaptive control is used in systems in which there is insufficient information about the plant parameters, making it impossible to derive a plant model. It is also used in systems where plant parameters or plant models change over time, making the controller unstable. An adaptive controller tracks realtime changes in the plant by redesigning the controller to give optimum control system.

The next step in designing the controller is to specify the performance requirements of the system.

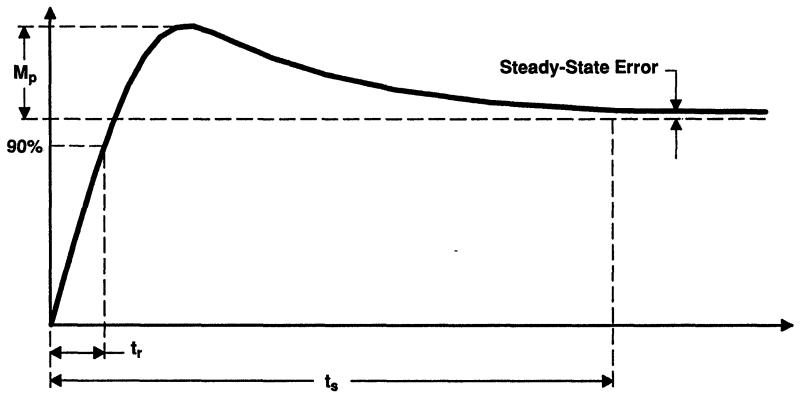
Performance Specifications: Performance requirements of the system dictate selection and design goals of an appropriate controller structure. The specifications can be given in terms of the step (or transient) response, the frequency response, or another criteria.

Step Response: For the step or transient response as shown in Figure 8, the controller requirements are given in terms of the following specifications:

- Steady-state accuracy
- Rise time
- Overshoot
- Settling time

The steady-state error is defined as the deviation at steady-state of the actual system response from the desired system response. For a discrete system (i.e., an integrator), the steady-state error becomes 0 if GH(z) has at least one pole at z = 1, where G(z) is the plant transfer function, and H(z) is the controller transfer function. For a ramp input, the steady-state error becomes 0 if GH(z) has double pole at z = 1. For a unit acceleration input, the steady-state error becomes 0 if GH(z) has a triple pole at z = 1.

Figure 8. Performance Specification for the Step or Transient Response



The rise time, overshoot, and settling time can be specified in terms of the damping ratio ζ and the natural frequency ω_n. To carry out the design in the digital domain, these performance requirements must be mapped to pole locations in the z-plane.

The rise time is specified as when the output reaches 90% of its final value.

$$t_r = \frac{\pi}{2\omega_n\sqrt{1-\zeta^2}}$$

This can be simplified to yield

$$\omega_n \geq \frac{1.8}{t_r}$$

The settling time is specified as when the output settles and remains within the desired range of its final output. This is specified as an absolute percentage of the final value usually 2%. It is given by:

$$t_s = \frac{4.6}{\zeta\omega_n}$$

This implies that

$$\zeta\omega_n \geq \frac{4.6}{t_s}$$

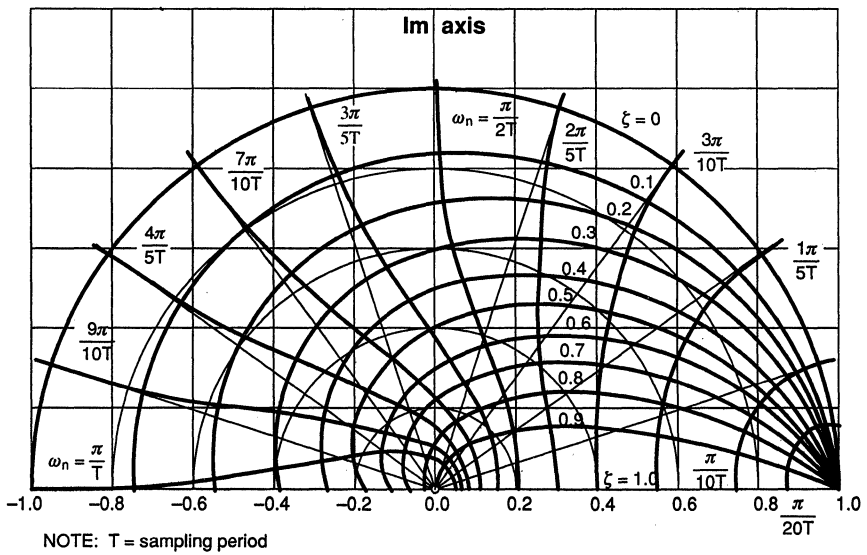
The overshoot is defined as the maximum deviation in percentage of the systems response from the desired value and is given by:

$$M_p = 100 \exp\left(-\frac{\zeta\pi}{\sqrt{1-\zeta^2}}\right)$$

A constant damping ratio, zeta, in the s-plane for $0 < \zeta < 1$, is mapped as a logarithmic spiral in the z-plane. If the poles are specified as having a damping ratio of not less than ζ_1, then the poles must lie within the region bounded by the logarithmic spiral corresponding to $\zeta = \zeta_1$. For a desirable second-order system, the damping rate must be between 0.4 and 0.8. Small values of ζ, such as $\zeta < 0.4$, yield excessive overshoot and large values, such as $\zeta > 0.8$, make the response sluggish.

A constant natural frequency ω_n in the s-plane maps as a straight line emanating from the origin. Figure 9 shows the loci of constant ζ and line of constant ω_n in the unit circle in the z-plane.

Figure 9. Root Locus of Constants ζ and ω_n



NOTE: T = sampling period

Frequency Response: If the performance specifications are specified in terms of the frequency response, they are given in terms of phase margin, gain margin, and cross-over frequency ω_c as shown in Figure 10 — essentially specifying the bandwidth of the closed-loop system.

The cross-over frequency ω_c is defined as the frequency where the phase angle, $\angle GH(j\omega)$, of an open-loop system equals $-180°$.

The gain margin is defined as the magnitude, $|GH(j\omega)|$, (in *decibels*) that lies both below 0 db and at the cross-over frequency.

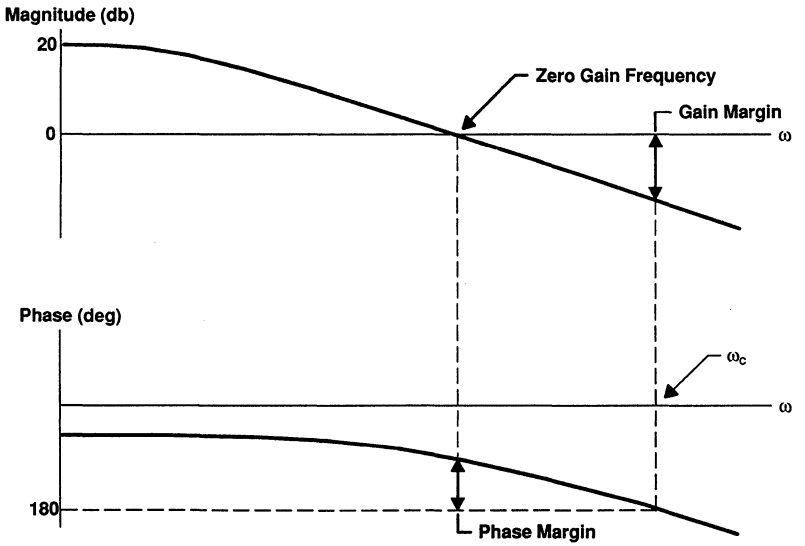
The phase margin is defined as the phase, $\angle GH(j\omega)$, (in *degrees*) that lies both above $-180°$ and at the zero gain frequency.

To directly use frequency response methods, the z-plane is mapped into the w-plane by using the inverse bilinear transformation given by

$$w = \frac{2}{t}\left(\frac{1-z}{1+z}\right)$$

The w-plane mathematics is similar to the s-plane mathematics. The controller is transformed to the w-plane, and most of the classical techniques like Bode analysis can be carried out in the w-plane. Once the compensator is designed in the w-plane, it can be transformed back into the z-plane.

Figure 10. Frequency Response Curves



Additional Criteria for Performance Specification: Some of the other performance requirements can be specified as:

- Disturbance rejection
- Control effort
- Sensitivity to parameter changes

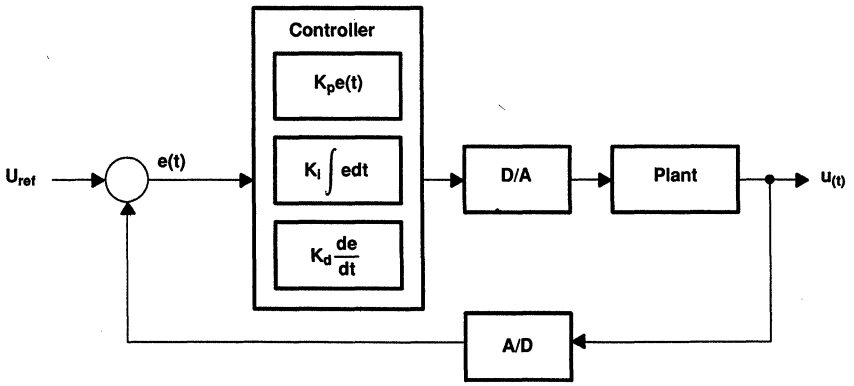
One of the primary goals of a control system is to reject disturbances while maintaining stability under a wide variety of operating conditions. In fact, without disturbances, there would be no need for closed-loop control systems. The feedback gains in a control loop act to minimize disturbances. For example, if a disturbance is constant, then integral action will cause the steady-state error to be zero. However, if the disturbance is of a different nature, then additional steps may have to be taken. It is important to take into account the source of the disturbance and make the preceding gain large. If the disturbance is outside the control loop and affects the measurement or reference input, then a feedforward path can minimize the disturbance. If the disturbance is inside the loop and affects the plant itself, then the loop gain must be made large.

Sensitivity to parameter changes can be an important consideration, especially if the plant has slow-varying parameters due to drift. Minimizing these effects is similar to handling disturbances. However, some controller structures like deadbeat controllers that perform pole-zero cancellations are more sensitive to parameter variations and should be avoided. If parameter variation is an extremely critical consideration, then adaptive control should be used.

Sometimes it is necessary to minimize either the control effort or other parameter(s) in the system. Optimal control techniques can be used to determine a control law and do pole placement. They are discussed in subsection **Optimal Control and Estimation**. In general, a system with either minimum response or a high bandwidth requires higher control efforts.

PID Controller: This topic describes the design and implementation of a PID controller. Figure 11 shows a block diagram of a control system using the PID controller. PID is a commonly used technique in classical control. In designing controllers, it is often found that just minimizing a term proportional to the error is not sufficient. The inclusion of the integral of the error term will reduce the steady-state error to

Figure 11. Block Diagram of a Control System Using PID Controller



zero because it represents the accumulated error. To further improve stability and plant dynamics, a differential of the error term is introduced. This term represents the error rate. A PID controller that includes all three terms can give very good results. It can be used in its discrete form with digital control systems. If both low-frequency and high-frequency responses are modified, this controller can be viewed as a special lead-lag compensator.

Controller Design: The trapezoidal approximation is used for conversion of PID into discrete form. Usually, the trapezoidal approximation is used for the integral term, and the backward difference is used for the differential term. However, when the design is carried out in the z-domain, the approximation techniques are not important. The design is carried out as a compensator with a pole at z=1 to ensure integral behavior. Hence, the following design is done directly in the z-domain using pole placement techniques.

The analog PID algorithm is given by:

$$u(t) = K_p e(t) + K_i \int e dt + K_d \frac{de}{dt} \tag{14}$$

where

K_p, K_i, and K_d = PID constants

u(t) = output of controller

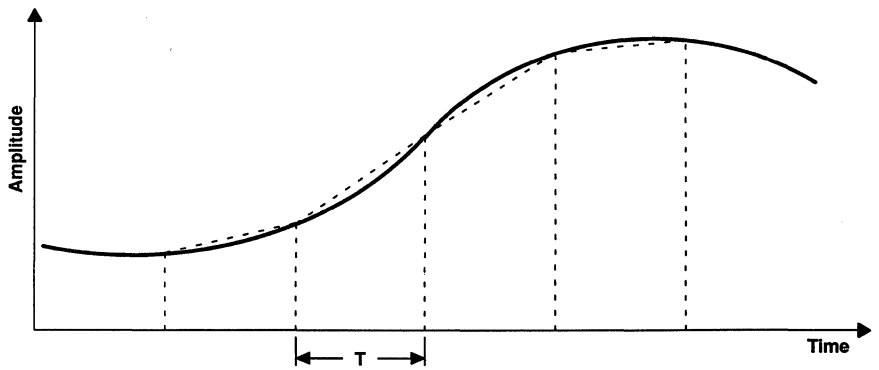
e(t) = error signal

In a trapezoidal approximation, also called Tustin transformation, the area of the integral $\int e dt$ is given by the summation of small trapezoids, see Figure 12.

The integral $\int e dt$ can also be solved by taking the Laplace transform of equation (14) and substituting for the s. The Laplace transform of (14) gives

$$U(s) = \left(K_p + sK_d + \frac{K_i}{s} \right) [E(s)]$$

Figure 12. Trapezoidal Approximation



Using the Tustin approximation or substituting for s where

$$s = \left(\frac{2}{T}\right)\left(\frac{z-1}{z+1}\right)$$

After substitution and solving where z^{-1} represents a delay of one sample time,

$$u(n) = u(n-2) + K_p[e(n) - e(n-2)] + \left(\frac{2K_d}{T}\right)[e(n) - 2e(n-1) + e(n-2)] + \left(\frac{K_iT}{2}\right)[e(n) + 2e(n-1) + e(n-2)]$$

Combining elements, the above equation can be restated as

$$u(n) = u(n-2) + K_1[e(n)] + K_2[e(n-1)] + K_3[e(n-2)] \tag{15}$$

where

$$K_1 = K_p + \frac{2K_d}{T} + \frac{K_iT}{2}$$

$$K_2 = K_iT - \frac{4K_d}{T}$$

$$K_3 = \frac{2K_d}{T} - K_p + \frac{K_iT}{2}$$

$u(n) = n^{th}$ sample of output of controller

$u(n-2) = (n-2)^{nd}$ sample of output.

This is the final form of the PID controller. At this point, the controller coefficients must be determined. The PID controller can be designed by determining K_p, K_i, and K_d, solving K_1, K_2, and K_3, and substituting into equation (15). Alternatively, the design can be carried out in the z-domain; and, constants K_1, K_2, and K_3 can be directly determined.

The gain constants K_1, K_2, and K_3 are designed by selecting the poles for the system transfer function (i.e., controller + plant). The dominant poles are selected by choosing a desired characteristic equation. The rest of the poles can be selected by placing them near the origin. These polar locations are chosen to ensure system stability and a desired system response. Note that pole locations can also be chosen by using both the step response performance criteria and the root locus from the z-plane's unit circle in Figure 9. However, some fine-tuning may be necessary to achieve an optimum response from the system. As the poles move toward the unit circle, the system response speed decreases while the overshoot increases, and the system may become unstable if the poles are selected just inside or outside of the unit circle's boundary. For example, Figure 13, Figure 14, and Figure 15 show step-response curves of a PID controller being influenced by the system's poles. The transfer function for the controller can be stated as

$$G_c(z) = \frac{K_1 + K_2z^{-1} + K_3z^{-2}}{1 - z^{-2}} \tag{16}$$

The transfer function of the plant is given by

$$G_p(z) = \frac{0.2694z + 0.2693}{z^2 - 1.999z + 0.999}$$

The overall system transfer function is expressed as

$$G_s(z) = \frac{[G_p(z)][G_c(z)]}{1 + [G_p(z)][G_c(z)]} \tag{17}$$

Figure 13. Position Step Response of a PID Controller

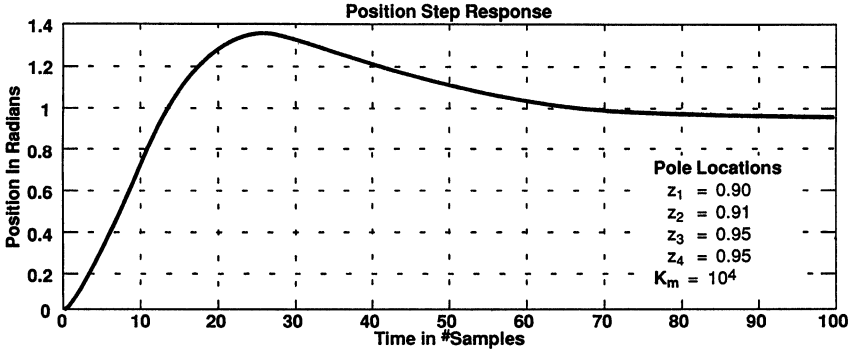


Figure 14. Position Step Response of a PID Controller

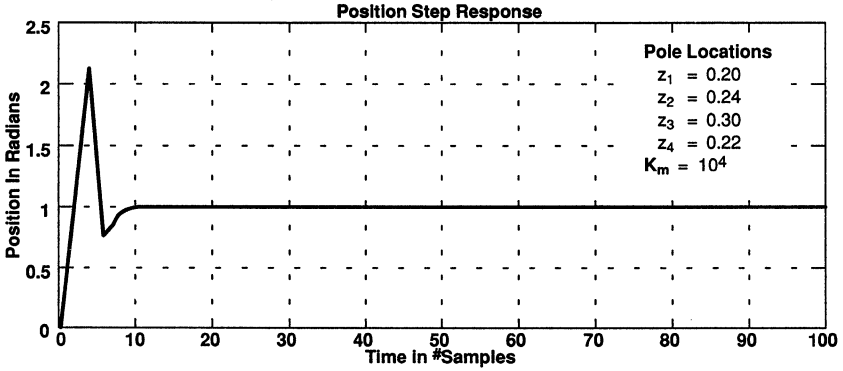
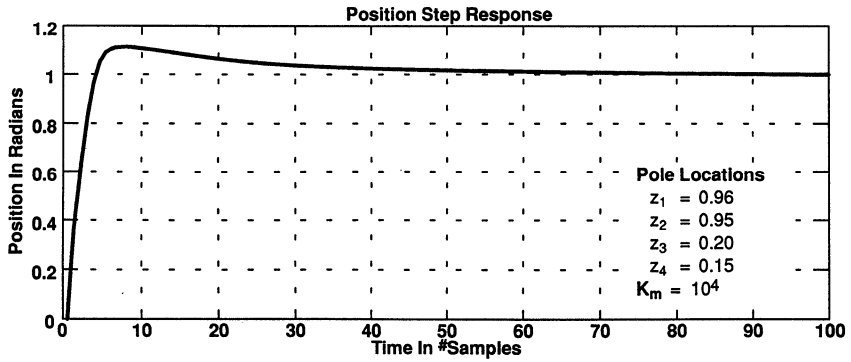


Figure 15. Position Step Response of a PID Controller



The denominator of the system transfer function provides us the poles of the overall system. The stability and robustness of the system depend upon the location of these poles in the z-domain. Assuming pole locations of 0.96, 0.95, 0.20, and 0.15, a desired characteristic equation is obtained. To solve for values of K_1, K_2, and K_3, the coefficients of powers of z for the denominator of the system transfer function are compared with the desired polynomial. Appendix 1 shows the design carried out by using PC–Matlab. The zero order hold represents the function of the D/A converter; the sampler represents the function of the A/D converter. The closed-loop system pole locations are input to the program, and the coefficients K_1, K_2, and K_3 are calculated to ensure desired pole locations. One of the pole locations is chosen at z=1 to ensure integral action. Solving for K_1, K_2, and K_3 gives

$K_1 = 1.4795$

$K_2 = -2.845$

$K_3 = 1.3636$

Our final algorithm comes out to:

$$u(n) = u(n-2) + 1.4795[e(n)] - 2.8405[e(n-1)] + 1.3636[e(n-2)] \quad (18)$$

Implementation Considerations: The PID design above has used the traditional or textbook definition. In practice, a number of refinements are made to the standard form to give it better behavior in some cases. Although designing directly in the z-domain avoids some of the problems, several concerns are discussed here.

One of the major problems faced in implementation of PID controllers is integral windup. A large change in the error signal can cause the integral to build up a large gain and make the actuator saturate. This essentially means that the control loop is running open. Even after the error goes to zero, the controller continues to integrate because of the integral action; consequently, the integral term could become very large. The error signal must change sign long before the controller normalizes; otherwise, the integral windup could cause large transients.

Several options can minimize the effect of integral windup. One possibility is to build an extra feedback loop around the actuator and control the error between the controller output and the actuator output. Another possibility is to stop the integral action when the output saturates. This can be done very easily in the processor by detecting output saturation (using saturation mode in TMS320) and using another set of coefficients that do not include integral action. Also, it is good practice to limit the contribution of the integral term between 10 % and 20 % of the control effort.

Another concern is the behavior of the derivative term. A large number of controllers are implemented as PI controllers to avoid derivative action. Differentiation enhances noise, and derivative term can contribute to high-frequency measurement noise. It is necessary to limit the derivative gain at high frequency by placing a pole in the derivative term given by

$$\frac{1}{1 + \frac{SK_p}{N}}$$

where N is in the range of 3 – 10.

The derivative term also will amplify the noise for any sudden changes in the set point. This is known as the derivative kick. For this reason, the set point is sometimes fed only to the integral term.

One of the disadvantages of carrying out the design in the discrete domain is that the PID gains are not explicit, and no direct control over integral, derivative, and proportional gains is available. However, pole

placement design techniques give more control over the frequency response and treat the controller as a standard compensator. Integral action is ensured by placing a pole at z=1. Actually, the PID controller is a special case of a phase lag-lead compensator. The PD control action affects the high-frequency region by increasing the phase-lead angle. It improves system stability and thus increases the speed of response. The PI control action affects the low-frequency portion by increasing the low-frequency gain and reducing the steady-state error.

Deadbeat Controller: One of the desired characteristics in a control system design is a quick settling time. In an analog controller, the system output theoretically uses an infinite time to settle exactly to the reference input signal. A deadbeat controller is used when a quick or finite settling time is required. A deadbeat controller will reach a steady-state in n+1 samples where n is the order of the controller. Essentially, a deadbeat controller cancels all the poles of the system and replaces them with poles at the origin. Another advantage of deadbeat controllers is that they require few calculations. Therefore, they can be used in systems where synthesis must be repeated frequently (e.g., adaptive control systems).

Controller Design: The transfer function of a deadbeat controller is given by

$$G_{db} = \frac{p_0 + p_1 z^{-1} + p_2 z^{-2} + \ldots + p_n z^{-n}}{q_0 + q_1 z^{-1} + q_2 z^{-2} + \ldots + q_n z^{-n}} \tag{19}$$

The order n of the controller transfer function is the same as the order of the plant transfer function, or n=2. The deadbeat controller will reach final state in n+1 or three sample time intervals.

To design the deadbeat controller, its coefficients $p_0 \cdots p_n$ and $q_0 \cdots q_n$ must be found from the parameters of the motor.

The general form of a plant (i.e., motor) is given by

$$G_p(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_n z^{-n}}{a_0 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_n z^{-n}}$$

If R(z) is the reference input, the coefficients p_n and q_n are

$$p_0 = \frac{r}{\sum b_i} = \frac{r}{b_0 + b_1 + b_2 + \ldots}$$

$p_1 = a_1 p_0$

$p_2 = a_2 p_0$

.

.

.

$p_n = a_n p_0$

and

$q_0 = r - b_0 p_0$

$q_1 = -b_1 p_0$

$q_2 = -b_2 p_0$

.

.

.

$q_n = -b_n p_0$

The transfer function of the DC servo motor is

$$G_p(z) = \frac{0.2694z^{-1} + 0.2693z^{-2}}{1 - 1.999z^{-1} + 0.999z^{-2}}$$

Since the plant transfer function is a second-order system, the deadbeat controller is also a second-order system (n = 2).

From the plant transfer function,

$a_0 = 1, \quad a_1 = -1.999, \quad a_2 = 0.999$

$b_0 = 0, \quad b_1 = 0.2694, \quad b_2 = 0.2693$

The numerator and denominator of $G_{db}(z)$ is divided by r. Thus, r disappears from the calculations of coefficients.

Solving for the coefficients yields

$$p_0 = \frac{1}{b_0 + b_1 + b_2} = 0.1566$$

$p_1 = a_1 p_0 = -0.3129$

$p_2 = a_2 p_0 = 0.1564$

$$q_0 = 1 - \frac{b_0 p_0}{r} = 1$$

$q_1 = -b_1 p_0 = -0.4218$

$q_2 = -b_2 p_0 = -0.4216$

The controller becomes

$$G_{db}(z) = \frac{0.1566 - 0.3129z^{-1} + 0.1564z^{-2}}{1 - 0.4218z^{-1} - 0.4216z^{-2}} \tag{20}$$

or, in time domain, it can be represented as

$$u(n) = 0.1566[u(n-1)] + 0.4216[u(n-2)] + 0.1566[e(n)] - 0.3129[e(n-1)] + 0.1564[e(n-2)] \tag{21}$$

Appendix 2 shows a PC–Matlab program that designs and simulates a deadbeat controller. Figure 16, Figure 17, and Figure 18 show the response of a deadbeat controller with different values of sampling rates and DC gain.

Implementation Considerations: Deadbeat controllers compensate for the poles of the system and place all the poles of the closed-loop system at the origin or z=0; therefore, they should not be applied to unstable systems with poles outside the unit circle or with poles in the vicinity of the unit circle in the z-plane. Similarly, zeroes outside the unit circle should not be cancelled with unstable poles. Thus, deadbeat controllers should be used only with stable plants or processes to prevent instability. Since deadbeat controllers do pole-zero cancellation, they are also sensitive to parameter variations. Deadbeat controllers can also be viewed as a special case of pole placement where all the poles are placed at the origin.

The only design parameter in deadbeat controllers is the sampling period; therefore, it is important to carefully choose the sampling period when using deadbeat control. Selection of the sampling period influences the magnitude of the control signal; the magnitude of the control signal increases with a decreasing sampling period. This can lead to a large amount of gain and then to actuator saturation. This is one of the main reasons why deadbeat controllers are not commonly used.

Figure 16. Position Step Response of a Deadbeat Controller

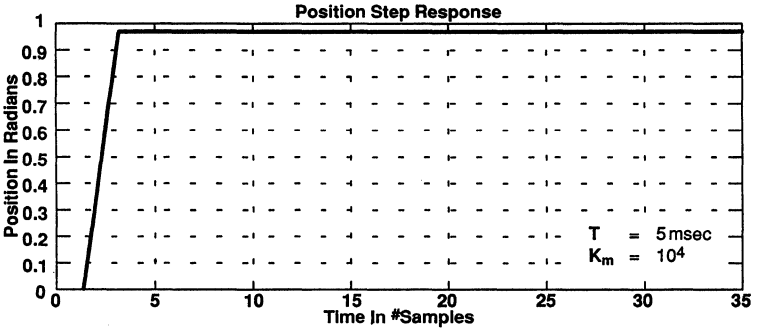


Figure 17. Position Step Response of a Deadbeat Controller

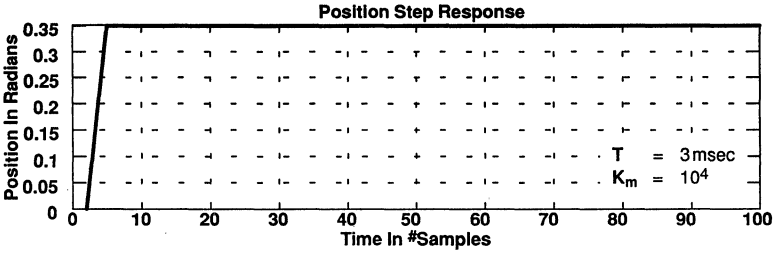
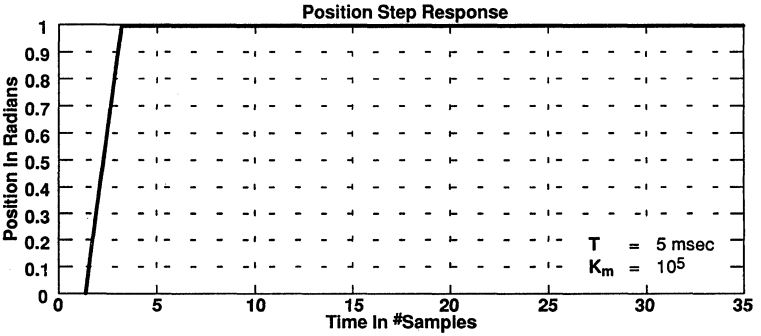


Figure 18. Position Step Response of a Deadbeat Controller



Deadbeat controllers are designed to optimize rise and settling time. They trade overshoot for rise time, so they may exhibit large overshoot. Overshoot can be reduced by increasing the settling time. Besides increasing the sampling period, there are two ways to reduce the overshoot. The first method is to design an extended order deadbeat controller that can specify either u(0) or the initial control action. Since u(0) has the largest magnitude, this controls the overshoot. An alternate method is to divide the r(t) or the desired final state into two or three sublevels and to reach final steady-state in [2(n+1)] or [3(n+1)] sample times

instead of (n+1) sample times. This essentially has the same effect as increasing the sample time. However, the final overshoot can be more precisely controlled depending upon how r(t) is subdivided.

State Space Model: State space formulation is one of the fundamental concepts of modern control theory. Most modern control systems are designed using a state space model approach. A state space model allows the representation of a complete system, whether single variable or multiple input/output. The state controller is able to simultaneously control all of the specified states or variables of that system. This type of controller lends itself naturally to solutions with computers and can be used to handle certain types of nonlinear and time-varying systems.

In a state space model, the system is described by a number of first order equations. An analog or continuous data system is represented by a set of first-order differential equations, called state equations. For a digital or discrete data system, the state equations are first-order difference equations. These are then combined into vector-matrix equations. The use of vectors and matrices greatly simplifies the mathematical description of the system.

State Controller Design: In a state controller, feedback gains are provided to all of the states or variables (i.e., position, velocity, torque) that are included in the state space model. These feedback gains can be either constant or time-varying. Pole placement techniques are used to place all the poles of the closed-loop systems at the selected locations and to calculate the feedback gains, thus obtaining the desired response. This analysis assumes that all the states are being measured and are known. In practice, this is unrealistic. Hence, the analysis is carried out in two phases. In the first phase, it is assumed that all the states are available and that an appropriate controller will be designed; the second phase shows the use of estimators. The estimator is used to reconstruct all the states of the system from measurement of some of the states. For the analysis below, it does not matter whether the states are being measured with sensors or are being reconstructed with estimators.

In general, the state space description of system is given by these equations:

$$\mathbf{x}(n+1) = \mathbf{A}[\mathbf{x}(n)] + \mathbf{B}[\mathbf{u}(n)] \tag{22}$$
$$\mathbf{y}(n) = \mathbf{C}[\mathbf{x}(n)] + \mathbf{D}[\mathbf{u}(n)]$$

The actual controller is given by the following equation:

$$\mathbf{u}(n) = -\{\mathbf{K}[\mathbf{x}(n)]\} \tag{23}$$

where $\mathbf{x}(n+1)$, $\mathbf{x}(n)$, **A**, **B**, **C**, **D**, and **K** are matrices described as follows.

$\mathbf{x}(n)$, $\mathbf{x}(n+1)$ — state vectors describe the system states
A — state transition matrix describes plant behavior
B — input matrix describes affects of inputs
C — output matrix describes which states are measured
D — direct link matrix describes feedforward gains
K — feedback matrix describes feedback gains
u — control vector describes control inputs
y — output vector describes measurements of plant output

The state space model for a DC motor can be derived from the electrical and mechanical characteristics of the motor. The mechanical characteristics are given by

$$J_m\ddot{\theta} + D\dot{\theta} + K\theta = T_L - J_L\ddot{\theta}$$

The electrical characteristics are given by

$$L\frac{di}{dt} + Ri = V - emf$$

Simplifying and combining above two equations yields

$$\ddot{\theta} = bu - a\dot{\theta} \tag{24}$$

$$a = \left(\frac{1}{J}\right)\left(B + \frac{K_t K_e}{R}\right)$$

$$b = \left(\frac{1}{J}\right)\left(\frac{K_t}{R}\right)$$

$$\mathbf{u} = V$$

$$\mathbf{K} = 0$$

Assuming that the states are described as

$x_1 = \theta$ for position

$x_2 = \dot{x}_1 = \dot{\theta}$ for velocity

$\dot{x}_2 = \ddot{\theta}$

The state space model can be defined as

$$\dot{x}_1 = x_2 \tag{25}$$

$$\dot{x}_2 = \ddot{\theta} = bu - ax_2 \tag{26}$$

Combining (25) and (26) in a matrix form, the model for a continuous data system can now be defined as

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -a \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ b \end{bmatrix} u(n) \tag{27}$$

Equation (27) is a set of first-order differential equations and describes the system in a continuous data form. A conversion to a set of difference equations is necessary for use in a discrete data system. The discrete form is given by equation (28). The derivation of this is fairly involved and not presented here. However, if the values of a and b are substituted in equation above, the discrete equivalent can be found by using PC–Matlab. The general form of the discrete state space model is given by:

$$\begin{bmatrix} x_1(n+1) \\ x_2(n+1) \end{bmatrix} = \begin{bmatrix} 1 & \frac{1}{a}\left(1 - e^{-aT}\right) \\ 0 & e^{-aT} \end{bmatrix} \begin{bmatrix} x_1(n) \\ x_2(n) \end{bmatrix} + \frac{b}{a} \begin{bmatrix} T - \frac{1}{a}\left(1 - e^{-aT}\right) \\ 1 - e^{-aT} \end{bmatrix} u(n) \tag{28}$$

where

T = sampling interval

$x_1(n)$ = position at time interval n

$x_2(n)$ = velocity at time interval n+1

Using the PC–Matlab function given below and substituting values for a, b, and T, we can obtain the discrete equivalent.

Using $\mathbf{c_2d(aa,bb) = A,B,C,D}$ and the given values for its parameters, we find the discrete equivalent of the model as:

$$\begin{bmatrix} x_1(n+1) \\ x_2(n+1) \end{bmatrix} = \begin{bmatrix} 1 & 0.041 \\ 0 & 0.219 \end{bmatrix} \begin{bmatrix} x_1(n) \\ x_2(n) \end{bmatrix} + 87.9 \begin{bmatrix} 0.039 \\ 0.78 \end{bmatrix} u(n) \tag{29}$$

The design problem is to find elements of feedback gain matrix $\mathbf{K}(k_1, k_2, \cdots)$ so that the closed-loop system has the desired response.

After substituting (23) for u(n), equation (22) can be restated as

$$\mathbf{x}(n+1) = \mathbf{A}[\mathbf{x}(n)] - \mathbf{B}\{\mathbf{K}[\mathbf{x}(n)]\}$$

or

$$\mathbf{x}(n+1) = (\mathbf{A} - \mathbf{BK})[\mathbf{x}(n)] \tag{30}$$

Equation (30) describes the closed-loop state model. The behavior of the closed loop is determined by solving of the characteristic equation given by

$$|z\mathbf{I} - \mathbf{A} + \mathbf{BK}| = 0$$

Solving this gives an equation with unknowns k_1 and k_2 (elements of $\mathbf{K}$). k_1 and k_2 can be solved by comparing coefficients with the polynomial with desired pole locations (in other words, if pole locations for the actual controller are chosen as 0.90 and 0.95, the characteristic polynomial is given by $z^2 - 1.85z + 0.855$). k_1 and k_2 will be found again by using PC–Matlab. The function **PLACE**, given pole locations r, will solve for k_1 and k_2. Using the following command,

K = PLACE(A,B,r)

we obtain the following values

$k_1 = 0.089$

$k_2 = 0.001$

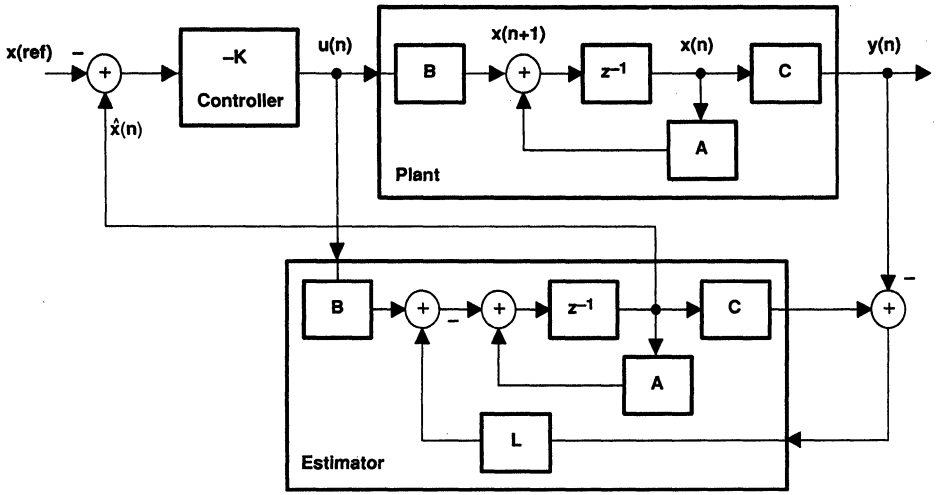
The state controller is then implemented as

$$u(n) = -0.089[x_1(n)] - 0.001[x_2(n)] \tag{31}$$

Implementation Considerations: See *Implementation Considerations* for the observer model on page 63.

Observer Model: The concept of being observable is another fundamental idea of modern control theory. An observer model is an estimate of all the states of a plant derived from measurements of some of the outputs (for instance, states such as velocity or current can be derived from measurement of displacement). Essentially, it reconstructs the state by simulating in realtime the behavior of the system and then compares the results to the measurements. In some cases, design of this type may be necessary if one of the states that is needed for the controller is not measurable. In other cases, use of an observer can reduce the number of sensors required or allow lower-cost sensors to be used. Figure 19 shows the block diagram of a state controller and an estimator.

Figure 19. State Controller/Estimator



Observer Model and Estimator Designs: The observer model is described by

$$\hat{\mathbf{x}}(n+1) = \mathbf{A}[\hat{\mathbf{x}}(n)] + \mathbf{B}[\mathbf{u}(n)] + \mathbf{L}[y(n) - \mathbf{C}\hat{\mathbf{x}}(n)] \quad (32)$$

where

$y(n) - \mathbf{C}\hat{\mathbf{x}}(n)$ = estimation error
$\hat{\mathbf{x}}(n)$ = estimated states
$\mathbf{C}$ = [1 0] for position measurement
$\mathbf{L}$ = observer gain matrix

Equation (32) can be rearranged in the form of

$$\hat{\mathbf{x}}(n+1) = (\mathbf{A} - \mathbf{LC})[\hat{\mathbf{x}}(n)] + \mathbf{B}[\mathbf{u}(n)] + \mathbf{L}[y(n)]$$

Substituting for u(n), the equation can be restated as

$$\hat{\mathbf{x}}(n+1) = (\mathbf{A} - \mathbf{BK} - \mathbf{LC})[\hat{\mathbf{x}}(n)] + \mathbf{L}[y(n)] \quad (33)$$

According to the principle of separation, the controller dynamics can be separated from the observer dynamics, and both can then be designed independently. The controller dynamics are determined by the characteristic equation given by

$$|z\mathbf{I} - \mathbf{A} - \mathbf{BK}|$$

The estimator dynamics are given by this characteristic equation:

$$|z\mathbf{I} - \mathbf{A} - \mathbf{LC}|$$

The estimator design is developed like the controller design in the previous paragraphs; that is, gains are designed for the estimator matrix L (i.e., l_1, l_2). l_1 and l_2 are found by selecting the poles of the observer to be slightly faster than the system to allow quicker convergence. If the poles are placed at z=0, the observer is then referred to as a deadbeat observer.

Solving the equation $|z\mathbf{I} - \mathbf{A} - \mathbf{LC}|$ provides an equation in term of unknowns l_1 and l_2. Selecting desired pole locations gives a characteristic polynomial. The poles of the controller are given by

$z_1, z_2 = 0.9, 0.95$

Using slightly faster poles for the observer and placing them at

$z_1, z_2 = 0.4, 0.5$ gives the characteristic polynomial as

$z^2 - 0.9z - 0.20$

l_1 and l_2 were found by using the PLACE function of PC–Matlab given pole location l, **L = PLACE(A',c',l)**,

$l_1 = 0.79$

$l_2 = 2.95$

Appendix 3 shows the complete listing of the PC–Matlab program. The observer model is

$$\begin{bmatrix} x_1(n+1) \\ x_2(n+1) \end{bmatrix} = \begin{bmatrix} 1.07 & 0.041 \\ -0.09 & 0.219 \end{bmatrix} \begin{bmatrix} x_1(n) \\ x_2(n) \end{bmatrix} + \begin{bmatrix} 2.8 \\ 68.5 \end{bmatrix} u(n) + \begin{bmatrix} 0.79 \\ 2.95 \end{bmatrix} y(n) \tag{34}$$

Figure 20 (a and b), Figure 21 (a and b), Figure 22 (a and b), and Figure 23 (a and b), show the response of a state controller and an estimator for various pole locations.

Transfer Function Form: The observer and the state controller can be implemented by using equation (34), or the mathematics can be further simplified by combining some of the matrices and obtaining an equivalent transfer function. This is possible only for a SISO (single-input/single-output).

The control law given by (31) is assumed, and the state controller designed earlier will be used. The controller is given by

$$\mathbf{u}(n) = -\mathbf{K}[\hat{\mathbf{x}}(n)] \tag{35}$$

Taking the z-transform of equation (33)

$z[\hat{\mathbf{x}}(z)] = (\mathbf{A} - \mathbf{BK} - \mathbf{LC})[\hat{\mathbf{x}}(z)] + \mathbf{L}[y(z)]$

or

$$\hat{\mathbf{x}}(z) = (z\mathbf{I} - \mathbf{A} + \mathbf{BK} + \mathbf{LC})^{-1}\mathbf{L}[y(z)] \tag{36}$$

Taking the z-transform of equation (35) and substituting (36), the controller/observer transfer function can be stated as

$$\frac{\mathbf{U}(z)}{\mathbf{Y}(z)} = -\mathbf{K}(z\mathbf{I} - \mathbf{A} + \mathbf{BK} + \mathbf{LC})^{-1}\mathbf{L} \tag{37}$$

Figure 20. Step Response and Control Effort of the State Controller/Estimator

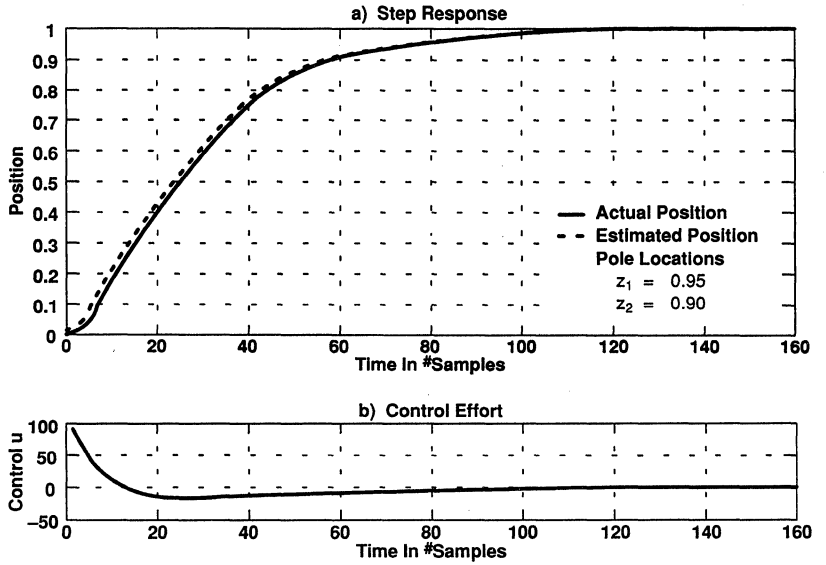


Figure 21. Step Response and Control Effort of the State Controller/Estimator

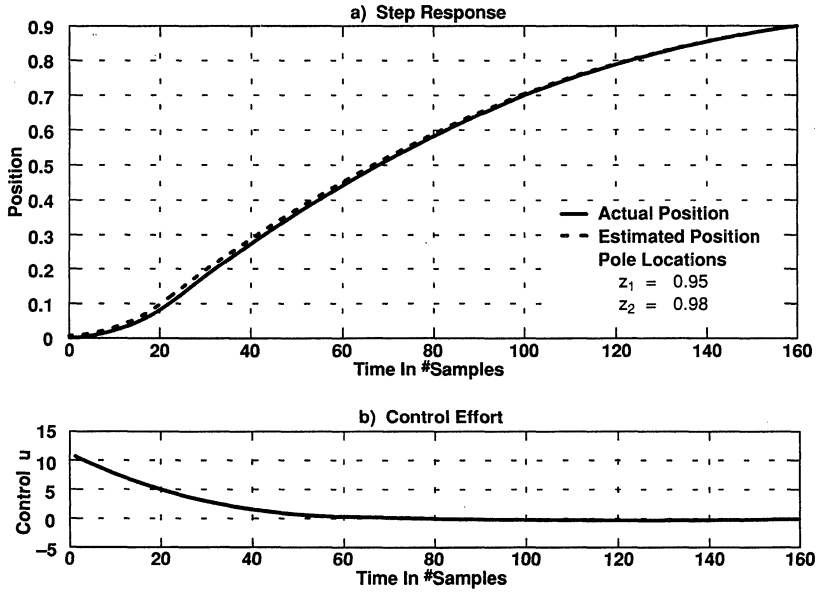


Figure 22. Step Response and Control Effort of the State Controller/Estimator

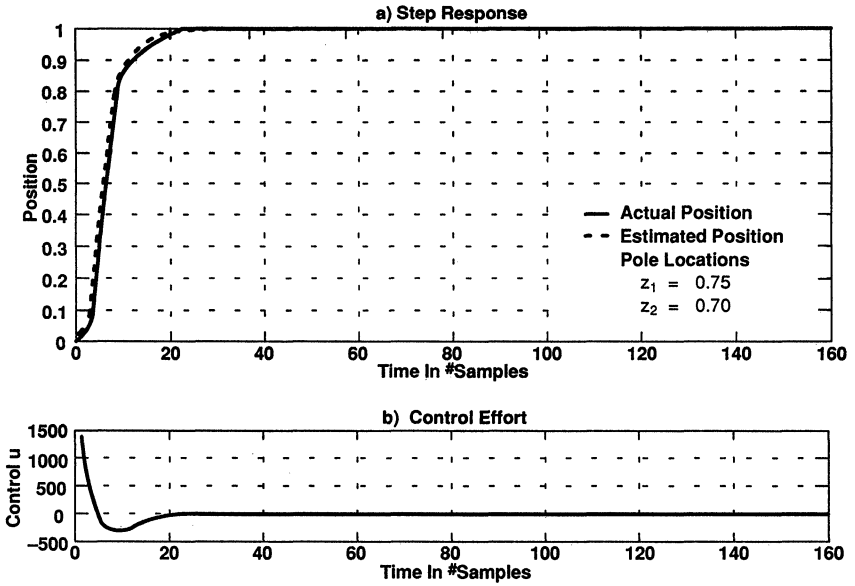
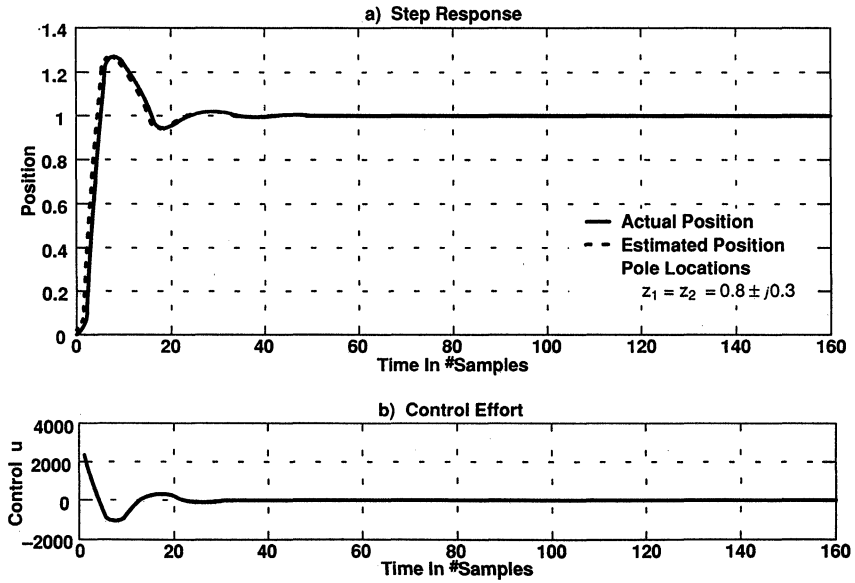


Figure 23. Step Response and Control Effort of the State Controller/Estimator



Substituting values of **A**, **B**, **L**, **C**, and **K** and solving (37), we obtain

$$G(z) = \frac{\mathbf{U}(z)}{\mathbf{Y}(z)} = \frac{0.008z^{-1} + 1.388z^{-2}}{1 - 0.95z^{-1} + 0.12z^{-2}} \tag{38}$$

where

$$\mathbf{A} = \begin{bmatrix} 1 & 0.041 \\ 0 & 0.219 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} 2.81 \\ 68.56 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

$$\mathbf{K} = \begin{bmatrix} 0.089 & 0.001 \end{bmatrix}$$

Transferring (38) in the time domain,

$$\mathbf{u}(n) = 0.95[\mathbf{u}(n-1)] - 0.12[\mathbf{u}(n-2)] + 0.008[y(n-1)] + 1.388[y(n-2)] \tag{39}$$

Equation (39) is the final form of the state controller plus observer model in a transfer function form. However, this form is not commonly used because it loses insight into the estimator dynamics.

State Controller and Estimator with Reference Input: The controller design in the previous topic was done assuming no reference inputs or commands. Instead, the problem deals with driving all the states to zero. This is known as a regulator application. However, there are cases when the state controller may be required to follow a reference command **r**(n). Such a system is known as a servo system.

The general description of a state controller and estimator with reference input can be stated as

$$\hat{\mathbf{x}}(n+1) = (\mathbf{A} - \mathbf{BK} - \mathbf{LC})[\hat{\mathbf{x}}(n)] + \mathbf{L}[y(n)] + \mathbf{M}[\mathbf{r}(n)] \tag{40}$$

and the controller can be described as

$$\mathbf{u}(n) = -\mathbf{K}[\hat{\mathbf{x}}(n)] + \mathbf{N}[\mathbf{r}(n)]$$

N is a scale factor and acts like a DC gain between input and output. **M** is given by **BN**.

If a reference signal is not available, but instead the error between the reference and output signal $e(n) = y(n) - \mathbf{r}(n)$ is available, then the same structure given by equation (40) can be applied. However, in this case, we let **N**=0 and **M**= –**L**; y(n) is replaced by $y(n) - \mathbf{r}(n)$. The state model can be described as

$$\hat{\mathbf{x}}(n+1) = (\mathbf{A} - \mathbf{BK} - \mathbf{LC})[\hat{\mathbf{x}}(n)] + \mathbf{L}[y(n) - \mathbf{r}(n)] \tag{41}$$

$$\mathbf{u}(n) = -\mathbf{K}[\hat{\mathbf{x}}(n)]$$

Appendix 3 lists the PC–Matlab program that also simulates a state controller and an estimator with reference input.

Implementation Considerations: State controllers and estimators are generic structures, which can be used to meet a variety of requirements. They allow full control of closed-loop dynamics and the use of CAD tools. Pole placement can be done by using either optimal methods or classical methods. Realtime pole placement techniques can be used for adaptive control. State controllers are good analytical tools for

designing controllers, but one must be sure that the design can be implemented. For example, if poles are placed at z=0, the problem becomes the same as a deadbeat controller. Care must also be exercised when selecting the states. Although it may be possible to represent the states mathematically, the states may not be controllable or observable. State controllers may also result in a higher-order system, which implies a more sophisticated processing requirement from the controller. The observer's pole locations need to be considered as well. Those pole locations that lie closer to the origin allow the observer to achieve a faster response while, at the same time, increasing its susceptibility to noise. Those that lie farther from the origin improve the observer's noise characteristics while slowing its dynamics.

Optimal Control and Estimation: The previous topics described the design techniques that use pole placement. It was assumed that pole locations were known. This can usually work well for single-input/ single-output or low-order systems; but, for high-order systems, pole placement becomes difficult. Especially where multi-input or multi-output systems are concerned, choosing pole locations can be difficult, and an exact solution may not occur. This topic describes an alternative method to selecting feedback gains that will provide an optimal solution. The first part explains how to choose optimal gains for a feedback controller known as LQR (linear quadratic regulator); the second part describes how to design an optimal observer known as Kalman filter; the final part considers implementation. Appendix 4 lists the PC–Matlab program that simulates both the linear quadratic regulator and the Kalman filter.

Linear Quadratic Regulator: The feedback gain for a LQR controller is chosen by minimizing a cost function or a performance index. This is typically a quadratic function given by

$$J = \sum [\mathbf{x}^T(n)\mathbf{Q}(n)\mathbf{x}(n) + \mathbf{u}^T(n)\mathbf{R}(n)\mathbf{u}(n)] \tag{42}$$

where $\mathbf{u}(n)$ is control vector, $\mathbf{x}(n)$ is state vector, and matrices $\mathbf{Q}$ and $\mathbf{R}$ are weighting matrices that have to be designed. Matrix $\mathbf{Q}$ is symmetric and positive semi-definite, and $\mathbf{R}$ is symmetric and positive definite (a matrix is positive definite if all its eigenvalues are real and positive, and a matrix is positive semi-definite if all its eigenvalues are either real and positive or zero). The first term in the cost function minimizes the states and drives the states to zero as rapidly as possible. The second term minimizes the control effort. Alternative cost functions can also be chosen.

The cost function for a second-order system can be written as

$$J = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} q_{11} & q_{12} \\ q_{21} & q_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} u_1 & u_2 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$= q_{11}(x_1)^2 + (q_{12} + q_{21})x_1x_2 + q_{22}(x_2)^2 + r_{11}(u_1)^2 + (r_{12} + r_{21})u_1u_2 + r_{22}(u_2)^2$$

If it is necessary to minimize x_1 and x_2, then only x_1^2 and x_2^2 need to be minimized. Or, q_{12} and q_{21} can be set to 0. Similarly, r_{12} and r_{21} can be set to 0.

The control law is given by

$\mathbf{u} = -\mathbf{K}\mathbf{x}$

where

$\mathbf{K}$ is the optimal gain.

K is given by

$$\mathbf{K} = [\mathbf{R} + \mathbf{B}^T\mathbf{P}\mathbf{B}]^{-1}\mathbf{B}^T\mathbf{P}\mathbf{A} \tag{43}$$

where

P is given by

$$\mathbf{P}(n+1) = \mathbf{A}^T\mathbf{P}\mathbf{A} + \mathbf{Q} - \mathbf{A}^T\mathbf{P}\mathbf{B}[\mathbf{B}^T\mathbf{P}\mathbf{B} + \mathbf{R}]\mathbf{B}^{-1}\mathbf{B}^T\mathbf{P}\mathbf{A} \tag{44}$$

Equation (43) is known as the algebraic Riccati equation. For steady-state, the above matrices can be assumed to be constant and the optimal gain **K** can be chosen off-line. The gain for the example is again calculated using facilities of PC–Matlab and is given by the following function

K=dlqr(A,B,Q,R)

The following values were selected for **Q** and **R**.

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 \\ 0 & 0.001 \end{bmatrix}$$

$$\mathbf{R} = 1$$

This puts a higher cost on minimizing x_1. The following value is obtained for the feedback gain **K**.

$$\mathbf{K} = \begin{bmatrix} 456.9 & 14.8 \end{bmatrix}$$

Figure 24 (a and b), Figure 25 (a and b), Figure 26 (a and b), and Figure 27 (a and b) show the response of the system with various values of **Q** and **R**. In practice, there is no method for determining exact values for matrices **Q** and **R**. A trial and error method is used to select these matrices. A common technique is to test the step response of the system with various values of **Q** and **R**. This technique allows explicit control of different variables. Even in cases where a loss function is not known, this gives better results than pole placement techniques.

Kalman Filter: The state estimation techniques discussed earlier assumed that accurate measurements are available. In some cases, this is not true. A Kalman filter allows estimation despite noise in the measurements. This topic discusses the Kalman filter with constant gains, which is called a stationary Kalman filter. The gains are calculated off-line. The structure of the filter is the same as the observer that was discussed earlier, except that a different procedure is used to calculate the observer gain matrix **L**. It is also possible to implement a Kalman filter with time-varying gains; in which, case the structure becomes similar to an adaptive control or system identification problem.

The plant can now be described with an equation given as

$$\hat{\mathbf{x}}(n+1) = \mathbf{A}[\hat{\mathbf{x}}(n)] + \mathbf{B}[\mathbf{u}(n)] + \mathbf{w}(n) \tag{45}$$

where

x, **u**, **A**, and **B** are as described earlier. **w**(n) is process noise or a disturbance acting upon the plant.

The output is given by

$$y(n) = \mathbf{C}[\hat{\mathbf{x}}(n)] + \mathbf{v}(n) \tag{46}$$

where

$y, \hat{\mathbf{x}}$, and **C** are as described earlier. **v**(n) is noise resulting from the sensor and/or acting upon the measurement. The inputs **w**(n) and **v**(n) are assumed to be unrelated and to have Gaussian distributions.

Figure 24. Step Response and Control Effort of the LQR

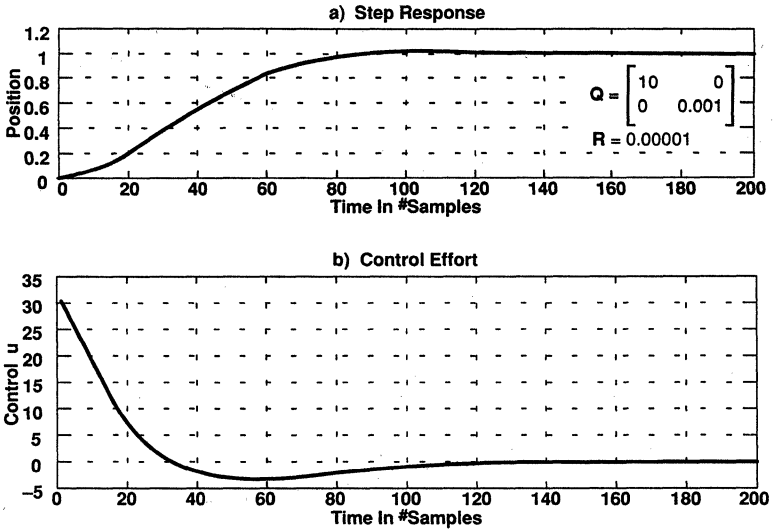


Figure 25. Step Response and Control Effort of the LQR

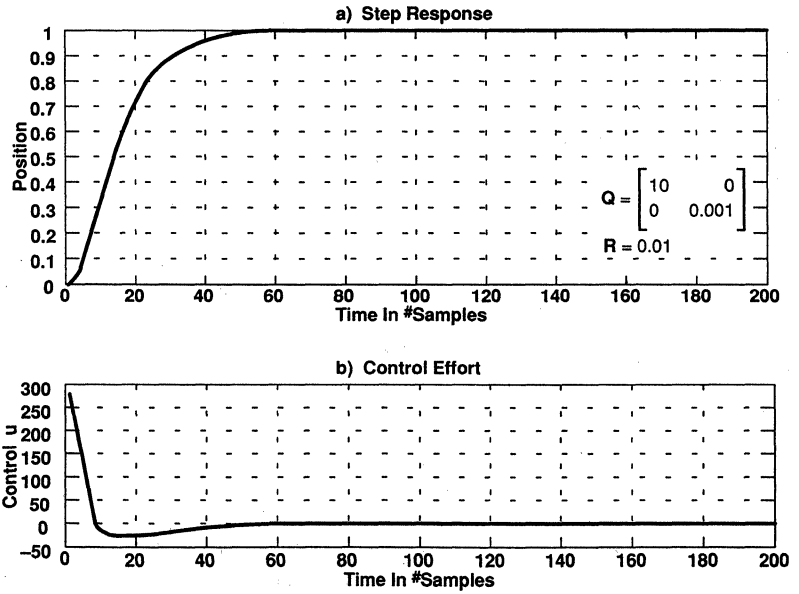


Figure 26. Step Response and Control Effort of the LQR

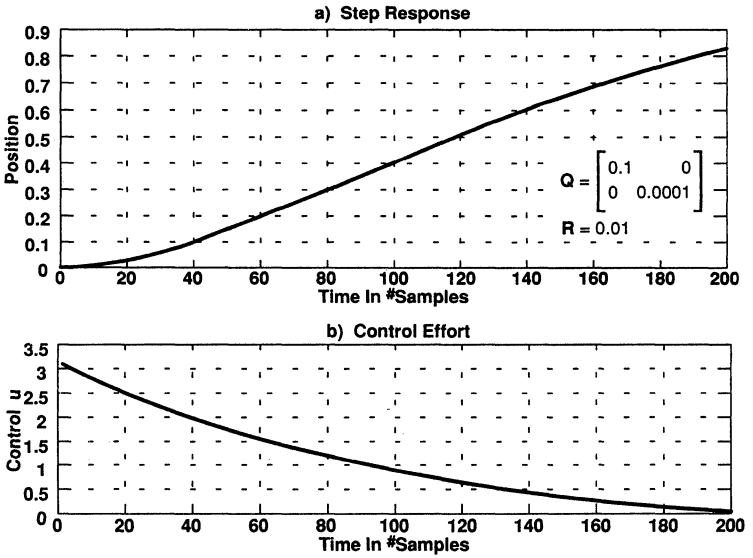
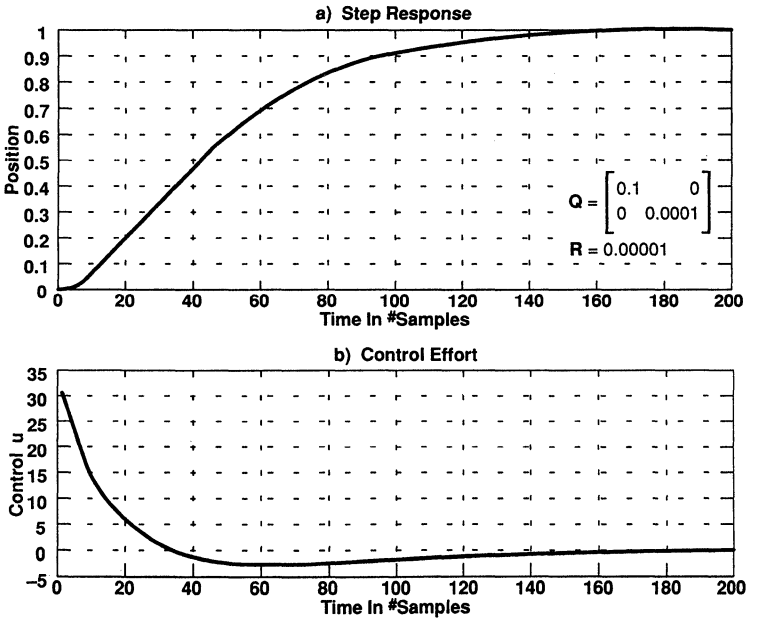


Figure 27. Step Response and Control Effort of the LQR



The Kalman filter requires minimizing the mean square estimation error. This is given by

$$J = \sum[e(n)e^T(n)] \tag{47}$$

where

e(n) is the estimation error given by

$$e(n) = x(n) + \hat{x}(n)$$

In general, the design equation for a steady-state Kalman filter is expressed as

$$\hat{x}(n+1) = \bar{x}(n+1) + L_p\{[y(n+1)] - C[\bar{x}(n+1)]\}$$

$$\bar{x}(n+1) = A[\hat{x}(n)] + B[u(n)]$$

This is also referred to as a current estimator because the latest measurement y(n+1) is used for the estimation error. When the latest measurement y(n+1) is made, $\bar{x}(n+1)$ will then be precomputed and updated. The estimator gain L_p is the Kalman gain. It minimizes the mean square estimation error and is represented by

$$L_p = PC^T(R_v + CPC^T)^{-1}$$

where

$$P = R_w + APA^T - APC^T(R_v + CPC^T)^{-1}CPA^T$$

Matrices R_w and R_v are known as covariance matrices and must be designed. They are usually selected as diagonal matrices because there is no information on cross-correlation of the noise elements. The rms value of the sensor noise can be directly used in the measurement covariance matrix R_v. The values given for R_w and R_v are chosen by using the facilities of PC–Matlab and the function **dlqe.**

The Kalman gain L_p is obtained by

$$L_p = \mathbf{dlqe}(A,G,B,R_v,R_w)$$

where

$$L_p = 1e^{-6}\begin{bmatrix} 0.3162 \\ 0.00003 \end{bmatrix}$$

and

$$R_w = \begin{bmatrix} .0001 & 0 \\ 0 & .000001 \end{bmatrix} R_v = 0.1$$

Figure 28 shows a block diagram of the Kalman Filter.

Figure 28. Kalman Filter

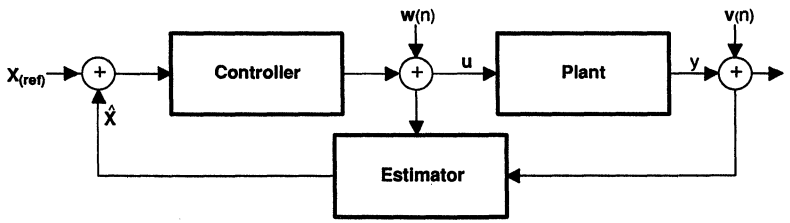
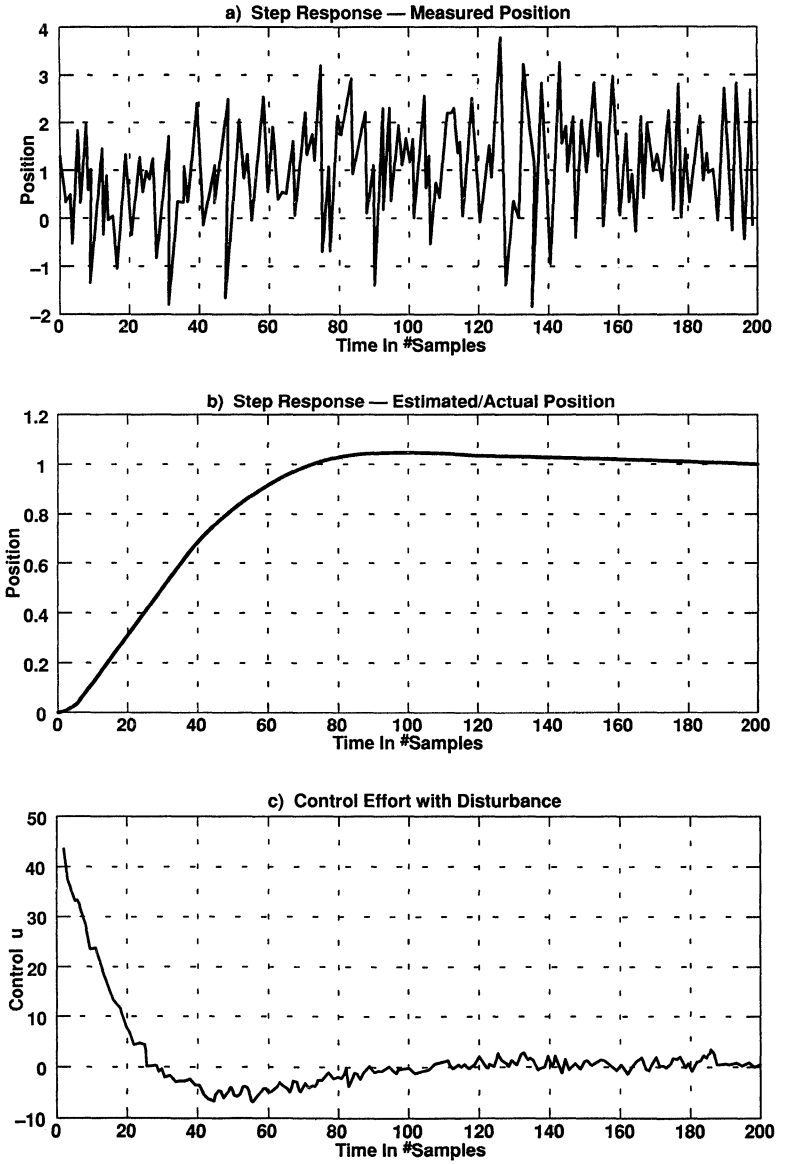


Figure 29 (a, b, and c) shows the response of a Kalman filter due to sensor noise and disturbance effects.

Figure 29. Response of the Kalman Filter



Implementation Considerations: Implementation considerations for LQR controllers and Kalman filters are not essentially different from those for state controllers and estimators. Their structures are the same, only the design approach is different. Still, the following should be taken into account.

When designing an LQR controller, some weight should be placed on the **R** matrix as control signals could become excessively large. Note that the LQR approach to controlling does not necessarily guarantee that the optimum solution will be found. Still, the **Q** and **R** matrices do allow the designer to trade-off between control effort and speed of the response while, at the same time, guaranteeing a stable system.

When designing a Kalman filter, $\mathbf{R}_v$ can usually be chosen realistically since some information on sensor characteristics and accuracy is available from the manufacturer. $\mathbf{R}_w$ is more difficult to chose. If it is chosen to be zero due to lack of information, the Kalman filter's gain is zero; the estimator runs open-loop. As a result, no adjustment is made to the estimated states. This causes the model to slowly drift. Again, as in the case of **Q** and **R** for the LQR controller, the designer can trade-off between reliability of measurements and plant model. As $\mathbf{R}_w$ increases, more reliance is placed upon the measurements, while less reliance is placed upon the plant model. As $\mathbf{R}_v$ increases, more reliance is placed upon the plant model, and less reliance is placed upon the measurements.

Summary

This paper has given a basic overview of digital control theory without going into too much mathematical detail. The use of CAD tools like PC–Matlab, Matrix–X, and Simnon is strongly recommended in order to eliminate some of the drudgery in the math calculations and to provide simulation of a system under design.

The choice of the appropriate controller structure will depend largely upon the user's background and application. Classical control techniques have been practiced for a long time, and people have acquired an intuitive feel of the behavior of those designs. Modern control theory now gives more capabilities to these systems; but, at the same time, most of the theoretical/implementation information is still fairly new and unfamiliar.

However, it should be emphasized that the behavior of any system in actual practice largely depends upon the implementation and not upon the elegance of its design. Elegant theories are attractive; but a simple design, when properly implemented, can yield a more superior performance, higher reliability, and better manufacturability than a sophisticated design that is poorly implemented.

In general, it is advised that modern control theory be used. With their powerful simulation capabilities, today's new CAD design tool can eliminate much of the user's fear and uncertainty, along with the laborious mathematical calculations. At the same time, powerful processors like DSPs are able to implement complex designs in practical and cost-effective systems.

References

1. Åström, K., and Wittenmark, B., *Computer Controlled Systems*, Prentice–Hall, 1984.
2. Phillips, C., and Nagel, H., *Digital Control Systems*, Prentice–Hall, 1984.
3. Isermann, R., *Digital Control Systems*, Springer–Verlag, 1981.
4. Franklin, G., Powell, D., and Workman, M., *Digital Control of Dynamic Systems*, Addison–Wesley, 1990.
5. Jacquot, R., *Digital Control Systems*, Marcel Dekker, 1981.
6. Katz, P., *Digital Control Using Microprocessors*, Prentice–Hall, 1981.
7. Lewis, F., *Optimal Control*, John Wiley, 1986.
8. Lewis, F., *Optimal Estimation*, John Wiley, 1986.
9. Åström, K., and Hagglund, T., *Automatic Tuning of PID Controllers*, Instrument Society of America, 1988.

Appendix 1

```
%  This program will do simulation of a PID controller using
%  trapezoidal approximation and a pole placement technique
%
%   If the plant transfer function is G(z) = A/B
%
%   and controller function is given by H(z) = C/D
%
%   then the closed loop response is given by
%
%       G(z)H(z)        AC
%      ---------- = --------
%    1 + G(z)H(z)   AC + BD
%
ggg=1
while ggg==1            % run simulation continously
%
% This section will implement simulation of a dc servo motor
% the motor used in the example is a Pittman motor, model 9412
%
Kt=0.0207;          % Torque constant
Ke=Kt;              %
j=0.00006;          % Armature inertia + assumed load inertia
R=6.4;              % Resistance
input('input sampling period in milliseconds')
T=ans/1000;         % get sampling period
a=(Kt^2)/(R*j)      % a, and b will give transfer function in s-domain
b=Kt/(j*R)
pause
ab=b/(a^2);         % Calculate values to transfer into z-domain
c=exp(-a*T);
d1=a*T;
d=(c-1+d1);
e=(1-c-(c*d1));
input('input numerator gain    ')
Kg=ans;              % get numerator gain
b1=ab*d*Kg;          % numerator terms
b2=ab*e*Kg;
a1=-(1+c);           % denominator terms
a2=c;
num=[0 b1 b2]        % numerator of transfer function in z-domain
den=[1 a1 a2]        % denominator of transfer function in z-domain
[A,B,C,D]=tf2ss(num,den)
%
```

```
% This section will design a PID controller using pole placement
% techniques. Desired pole locations have to be input. The PID
% is converted into discrete form using trapezoidal approximation
%
% Enter desired pole locations in the next step
'Enter the location of your poles'
input('Input location of pole 1:    ')
p1=ans;
input('Input location of pole 2:    ')
p2=ans;
input('Input location of pole 3:    ')
p3=ans;
input('Input location of pole 4:    ')
p4=ans;
p=[p1 p2 p3 p4];
% The desired characteristic polynomial is found as
Q(1:5)=poly(p)
% The coefficients of different powers are given by
q2=Q(:,2);
q3=Q(:,3);
q4=Q(:,4);
q5=Q(:,5);
% The system polynomial is given by
% (K1z**2 + K2*z + K3)(b1*z + b2) + (z - 1)(z - r)(z**2 - a1*z +a2)
% Equating coefficients of different powers we get
% four linear equations. The next few steps will solve for
% K1, K2, K3 and r, where r is an arbitrary location of one of the
% poles of the controller.
D = [ b1      0     0      -1
      b2      b1    0      1-a1
      0       b2    b1     a1-a2
      0       0     b2     a2 ];
%
D1= [ q2+1-a1  0     0      -1
      q3+a1-a2 b1    0      1-a1
      q4+a2    b2    b1     a1-a2
      q5       0     b2     a2 ];
%
D2= [ b1     q2+1-a1     0      -1
      b2     q3+a1-a2    0      1-a1
      0      q4+a2  b1     a1-a2
      0       q5    b2     a2 ];
%
D3= [ b1      0        q2+1-a1       -1
      b2      b1       q3+a1-a2   1-a1
      0       b2       q4+a2    a1-a2
      0       0     q5     a2 ];
%
D4= [ b1      0     0     q2+1-a1
      b2      b1    0     q3+a1-a2
      0       b2    b1    q4+a2
      0       0     b2     q5 ];
```

```
d=det(D);
d1=det(D1);
d2=det(D2);
d3=det(D3);
d4=det(D4);
K1=d1/d
K2=d2/d
K3=d3/d
r=d4/d
%     This section will implement closed loop simulation of
%     PID controller and the DC motor
%
num1=[K1 K2 K3];     % numerator of PID controller
R1=[1,r];            % poles of the PID controller
den1=poly(R1);       % calculate denominator
compnum=num1;
compden=den1;
procnum=num;
procden=den;
num5=conv(num1,num);  % Multiply numerators
den5=conv(den1,den);  % Multiply denominators
input('specify the time in secs over which you want to see the step: ')
t=ans;
n=t/T;               %    Calculate number of samples to see simulation
input('input a loop gain: ')  % Enter any additional loop gain
g=ans;
u=ones(n,1);          % Number of samples to see simulation
closnum=g*num5        % numerator of closed loop system transfer function
closden=g*num5+den5   % denominator of closed loop system trnasfer function
y=dlsim(closnum,closden,u);    % do discrete simulation
plot(y)
title('Position Step Response')
xlabel('Time in # of samples')
ylabel('Position in radian')
grid
pause
end
```

Appendix 2

```
%  This file will do simulation of a closed loop deadbeat controller
%
%   If the plant transfer function is G(z) = A/B
%
%   and controller function is given by H(z) = C/D
%
%   then the closed loop response is given by
%
%       G(z)H(z)        AC
%      ---------- = --------
%    1 + G(z)H(z)   AC + BD
%
ggg=1
while ggg==1       %  Keep doing
%
% The next section will implement simulation of a dc servo motor
% the motor used in the example is a Pittman motor, model 9412
%
Kt=0.0207;         % Torque constant
Ke=Kt;             %
j=0.00006;         % Armature inertia + assumed load inertia
R=6.4;             % Resistance
input('input sampling period in milliseconds')
T=ans/1000;        % get sampling period
a=(Kt^2)/(R*j)     % a, and b will give transfer function in s-domain
b=Kt/(j*R)
pause
ab=b/(a^2);        % Calculate values to transfer into z-domain
c=exp(-a*T);
d1=a*T;
d=(c-1+d1);
e=(1-c-(c*d1));
input('input numerator gain   ')
Kg=ans;             % get numerator gain
b1=ab*d*Kg;         % numerator terms
b2=ab*e*Kg;
a1=-(1+c);          % denominator terms
a2=c;
num=[0 b1 b2]       % numerator of transfer function in z-domain
den=[1 a1 a2]       % denominator of transfer function in z-domain
[A,B,C,D]=tf2ss(num,den)
%
% This section will implement design of a deadbeat controller
% The form of the controller is given by the following equation
%
%                    -1       -2       -3             -n
%         p0 + p1*z    + p2*z    + p3*z  ........pn*z
%   G  (z) = ------------------------------------------
%    db             -1       -2       -3             -n
%          q0 + q1*z    + q1*z    + q3*z  ........qn*z
%
```

```
%    If the plant transfer function is given by
%
%                    -1        -2        -3            -n
%          b0 + b1*z    + b2*z    + b3*z  ........bn*z
%    G  (z) = ------------------------------------------
%     p              -1        -2        -3             -n
%          a0 + a1*z    + a2*z    + a3*z  ........an*z
%
%    then the following procedure can be used to design a
%    deadbeat controller
p0 = 1/(1 + b1 + b2)
p1 = a1*p0
p2 = a2*p0
q0 = 1
q1 = -b1*p0
q2 = -b2*p0
%
%   This section will implement closed loop simulation of the
%   deadbeat controller and the DC motor
%
num1=[p0 p1 p2];    %  Numerator of the controller
den1=[q0 q1 q2];    %  denominator of controller
compnum=num1;
compden=den1;
procnum=num;
procden=den;
num5=conv(num1,num);    % multiply both numerators
den5=conv(den1,den);    % multiply both denominators
input('specify the time in secs over which you want to see the step: ')
t=ans;
n=t/T;                % Calculate number of samples to see simulation
input('input a loop gain: ')
g=ans;
u=ones(n,1);
closnum=g*num5;         % Enter additional closed loop gain
closden=g*num5+den5;    % Calculate denominator of closed loop system
y=dlsim(closnum,closden,u); % Do closed loop simulation
plot(y)
title('Position Step Response')
xlabel('Time in # of samples')
ylabel('Position in radian')
grid
pause
end
```

Appendix 3

```
%  This file will do simulation of a closed looped system with
%  a DC servo motor and a state controller/estimator. The estimator
%  will make a full estimate of states from position measurement>
%  The state controller is given by the following equations
%
%  x(n+1) = A*x(n) + B*u(n) + L[y(n) - C*x(n)]   --- State estimation
%  y =  C*x(n)  ------ estimation of measured variable
%  u = -K*x(n)  ------ control law
%
%  States of the system will be position and velocity
%
aaa=1
while aaa==1    % Do simulation continously - to exit do CTRL C
%
%  The first section will build model of dc motor
%  The motor used in this example is a Pittman motor, model 9412
%
clear;
Kt=0.207;       % Torque constant
Ke=Kt;          % Back emf constant
j=0.0006;       % Armature inertia + assumed load intertia
R=6.4;          % resistance
a=(Kt^2)/(R*j)  % a, and b will give transfer function in s-domain
b=Kt/(j*R)
pause
num=[0 1 b]     % define numerator and denominator of transfer function
den=[1 a 0]
pause
F=[0 1          % state representation of motor in continous time
   0 -a]
G=[0
   b]           % convert state model to discrete form
input('Input sampling period in milliseconds   ')
T=ans/1000;     % get sampling period
[A,B]=c2d(F,G,T)
C=[1 0]         % Assume position measurement
%
%   The next section will implement design of the state controller
%   and observer using pole placement techniques. Pole locations will
%   have to be input for the controller. The estimator poles will be
%   chosen to faster than the controller.
%
' Enter 0 if you will have complex poles'
input(' and 1 if you will have real poles:      ')
X=ans;
if X==0
input('input real part of pole location:   ')
r1real=ans;
input('input imaginary part of pole location:   ')
r1imag=ans;
i=sqrt(-1);
r=[r1real+i*r1imag; r1real-i*r1imag];
end
```

```
if X==1
input('input location of pole 1:    ')
r1=ans;
input('input location of pole 2:    ')
r2=ans;
r=[r1; r2];
end
K=place(A,B,r)      % do pole placement for controller
l=r/2               % choose observer poles to 1/2 distance from origin
ll=place(A',C',l)   % do pole placement for observer
L=ll'
%
%  The next section will do simulation of the closed loop system
%
D=[0]               % direct link matrix is 0
input('input reference signal in radians:  ')
re=ans;
N=[1;0]      % position command will be input
xr=N*re      % reference state
input('specify time in sec over which you want to see step:  ')
t=ans;
n=t/T;       % calculate number of samples
x=[0;0]      %  actual states - initial values
xe=[0;0]     %  estimated states - initial values
u= 0         %  control action - initial value
%
%   This section will do simulation of the motor
%
for i=1:n,
x = A*x + B*u;    %   simulation of actual plant
y(i)= C*x;
%   This section  will do simualtion of the controller and estimator
%
u = -K*xe + K*xr; %  implement control law
yu(i) = u;
xe = A*xe + B*u + L*(y(i)-C*xe);   % do state estimation
ye(i) = C*xe;      % estimated postion
end
clg
plot(y)             % plot actual postion
hold on
plot(ye,'+g')       % plot estimated position
ylabel('Postion')
xlabel('Time in # of samples')
title('Step response of State Controller/Estimator')
text(0.60,0.40,'---- actual postion','sc')
text(0.60,0.30,'**** estimated postion','sc')
grid
pause
hold off
clg
subplot(211),plot(y),title('Step response - Actual Position'),
subplot(212),plot(ye),title('Step response - Estimated Position'),
```

```
pause
plot(yu),title('Control effort'),
grid
ylabel('u')
xlabel('Time in # of samples')
end
```

Appendix 4

```
%   This program will do simulation of Linear Quadratic Regulator (LQR)
%   and a stationary Kalman Filter.
%   The controller and estimator are given by the following equations:
%   x(n+1) = A*x(n) + B*u(n) + Lp[y(n) - C*x(n)] --- State equation
%   y = C*x(n)    --- estimation of measured variable
%   u = - K*x(n) --- control law
%   K is optimal gains and Lp is kalman gains
%
aaa=1
while aaa==1   % run simulation simultaneously - to exit use CTRL C
%
%   This section will build model of a dc servo motor
%   The motor used in the example is a Pittman motor, model 9412
%
clear;
Kt=0.207;        % Torque constant
Ke=Kt;           % back e.m.f. constant
j=0.0006;        % rotor inertia + assumed load inertia
Res=6.4;         % resistance
a=(Kt^2)/(Res*j)
b=Kt/(j*Res)
F=[0,1;0,-a]     % state representation in continous time
G=[0;b]
input('Input sampling period in milliseconds   ')
T=ans/1000;      % get sampling period
[A,B]=c2d(F,G,T) % convert state model to discrete time
C=[1 0]          % Assume position measurement
%
%   The next section will design the LQ Regulator and the
%   Kalman filter. The cost functions will be input to
%   to calculate the optimal gains and noise characteristics
%   will be input to calculate Kalman gains
%
input('enter cost function matrix Q:')
Q=ans;
input('enter cost function R:')
R = ans;
input('enter measurement noise covariance Rv:')
Rv=ans;
input('enter disturbance matrix g:')
g=ans;
input('enter disturbance covariancce matrix Rw:')
Rw=ans;
K=dlqr(A,B,Q/T,R*T)        % calculate optimal gains
Lp=dlqe(A,g,C,Rw*T,Rv/T)          % calculate Kalman gains
pause
%
```

```
%   The next section will do simulation of the closed loop
%   system
%
D=[0]                              % no direct link input
input('input reference signal in radians:  ')
re=ans;
N=[1;0]          % position command will be assumed
xr=N*re          % reference state
input('specify time in sec over which you want to see step:  ')
t=ans;
n=t/T;           % calculate number of samples to do simulation
x=[0;0];         % actual states - initial values
xe=[0;0];        % estimated states initial value
yv= rand('normal'); % characteristics for injected sensor noise
yv=rand(n,1);
uv=rand('normal');  % characteristic for disturbance noise
uv=rand(n,1);
u=0;             % control signal - initial value
%
%   Next section will do simulation of the motor
%
for i=1:n,
x = A*x + B*u;                     % simulation of actual plant
y(i)= C*x + yv(i,1);             % measured position
%
%   Next section will simulate regulator and kalman filter
%
u = -K*x + K*xr + uv(i,1);      % control action with disturbance
yu(i) = u;
xe = A*xe + B*u + Lp*(y(i)-C*xe); % state estimator
ye(i) = C*xe;                      % estimated position
end
clg
plot(y,'r');                       % plot actual position
hold on
plot(ye,'.g')                      % plot estimated position
title('Measured position vs Estimated postion')
grid
text(0.60,0.24,'---- measured postion','sc')
text(0.60,0.18,'.... estimated postion','sc')
xlabel('Time in # of samples')
ylabel('position')
pause
hold off
clg
plot(y),title('Step response - Measured Position'),
grid
xlabel('Time in # of samples')
ylabel('position')
pause
clg
plot(ye),title('Step response - Estimated Position'),
grid
xlabel('Time in # of samples')
```

```
ylabel('position')
pause
clg
plot(yu),title('Control effort with disturbance'),
xlabel('Time in # of samples')
ylabel('Control u')
grid
end
```

Matrix Oriented Computation Using *Matlab*

Jeffrey C. Kantor
Department of Chemical Engineering
University of Notre Dame
Notre Dame, IN 46556

Phone: (219) 239 5797
Email: jeff@ndcheg.cheg.nd.edu
Fax: (219) 239 8007

Matlab is a tool for interactive numerical computation. It contains as built-in functions essentially all of the numerical linear algebra algorithms in LINPACK and EISPACK. Coupled with a programmable interpreter and good scientific graphics capability, Matlab can be used for algorithm development in many areas of engineering and science.

To demonstrate some of its functionality, I've included in this article several examples where Matlab has proven useful in my own teaching and research activities. These examples are not comprehensive since they neither fully exploit all of the features of Matlab or do they show all of our applications. The examples were chosen only because they seemed to be relatively straightforward and self-contained illustrations of how Matlab can be used.

1 Some Background

Matlab was originally conceived by Cleve Moler just over a decade ago while he was teaching numerical methods at the University of New Mexico. He found it frustrating to simultaneously teach numerical methods and the programming tricks it takes to implement them. The effort required to write numerically sophisticated FORTRAN code can simply overwhelm a student and not leave much time left over for doing applications. So to address the problem, Cleve Moler wrote a simple interpreter in portable FORTRAN for a high-level matrix oriented language. The interpreter was based on one given by N. Wirth for a model language called PL/0 [12]. Naturally, the numerical algorithms were based on the recently completed Linpack and Eispack projects to which Cleve Moler had made substantial contributions. This primitive Matlab interpreter was evidently quite successful and ported to a number of machines during the late 1970's and early 1980's, undergoing minor revisions in the process.

Several companies subsequently adopted Matlab as a platform for developing and delivering commercial control synthesis and analysis software. Systems Control Technology produced a package called Control-C, and at about the same time, Matrix-X was developed by Integrated Systems, Inc. Both companies found many shortcomings in the original Matlab interpreter including workspace constraints, lack of function definitions, and overall performance. The Matlab interpreter was largely rewritten at each of these companies to support their products.

A few of the professional staff from these companies joined together to form a new company called the MathWorks, Inc. There they produced an entirely new version of Matlab written in C for portability and efficiency. The interpreter was greatly enhanced to include an ability for the user to program Matlab functions. They also developed an integrated facility for producing a basic set of publication quality scientific graphs. The MathWorks currently markets this version of Matlab for a variety of hardware platforms, the details are given at the end of this article.

Beyond the basic interpreter, there are several 'toolboxes' intended for specific application areas. A 'toolbox' is typically a collection of functions and scripts that implement specialized numerical algorithms. These generally are not finished applications in the sense of a well-developed user interface with a lot menus and the like, but are rather integrated collections of algorithms that you either can use directly or build into your own scripts. It is sort of like using a FORTRAN subroutine library, but with the advantage of being able to directly execute the routines in the interactive Matlab environment. The MathWorks distributes a Signal Processing Toolbox with Matlab, and markets several others including a Control Design Toolbox, Robust Control Toolbox, System Identification Toolbox, a Chemometrics Toolbox. There are also toolboxes commercially available from third parties, in addition to a number that University researchers may have put together for their

own purposes.

Now for the confusing part. There is a 'public domain' IBM PC version of Matlab. In addition, several universities sell very low cost versions of Matlab available for the Macintosh and IBM PC. These are based on Moler's original FORTRAN code, sometimes with enhanced graphics and macro writing facilities.[1] A person should be careful with these since they are not of the same calibre as the MathWorks and simply don't include the tools necessary for doing real work. Nor will the toolboxes cited above work with these versions. A corollary of this advice is to not let an exposure to these other versions color your view of Matlab.

2 What is Matlab?

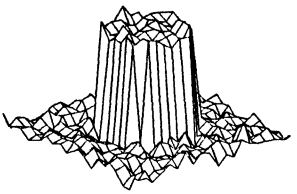
In some ways, the Matlab interpreter vaguely resembles a cross between BASIC and APL in the sense that it is programmable and endowed with a rich set of operators for matrix manipulations. The key distinction is that Matlab incorporates well-developed and reliable algorithms for numerical linear algebra. Moreover, the built-in graphics capability is often entirely sufficient for presenting results in final published form. (The graphics in this article, for example, were pasted in directly from Matlab).

Let me give an example of how these capabilities can be used for day-to-day 'scratchpad' kind of calculations that pop up. A few days ago a colleague of mine walked into my office with an idea for processing video images to enhance the edges of discs that appear in the picture. He acquires these images in his experiments on concentrated suspensions of noncolloidal particles. He started off by saying (roughly) "Suppose you have a noisy image of a disc " at which point I stopped him, turned on my computer, and typed the following commands in Matlab

```
x = -1:.1:1;                % X mesh
y = -1:.1:1;                % Y mesh
[xx,yy] = meshdom(x,y);     % 2D mesh

z=sqrt(xx.^2+yy.^2)<0.5; % make disk
rand('normal');          % white noise
z = z + 0.05*rand(z);    % add to disk
mesh(z);                 % 3D plot
xlabel('Noisy Disk');    % add title
```

which produced the image shown in Figure 1.

This code segment demonstrates several of the key features of Matlab. First of all, the variables `x`, `y`, `z` represent vectors and matrices. Matrices are an elementary data type within Matlab. Because matrices can be manipulated directly as single objects, much of tedium of writing loops to do element by element calculations is removed, along with the need for a lot of extraneous indexing. In the sixth line, for example, a matrix is constructed with the same dimensions as `z` consisting of normally distributed random numbers (`rand(z)`), multiplied by 0.05, and the result added to `z`. The third line demonstrates how Matlab functions can return multiple results, which in this case are two matrices `xx` and `yy`.



Noisy Disk

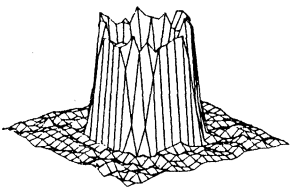
Figure 1: Noisy image of a disk.

Duly impressed, my collcague went on at the blackboard to describe a simple algorithm requiring that the image be processed by a pair of 2D convolutions. Since this might be done more than once to different data sets, it seemed sensible to encapsulate the algorithm as a Matlab function.

```
function [y] = sobel(z)

% SOBEL
%  Do edge detection on a 2D array

s = [1 2 1; 0 0 0; -1 -2 -1];
h = conv2(z,s);      % 2D convolution
v = conv2(z,s');     % 2D convolution
y = sqrt(h.^2 + v.^2);
```

A function is prepared as a separate text file that is subsequently read by the Matlab interpreter when its name is encountered in a command line. A user written function behaves in the same way as any built-in function. In this example, a function named sobel is defined which takes a single input argument `z`, then utilizes a built-in Matlab function `conv2` to construct

[1] Incidently, the original FORTRAN code was never declared to be public domain. Thus its ownership status is a bit confused.



Edge Filtered Disk

Figure 2: The result of filtering the noisy disk shown in Figure 1 to enhance the disk edges.

two 2D convolutions with a matrix, `s`, and its transpose, `s'`. The function output, `y`, is found by taking the harmonic mean of the two convolutions.

The edge detection function was used in the following commands

```
zf = sobel(z);
mesh(zf);
xlabel('Edge Filtered Disk');
```

to produced the edge enhanced picture shown in Figure 2.

In general, functions can have multiple-input and multiple-output arguments. Just as in FORTRAN, any variables used in writing a function are treated as local and will not be confused with other variables of the same name used in other functions or the command environment.

So during the course of a half-hour conversation, my colleague was able to (watch me) construct and test an edge detection algorithm. It is this ability to quickly prototype and test algorithms using a rich base of numerical tools that makes Matlab a valuable computational tool.

3 Using Matlab in the Classroom

I have used Matlab in teaching a graduate course on Process Control (Fall, 1987), the linear algebra portion of a course covering Mathematical Methods for first-year graduate students (Fall 1988 and Fall 1989), and for a Junior-level course on Computer Methods for Chemical Engineers (Spring, 1989). Matlab seems to provide an appropriate software base for each of these courses.

In the case of teaching Advanced Process Control, the main goal in using Matlab was to provide the student with experience in doing time-series analysis, model identification, control design, and simulation. There are competing software packages that could also be used for these purposes, among them Program-CC, but none seemed to offer any significant advantage over Matlab for linear analysis. Besides, my teaching assistant had already had some experience with Matlab, and it was already installed on several Sun workstations in the Department. Overall, the teaching experience was a very good one. By the end of course the students demonstrated a real facility with Matlab, the Control Design Toolbox, and the System Identification Toolbox. Later in the article there is an example that came from a homework problem assigned in the course.

For the undergraduate Computer Methods course, there were additional considerations that came up when considering a choice of software tools. Among them was the choice between using Matlab or a package of FORTRAN subroutines such as given in Press, et al. [11]. On the one hand, FORTRAN remains as the principle programming language for numerically intensive engineering applications, therefore a facility with FORTRAN is highly desirable. Moreover, our students all take a required Freshman Engineering course that teaches the elements of FORTRAN.

On the other hand, it is significantly faster to write and test small codes using the high-level Matlab interpreter. The students also indicated a strong preference for microcomputer based software tools which could be used on various workstation clusters about campus rather than be tied to a single minicomputer located in the Engineering College.

On balance, I felt that a more productive environment would allow the course to survey more topics with more emphasis on applications, so I chose to use Matlab. I have been pleased to note how students have transfered their new computational skills to other courses. They continue to use Matlab to do routine laboratory calculations, data fitting, and for computations in their Senior Design courses.

Recent textbooks have appeared which incorporate various amounts of Matlab into the text and exercises. The third edition of the classic linear algebra text by Noble [10] contains is number of Matlab exercises and examples. Another linear algebra textbook by Hill is basically centered on Matlab, with chapters regarding programming technique [7]. It is so complete that it could serve as a low-cost Matlab manual for students. The *Handbook for Matrix Computations* is useful to

anyone doing numerical linear algebra, and includes a survey of relevant Fortran, BLAS, Linpack, as well as Matlab [4].

Lennert Ljung's book on system identification [9] is closely coupled to the System Identification Toolbox. The toolbox, in fact, was written by Ljung, and the text provides excellent technical documentation.

The following two sections present two examples of incorporating Matlab into classroom activities.

4 Classroom Example: Linear Programming

Three years ago our Department introduced a new required course for our undergraduate majors entitled *Computer Methods for Chemical Engineers*. This course is normally taken by Spring semester Juniors after having completed the normal Mathematics sequence, and before commencing the two-semester Senior design sequence. The course covers elements of numerical methods with application to problems in chemical engineering.

Linear programming is discussed in some detail in the course because it is one of those skills that an engineer can transfer to a wide variety of problem areas. A key teaching goal is for the student to be able to recognize a problem as a linear program, and then to formulate the requisite objective and constraints.

I prefer to use the Active Set method as outlined by Fletcher [5] to teach the principles behind linear programming. It seems to leave the student with a more intuitive understanding of the role of constraints and their sensitivities than does the usual presentation of the Simplex method. If the students can understand the relatively simple strategy to solving a linear program, it is then much easier to motivate and teach the numerical tricks it takes to implement an efficient algorithm.

The linear programming problem is formulated as minimizing the linear objective

$$\min_x z = c^T x$$

where x is a n vector, subject to m linear constraints

$$a_i x \geq b_i \quad i = 1, 2, \ldots, m$$

where $n \leq m$. If positivity constraints are present, then these are explicitly included in the constraint list. It is easy to show that if the feasible region is bounded, then optimum will always be found at a vertex defined by the intersection of n active constraints.

The basic algorithm is, firstly, to find any active set of n constraints forming a feasible vertex, then to move systematically from one vertex to another so as to reduce the value of the objective function at each step. Each step of the algorithm is defined by just two rules. The first rule identifies a constraint to throw out of the active constraint set in order to decrease the objective. The second rule determines which constraint to add to the active set to establish a new feasible vertex.

Let A be the set of active constraints that determine a feasible vertex. The vertex is given by solving a set linear equations to give

$$x = A_A^{-1} b_A$$

where A_A and b_A are constructed from the coefficients of the active constraints. Now suppose the right hand side of each active constraint is altered by a small positive amount ϵ_i. Positive values of the ϵ_i's correspond to feasible perturbations, while negative values would cause constraint violations. As a result of a feasible perturbation, the vertex then shifts from x to x_ϵ, where

$$x_\epsilon = A_A^{-1} b_A + A_A^{-1} \epsilon$$

Substituting x_ϵ into the objective function yields

$$z = c^T A_A^{-1} b_A + c^T A_A^{-1} \epsilon$$

The second term shows the change in the objective function due to independent perturbations in the active constraint set. Thus the elements of the row vector

$$\lambda = c^T A_A^{-1}$$

play the role of 'sensitivity coefficients' revealing how the objective function responds to feasible perturbations in the active constraint set. If any element of λ is negative, then the objective function can be reduced by removing that constraint from the active set. Just as in the Simplex method, we choose to remove the constraint corresponding to the most negative element of λ.

Let λ_p be the most negative element of λ. Then the effect of removing the p^{th} active constraint is given by

$$x_\epsilon = x + \epsilon_p s_p$$

where s_p is the p^{th} column of A_A^{-1}. How large can ϵ_p be before some other constraint becomes active? This can be computed explicitly as

$$\epsilon_p = \min_{\substack{i \notin A \\ a_i s_p < 0}} \frac{b_i - a_i x}{a_i s_p}$$

The search is done over all constraints not in the active set ($i \notin A$), but only for those constraints in

which the right hand side becomes smaller as ϵ_p increases ($a_i s_p < 0$).

The constraint which realizes the minimum ϵ_p is exactly the one to be added to the active constraint set. Having done that, the procedure repeats itself until no further improvement in the objective is possible, i.e., until all of the sensitivity coefficients are non-negative.

This basic algorithm cleanly translates to the following Matlab function. The function `lp` takes four arguments specifying the coefficients on the left and right hand sides of the constraints, coefficients of the objective function, and an initial feasible constraint set. The function returns the optimal value of the objective function, the optimal solution for the decision variables, the value of the sensitivity coefficients, and the final active constraint set.

```
function[z,x,lamb,activ]=lp(a,b,c,feas)

% Initialization

[m,n] = size(a);
activ = feas(:);

% Compute Initial Vertex

ainv = inv(a(activ,:));
x = ainv*b(activ,:);
lamb = c*ainv;

while any(lamb < 0),

  % Find which constraint to drop, p
  [tmp,p] = min(lamb);
  sp = ainv(:,p);

  % Find which constraint to add, q

  alpha = Inf;
  q = 0;
  for i=1:m,
    if ~any(i==activ),
      den = a(i,:)*sp;
      if den < 0,
        tmp = (b(i)-a(i,:)*x)/den;
        if tmp < alpha,
          alpha = tmp;
          q = i;
        end
      end
    end
  end
  % Recompute x, lamb, and z
  activ(p) = q;
  ainv = inv(a(activ,:));
  x = ainv*b(activ,:);
  lamb = c*ainv;

end

z = c*x;  % Compute objective function
```

This example uses several of the Matlab control structures to simplify the coding process. The construction

```
while any(lamb < 0),
   [...]
end
```

controls the main iteration over vertices of the feasible region. The iteration continues as long as any element of the vector lamb is less than zero. Nested within this loop is an iteration

```
for i=1:m,
   [...]
end
```

which specifies a conventional indexed iteration loop where `i` successively takes values between `1` and `m`. Within this loop are several nested conditional statements such as

```
if ~any(i==activ),
  [...]
end
```

In this case, the conditional code is executed if 'not any' of the elements of the vector `activ` are equal to `i`. The practice of indenting nested control structures graphically reveals program flow and is strongly urged on the students.

This function is a zeroth order cut at a practical algorithm for linear programming, it will work for small problems but will be inefficient and error prone when applied to larger problems. As exercises, the students are asked to correct several of the glaring deficiencies. Foremost is to avoid the repeated inversions of the active constraint matrix with a more efficient procedure using rank-one updates (i.e., the Sherman-Morrison formula). Having done this, the algorithm is then identical to the usual revised simplex method as discussed in most textbooks. Other exercises in algorithm development could include writing a code to identify an initial feasible constraint set, or to modify the algorithm to handle equality constraints.

5 Classroom Example: Process Control

The next example illustrates the use of several toolboxes to do model identification and a simple control design. Students taking a graduate course in Advanced Process Control during Fall, 1987, were assigned a homework project in which they were to analyze input-output data for a small gas furnace. They were to first obtain a transfer function model, then use the model to design a PID, minimum variance, and optimal LQG controllers. The three controllers were to be evaluated by simulation. The students were given one week to complete the assignment.

The gas furnace data was adapted from Appendix B of Box and Jenkins [2] consisted of 300 pairs of input-output measurements `{u(k), y(k)}` obtained at 9 second intervals. The manipulated input is gas flowrate, and the measured output is the percentage of CO_2 in the stack gas. These data were given to the students as a Matlab file called `GasFurnaceData`. The file can be read and plotted using the following commands to produce the following plots shown in Figure 3.

```
%     Read data record

GasFurnaceData;
udata = u;
ydata = y;

%     Plot input-output data

subplot(211);  % Specifies upper plot
plot(udata);
title('Gas Flow (Input)');
ylabel('CFM');

subplot(212);  % Specifies lower plot
plot(ydata);
title('CO2 Composition (Output)');
ylabel('% CO2');
xlabel('Time');
```

The first task for the students was to identify a discrete-time transfer function model for the gas furnace. A non-parametric spectral analysis provides a starting point for estimating model order. This is done with the following commands:

```
y = detrend(ydata);
u = detrend(udata);
z = [y u];
g0 = spa(z);        % System_ID toolbox
bodeplot(g0);       % System_ID toolbox
```

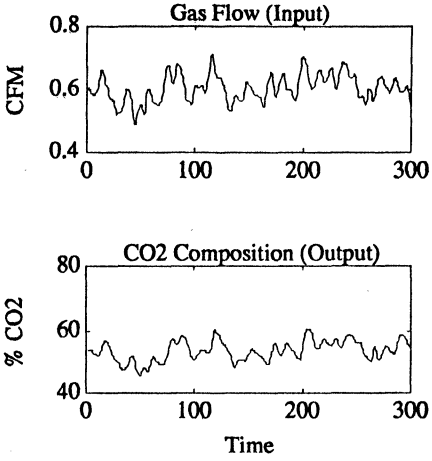


Figure 3: Input-Output data for a gas furnace. The data is adapted from Appendix B of Box and Jenkins.

The function `detrend` (from the Signal Processing Toolbox) is used to remove means and linear trends from the input and output data series. Then `spa` (also from the System Identification Toolbox) is applied to construct a transfer function estimate that is stored as `g0`. The transfer function is displayed using `bodeplot` to give the result shown in Figure 4.

There are a number of possible models that could be used to describe this data. Of these, an ARMAX model in the form

$$y(t) = \frac{B(q)}{A(q)}u(t-n_k) + \frac{C(q)}{A(q)}e(t)$$

or, explicitly, as

$$y(t) = \frac{b_1 q^{-1} + \cdots + b_{n_b} q^{-n_b}}{1 + a_1 q^{-1} + \cdots + a_{n_a} q^{-n_a}} u(t-n_k) + \frac{c_1 q^{-1} + \cdots + c_{n_c} q^{-n_c}}{1 + a_1 q^{-1} + \cdots + a_{n_a} q^{-n_a}} e(t)$$

does an adequate job (q^{-1} is the backward shift operator). The following commands use functions in the System Identification Toolbox to fit an ARMAX model for the case $n_a = n_b = n_c = 2$, $n_k = 1$. The fitted transfer function is then evaluated and Bode plot is displayed to compare the fitted transfer function to the previous non-parametric estimate.

```
th = armax(z,[2 2 2 1]);
```

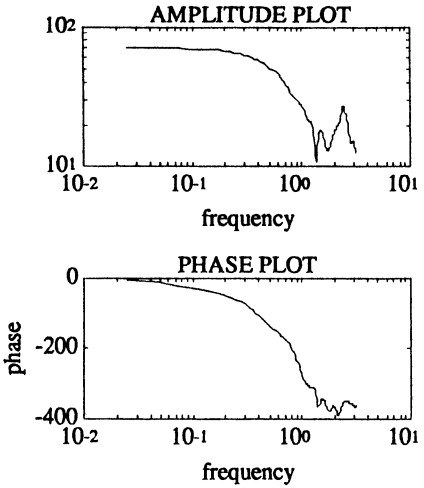


Figure 4: A nonparametric estimate of the transfer function between the input and output of the gas furnance based on the data in Figure 3. The results are computed using the System Identification Toolbox.

```
g = trf(th);
bodeplot([g g0]);
```

The resulting Bode plots shown in Figure 5 demonstrate a reasonable fit of the data using a second order model. 'Goodness of fit' can also be explored by computing an estimated autocorrelation function for the residuals, and an estimated cross correlation between the input. This is done with the command `e = resid(z,th)` to produce the results shown in Figure 6.

These plots indicate that there is little significant correlation left in the residuals so there is no statistical justification for employing higher order models. (Attempting to fit a first-order model to this data provides an example where statistically significant correlations do remain in the residuals.) The fitted model coefficients are displayed as follows:

```
present(th)
This matrix was created by the command
ARMAX on 2/28 1989 at 10:47

Loss fcn: 0.09217
Akaike's FPE:  0.09593
Sampling interval 1
```

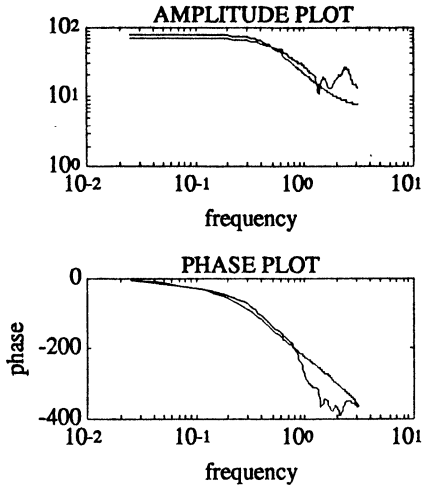


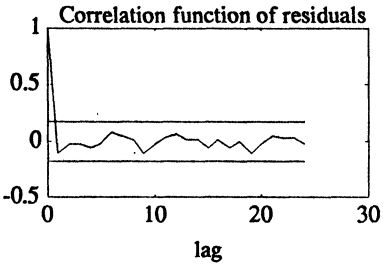
Figure 5: Comparison between the nonparametric estimate of the gas furnace transfer function, and the transfer function by fitting a second order ARMAX model. A good fit is obtained except at relatively high frequencies where noise is expected to be the dominant contribution.

```
The polynomial coefficients and their
standard deviations are

B =
         0   -6.3133   16.9243
         0    1.9007    2.3403

A =
    1.0000   -1.3899    0.5299
         0    0.0516    0.0460

C =
    1.0000    0.1385    0.1307
         0    0.0856    0.0659
```

At this point in the exercise, the student has developed a transfer function model for the gas furnace that can be used for designing simple control systems. Omitting the details, an optimal LQG controller can be designed to minimize the loss function

$$J_{lq} = E[y^2(k) + \rho u^2(k)]$$

by the computational method outlined in Chapter 12



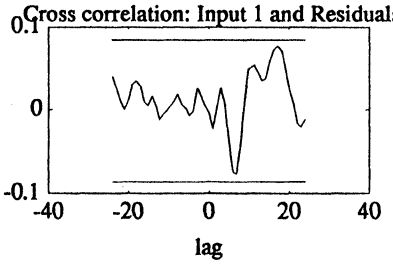


Figure 6: The auto- and cross-correlation functions of the residuals obtained after fitting a parametric model provides a simple test of model fit. In this case, a second-order ARMAX model appears to adequately account for all of the essential correlations in the gas furnace data. The horizontal lines mark the 95% confidence intervals for the null hypothesis.

of Astrom and Wittenmark [1]. The necessary calculations are encapsulated in the function `dlqg` given below. This function makes use of others defined in the Control Systems Toolbox. These are `dlqr`, which computes a solution to the algebraic discrete time Ricatti equation, and `ss2tf`, which converts a state-space model representation to a transfer function description.

```
function [s,r]=dlqg(th,rho)
%DLQG
%  [r,s] = DLQG(theta,rho) computes
%  the LQ optimal controller to
%  minimize the objective function
%
%           2          2
%        E[y (k) + rho*u (k)]
%
%  The resulting controller is given
%  in transfer function form
%
%                  S(q)
%          u(k) = - ---- y(k)
%                  R(q)
%
%  The plant model is given by theta
%  in the standard form of the System
%  Identification Toolbox.

% Ref:Chapter 12, Astrom & Wittenmark

% J.C. Kantor, 3 December 1987

  [a,b,c,d,f]=polyform(th);
  a=conv(a,f);
  na = length(a)-1;
  nb = length(b)-1;
  nc = length(c)-1;
  n = max([na,nb,nc]);
  A = [zeros(n,1),[eye(n-1);..
       zeros(1,n-1)]];
  A(1:na,1) = -a(2:na+1)';
  B = zeros(n,1);
  B(1:nb,1) = b(2:nb+1)';
  K = zeros(n,1);
  K(1:nc,1) = c(2:nc+1)';
  K = K + A(:,1);
  C = [1,zeros(1,n-1)];

  L = real(dlqr(A,B,C'*C,rho));

  [s,r] = ss2tf(A-K*C-B*L,K,L,[0],1);
```

Letting $\rho = 10^{-5}$ gives an approximation to minimum variance control. The resulting controller is given by $u(t) = -G_c(q)y(t)$ where

$$G_c = \frac{S(q)}{R(q)} = \frac{0.0762q^{-1} - 0.0512q^{-2}}{1 + 1.1555q^{-1} + 1.6364q^{-2}}$$

Finally, the student can compute the simulated response of the closed-loop gas furnace control system. The closed-loop transfer function between the output and exogenous disturbances $e(t)$ is given by

$$y(t) = \frac{C(q)R(q)}{A(q)R(q) + B(q)S(q)}e(t)$$

The following sequence of commands computes the products of polynomials using the Matlab convolution operator `conv`, does a simulation of the closed-loop plant models, and displays the results.

```
% Compute control and closed-loop
% transfer fuctions
```

```
[s,r] = dlqg(th,0.00001);
[a,b,c]=polyform(th);
p = conv(a,r) + conv(b,s);
qy = conv(c,r);
qu = conv(c,s);

% Construct a white noise input

rand('normal');
w = 0.1*rand(200,1);

% Output simulation

subplot(211);
plot(dlsim(qy,p,w));
title('Output');
ylabel('CO2');

% Control simulation

subplot(212);
plot(dlsim(qu,p,w));
title('Control Action');
ylabel('Gas Flow');
```

The simulated performance of the closed-loop regulator results in a 20.5% reduction in the variance of the CO_2 stack gas composition compared to the case of no control. The results are shown in Figure 7. Many additional aspects of the problem can be readily treated using simple Matlab procedures.

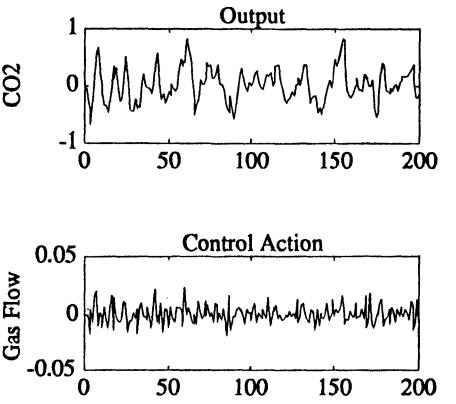


Figure 7: Response of the gas furnace with LQ controlin place.

6 Summary Remarks (Why Matlab Can't be Used for Everything?)

In spite of its many useful features, Matlab is not an appropriate tool for all applications. While it is difficult to draw precise boundaries, there are some general guidelines.

- *Matlab is useful when your problems are 'vectorizable'.*

  Matlab exhibits excellent floating point performance when using its matrix oriented primitive operations. However, because it is an interpreted (not compiled) language, it suffers some performance degradation on scalar and non-numeric operations. Some algorithms, such as for integrating ordinary differential equations, can be quite slow in Matlab for this reason.

- *Matlab is useful for prototyping algorithms.*

  Matlab is a high-level language with a large number of primitives so that even complex algorithms can be written in a minimal number of lines. The interpreter provides a convenient mechanism for debugging numerical algorithms. For example, simply by deleting the semicolon at the end of a line, the intermediate results of any computation are printed. There are also facilities for introducing keyboard interrupts and monitoring intermediate values.

- *Matlab is useful when you need results fast.*

  In addition to the points given above, the available toolboxes and graphics facilities are often sufficient for solving problems from start to finish, including the production of publication graphics.

- *Matlab does not replace either FORTRAN or specialized application software.*

  Matlab is not a replacement for a FORTRAN compiler and a good package of scientific subroutines. It not suited to truly large scale computation, nor can it be used effectively in a batch mode. Linear programming provides an example of the tradeoffs. Straightforward Matlab LP codes might be useful for problems with, say, up to a few hundred constraints. This is no match

for commercial that can handle many thousands of constraints.

- *Matlab is not very effective for non-numerical algorithms.*

  Matlab treats essentially all information as matrices of real or complex floating point numbers. The simple facilities for handling textual data in Matlab are inadequate for anything beyond manipulating titles and labels. It would be a mistake to use Matlab to do data base programming, for example, or for writing compilers.

7 Where to Obtain Matlab

Academic institutions can purchase Matlab directly from the MathWorks, Inc. Their address is

The MathWorks, Inc.
21 Eliot Street
South Natick, MA 01760
Phone: (508) 653-1415
Fax: (508) 653-2997
E-mail: tung@mathworks.com

The MathWorks has special licensing provisions for classroom and educational use. For commercial uses, Matlab is also distributed by

MGA, Inc.
73 Junction Square Dr.
Concord, MA 01742
Phone: (508) 369-5115

Versions of Matlab are available for IBM PC, AT, and 80386 platforms, including Weitek support. Also for the Apple Macintosh (with and without support for the 68881), Sun and Apollo workstations, DEC Vax, Gould, and Ardent machines. The Ardent version has facilities for 3D solids rendering.

References

[1] Astrom, Karl J., and Bjorn Wittenmark (1984). *Computer Controlled Systems.* Prentice-Hall, Englewood Cliffs, NJ.

[2] Box, George E. P., and Gwilym M. Jenkins (1976). *Time Series Analysis: Forecasting and Control.* Rev. Edition. Holden-Day, San Francisco.

[3] Chapra, Steven C., and Raymond P. Canale (1988). *Numerical Methods for Engineers,* Second Edition. McGraw-Hill, New York.

[4] Coleman, Thomas F., and Charles Van Loan (1988). *Handbook for Matrix Computations.* SIAM, Philadelphia.

[5] Fletcher, R. (1987). *Practical Methods of Optimization,* Second Edition. John Wiley & Sons, New York.

[6] Forsythe, George E., Michael A. Malcolm, and Cleve B. Moler (1977). *Computer Methods for Mathematical Computations.* Prentice-Hall, Englewood Cliffs, NJ.

[7] Hill, David R. (1988). *Experiments in Computational Matrix Algebra.* Random House, New York.

[8] Kahaner, David, Cleve Moler, and Stephen Nash (1989). *Numerical Methods and Software.* Prentice-Hall, Englewood Cliffs, NJ.

[9] Ljung, Lennart (1987). *System Identification: Theory for the User.* Prentice-Hall, Englewood Cliffs, NJ.

[10] Noble, Ben, and James W. Daniel (1988). *Applied Linear Algebra, Third Edition.* Prentice-Hall, Englewood Cliffs, NJ.

[11] Press, William H., Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling (1986). *Numerical Recipes - The Art of Scientific Computing.* Cambridge University Press, Cambridge.

[12] Wirth, Nicklaus (1976). *Algorithms + Data Structures = Programs.* Prentice-Hall, Englewood Cliffs, NJ.

Application Note

Modeling and Analysis of a 2-Degree-of-Freedom Robot Arm

This application note describes the modeling and analysis of a two-linkage robot arm using MATRIX$_x$® and SystemBuild™. The nonlinear equations of motion of the system are presented, followed by the SystemBuild block diagram description of those equations. The SystemBuild model is linearized and an optimal regulator is designed based on the linearized model. The response of the closed-loop system is found through simulation and the results are plotted.

Modeling

Consider the two-linkage robot arm shown in Figure 1. Both links are assumed to be perfectly rigid and are connected by a frictionless pin joint. The system thus has two degrees of freedom, θ_1 and θ_2. There are two control inputs to the system, the motor torques τ_1 and τ_2 at the rotating joints. For a particular set of arm masses, lengths, and inertias, the nonlinear equations of motion for the system are:

$$(1)\quad \ddot{\theta}_1 = \frac{1}{I}\left[\tau_1 - \tau_2 + 0.01\ \dot{\theta}_1\dot{\theta}_2 \sin 2\theta_2\right]$$

$$(2)\quad \ddot{\theta}_2 = \frac{\tau_2}{0.01} - \frac{1}{2}\dot{\theta}_1^2 \sin 2\theta_2$$

where I is given by:

$$(3)\quad I = 0.07 + 0.06 \cos^2\theta_2 + 0.05 \sin 2\theta_2$$

These nonlinear dynamic equations can be represented in SystemBuild, the interactive block diagram modeling facility of MATRIXx, using combinations of algebraic and dynamic blocks. Block diagrams constructed in SystemBuild are hierarchical. Each node in the hierarchy is represented by a SuperBlock, which can contain up to 99 other blocks, including other SuperBlocks.

Figure 2 illustrates how the dynamics of the two-linkage robot arm can be modeled in SystemBuild. The ROBOT super-block shown in Figure 2 contains two algebraic general expression blocks, four *N*th order integrator blocks, two trigonometric function blocks, and one gain block. Figure 3 gives the necessary details required to define each block.

General expression blocks are defined by passing text strings in the block form. The following text string was used in defining

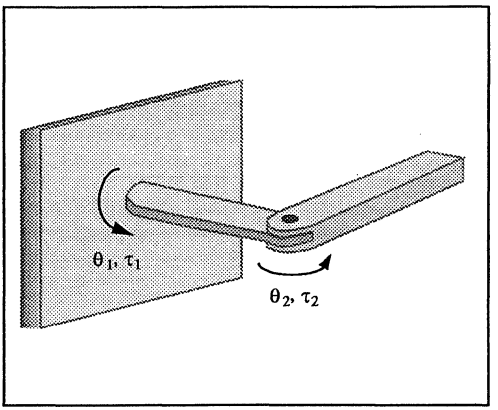


Figure 1: Two-Degree-of-Freedom (2-DOF) Robot Arm

the block with an ID of [32] and having inertia as an output (see Figures 2 and 3).

$$Y = 0.07 + 0.06 * U2 * U2 + 0.05 * U1 * U1$$

This block calculates inertia as defined in Equation (3). The strings used to calculate $\ddot{\theta}_1$ and $\ddot{\theta}_2$ in the algebraic expression block with an ID of [12] are:

$$Y1 = (U1 - U2 + 0.01 * U3 * U4 * U5)/U6$$
$$Y2 = U2/0.01 - 1/2 * U3 * U3 * U5;$$

Note: $Y1$ is the calculation of $\ddot{\theta}_1$, and $Y2$ is the calculation of $\ddot{\theta}_2$ as defined in Equations (1) and (2).

Once all of the blocks have been defined as illustrated in Figure 2, the system can be analyzed through the ANALYZE option of SystemBuild. When this option is selected under the BUILD menu, SystemBuild creates and internal simulation model by assembling all of the SuperBlocks in the hierarchy. A reference map is then created which displays the structure of the super-block hierarchy:

```
Super-Block Reference Map:
        ROBOT
All super-blocks identified
System Built with 0 error(s) and 0 warning(s).
Use SIM('IALG') to set the integration algorithm
```

The ANALYZE option in SystemBuild returns the user to the MATRIXx command level where he can simulate the system, linearize it, or issue any MATRIXx command.

integrated systems inc.

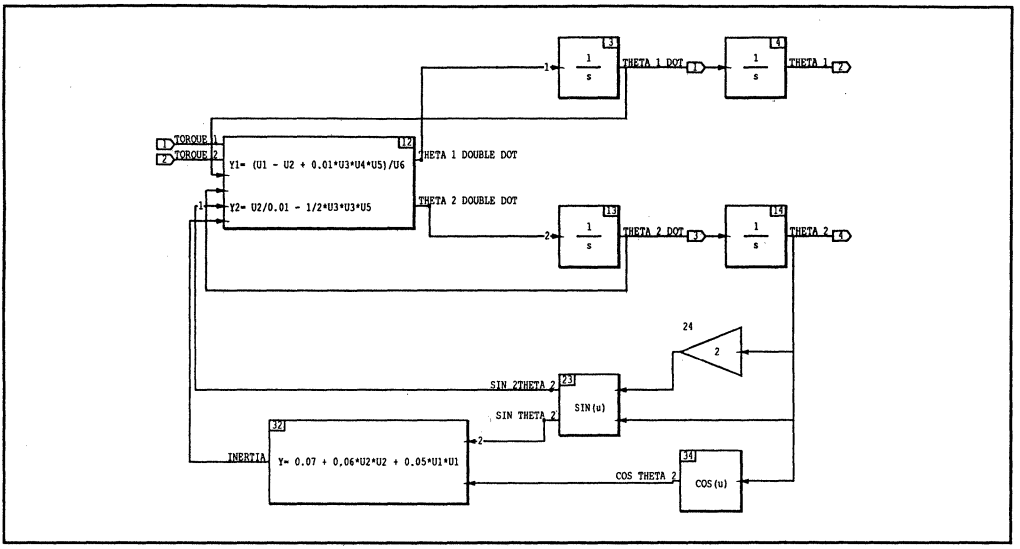


Figure 2: Robot Arm Dynamics

Linearization and Controller Design

In MATRIXx the continuous state space model is described by a system matrix, *S* and the number of states, *NS*. The *S* matrix is defined as the concatenation of the four matrices, (*A*, *B*, *C*, and *D*) used in describing a linear system as given by the following relation between the system output, *y* and the inputs, *u*.

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$
$$\text{where } x(0) = x$$

and

$$S = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

Once back at the MATRIXx command level (the <> prompt), the system built in SystemBuild can be linearized with the LIN command:

```
<> [SL,NSL]=LIN(.1)
```

where the argument of the LIN command is the size of the perturbation to be applied to all system states and inputs when the partials are computed numerically. MATRIXx returns the system state space matrix, *SL*, and the number of states, *NSL*, which represent the linearized system:

```
NSL       =
 4.

SL        =

  0.0000    0.0000    0.0000    0.0000    7.6923   -7.6923
  1.0000    0.0000    0.0000    0.0000    0.0000    0.0000
  0.0000    0.0000    0.0000    0.0000    0.0000  100.0000
  0.0000    0.0000    1.0000    0.0000    0.0000    0.0000
  1.0000    0.0000    0.0000    0.0000    0.0000    0.0000
  0.0000    1.0000    0.0000    0.0000    0.0000    0.0000
  0.0000    0.0000    1.0000    0.0000    0.0000    0.0000
  0.0000    0.0000    0.0000    1.0000    0.0000    0.0000
```

The system has four states ($\dot{\theta}_1$, θ_1, $\dot{\theta}_2$ and θ_2), two inputs (the motor torques) and four outputs (which are the states).

Once the ROBOT SuperBlock has been linearized, one can design a linear regulator controller. The REGULATOR command computes the optimal constant gain, state-feedback matrices for continuous-time systems under the assumption of full state feedback.

Inputs into the REGULATOR command include the *A* and *B* (plant and input) components of the system matrix, *S* and the design weighting matrices, R_{xx}, R_{uu}, and R_{xu}, where R_{xu} is optional. The design weighting matrices provide weights on the states, *x*, and controls, *u*, as defined by the following quadratic cost function:

$$COST = \int_0^{\infty} (x'R_{xx}x + u'R_{uu}u + x'R_{xu}u + u'R'_{xu}x)dt$$

Note: *Ruu* must be positive definite and *Rxx* must be positive semi-definite.

The *A* and *B* parts of the system matrix, *SL*, can be extracted with the SPLIT command:

```
<> [A,B]=SPLIT(SL,NSL)

B          =

  7.6923    -7.6923
  0.0000     0.0000
  0.0000   100.0000
  0.0000     0.0000
```

```
A          =

  0.    0.    0.    0.
  1.    0.    0.    0.
  0.    0.    0.    0.
  0.    0.    1.    0.
```

Diagonal state (*RXX*) and control (*RUU*) weighting matrices are defined for the purpose of designing an optimal regulator.

```
<> RXX=DIAG([10  100  1  100])

  RXX          =

 10     0.    0.    0.
  0.  100.    0.    0.
  0.    0.    1.    0.
  0.    0.    0.  100.
```

```
<> RUU=DIAG([20  100])

  RUU          =

 20.    0.
  0.  100.
```

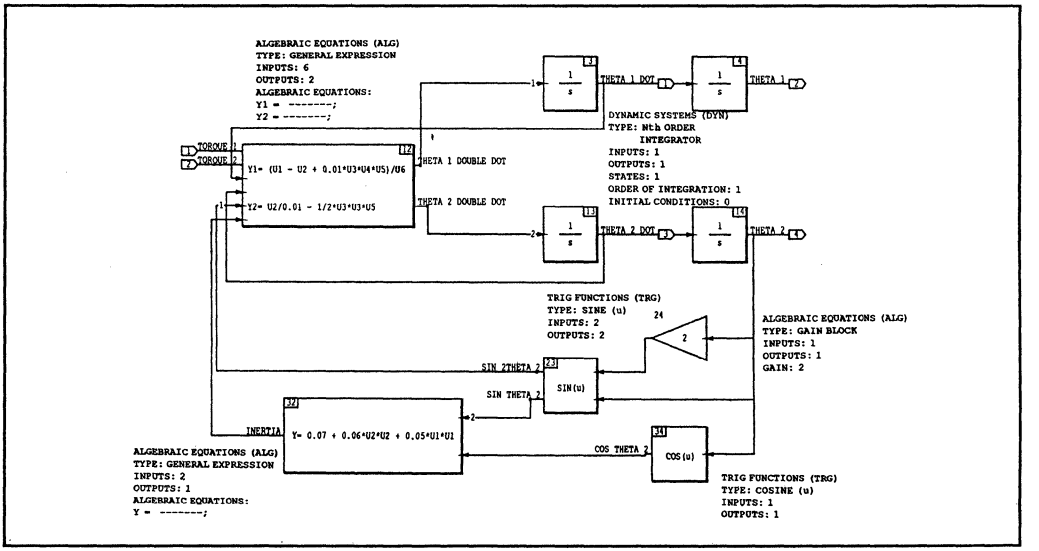


Figure 3: Block Form Details (ROB SuperBlock)

The optimal regulator is designed with the REGULATOR command:

```
<>[EV,KR]=REGULATOR(A,B,RXX,RUU)
```

MATRIXx returns the closed-loop eigenvalues (of the linearized system) and the optimal regulator state feedback gains, *KR*.

```
KR       =

   1.0365  2.2261  0.0683  0.2112
  -0.0298 -0.0944  0.1727  0.9955

EV       =

  -8.7233  +  4.8199i
  -8.7233  -  4.8199i
  -4.0127  +  1.1024i
  -4.0127  -  1.1024i
```

The closed-loop system can now be completed in SystemBuild as the SuperBlock SYSTEM, which is illustrated in Figure 4. This SuperBlock includes the SuperBlock ROBOT (the open-loop plant), the gain block, MINUS KR, and two summing junctions. Rectangular gain matrices are defined in SystemBuild as state space systems with zero states. Thus the gain block, MINUS KR is defined as a state space system with zero states, four inputs, and two outputs, and with the gain matrix *–KR* passed from the MATRIXx stack (note: input as [–KR]). The SuperBlock SYSTEM has six external inputs, the first four being the reference states being the last two are reference (disturbance) torques. SYSTEM also has four external outputs which are the actual states. The summing junction in the top left of Figure 4 computes the difference between the reference states and the actual states. This error vector goes to the gain block, which computes the control torques. These control torques are differenced from the reference torques in the summing junction which is just below the MINUS KR block in Figure 4. The outputs of this summing junction are the actual torques which are inputs to the differential equations in the ROBOT SuperBlock. Figure 5 gives the details necessary to fill out each of the block forms in the SYSTEM SuperBlock.

Closed-Loop Simulation

The closed-loop system can be analyzed through the ANALYZE option of SystemBuild. Selecting SYSTEM for analysis results in:

```
Super-Block Reference Map :
        SYSTEM
           ROBOT
All super-blocks identified
System Built with 0 error(s) and 0 warning(s).
Use SIM('IALG') to set the integration algorithm
```

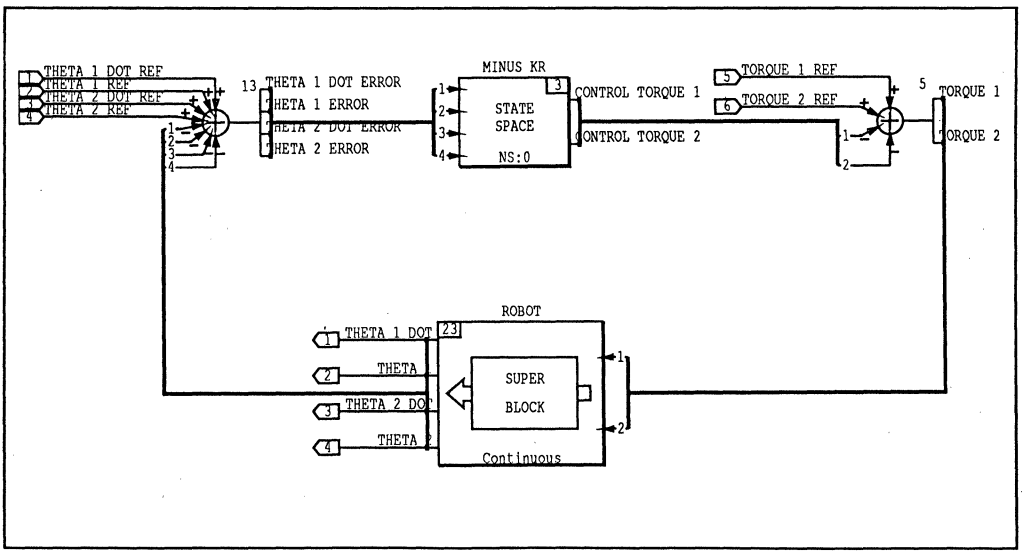


Figure 4: Closed-Loop System

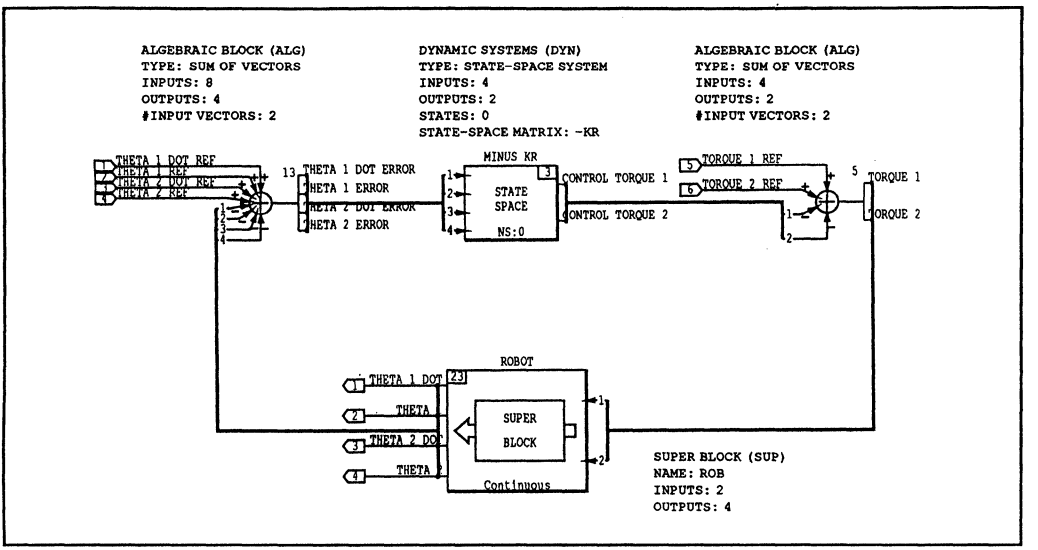


Figure 5: Block Form Details (SYSTEM SuperBlock)

After receiving the above message you will be at the MATRIXx command level. The time vector used for simulation is defined as starting at 0 and going to 10 seconds in steps of 0.1 seconds.

```
<> T=[0:0.1:10]';
```

The reference states call for step rotations of both joints at consistent angular velocities (0.5 and 0.375 radians/seconds) from 0 to 2 seconds, after which the final angles (0.1 and 0.075 radians) are to be held. The reference states are then defined as:

```
<> THE1DOT=[0.05*ONES(21,1);0*ONES(80,1)];
<> THE1=[0.05*T(1:21);0.10*ONES(80,1)];
<> THE2DOT=[0.0375*ONES(21,1);0*ONES(80,1)];
<> THE2=[0.0375*T(1:21);0.075*ONES(80,1)];

<> USTATE=[THE1DOT THE1 THE2DOT THE2];
```

The reference states can be plotted by typing the following command (see Figure 6):

```
<> PLOT(T,USTATE, 'STRIP REPORT XLAB/TIME (sec)/...
   YLAB/THETA1 DOT|THETA1|THETA2 DOT|THETA2|/...
   TITLE/REFERENCE INPUT VS TIME/')
```

The reference (disturbance) torques are defined as:

```
<> TAU1=0*ONES(T);
<> TAU2=0*ONES(T);

<> UTORQ=[TAU1 TAU2];
```

The reference states and torques can be combined to define the system input matrix:

```
<> USYS=[USTATE UTORQ];
```

The closed-loop response is then simulated with the SIM command as follows:

```
<> Y=SIM(T,USYS)
```

Figure 7 illustrates the system response to the system inputs. This plot can be generated by typing the following:

```
<> PLOT(T,USTATE, 'STRIP REPORT XLAB/TIME (sec)/...
   YLAB/THETA1 DOT|THETA1|THETA2 DOT|THETA2|/...
   TITL/SYSTEM RESPONSE VS TIME/')
```

We can compare the commanded and the actual trajectory by plotting both the input and the response on the same plot (see Figure 8).

```
<> PLOT(T,[USYS Y],'STRIP2 REPORT XLAB/TIME(sec)/...
   YLAB/THETA1 DOT|THETA1|THETA2 DOT|THETA2|/...
   TITL/REFERENCE INPUT & SYSTEM RESPONSE VS TIME/')
```

The response of the system over a larger (more nonlinear) state trajectory can be computed with:

```
<> Y2=SIM(T,2*USYS);
```

The results are shown in Figures 9 through 11. Note the responses are similar to those obtained with the smaller trajectory.

Alternate Methods

We have described an approach to modeling and analyzing a two-linkage robot arm using the ISI Product Family. Many different modeling approaches could also be taken. Using algebraic loops, the joint angular accelerations could be written in terms of themselves, i.e.:

$$\ddot{\theta}_1 = f(\ddot{\theta}_1, \ddot{\theta}_2, \dot{\theta}_1, \dot{\theta}_2, \theta_1, \theta_2, \tau_1, \tau_2)$$
$$\ddot{\theta}_2 = f(\ddot{\theta}_1, \ddot{\theta}_2, \dot{\theta}_1, \dot{\theta}_2, \theta_1, \theta_2, \tau_1, \tau_2)$$

This approach could be useful if the equations are hard to separate. FORTRAN blocks could also be used to define the dynamic equations. This would allow one to include existing FORTRAN simulation code into SystemBuild where the dynamics could be analyzed and controllers designed. One could use a symbolic manipulation program to generate the dynamic equations and the FORTRAN code to simulate them.

The controller design presented is a very simple continuous-time linear one. In practice, robot controllers tend to be nonlinear and multi-rate digital. Designing nonlinear multi-rate controllers is very easy with SystemBuild, as there are a wide variety of nonlinear blocks available. Sampling rates are defined at the SuperBlock level. Different sampling rates can be used for different SuperBlocks, without restricting the rates to being multiples of each other. Adaptive controllers could also be designed.

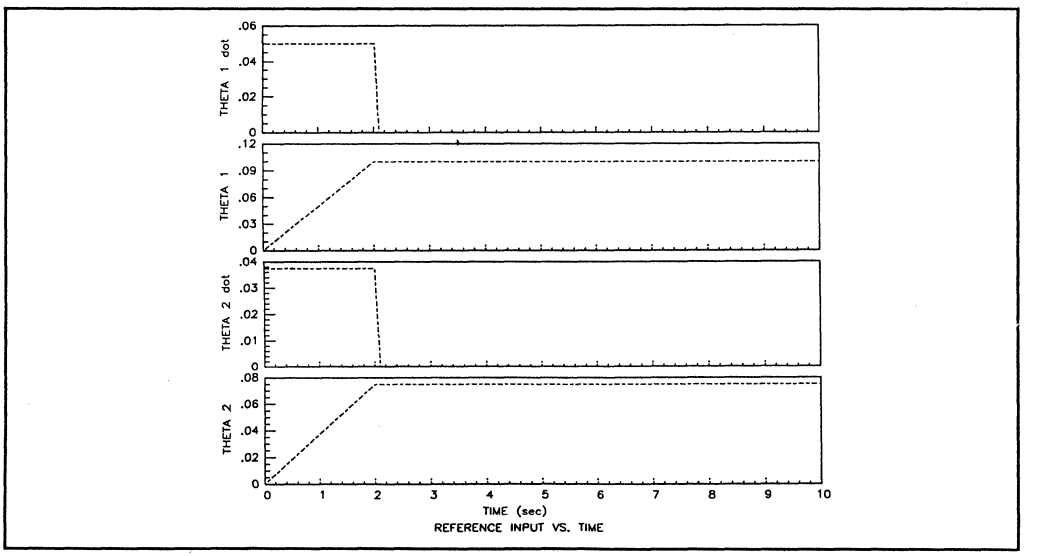


Figure 6

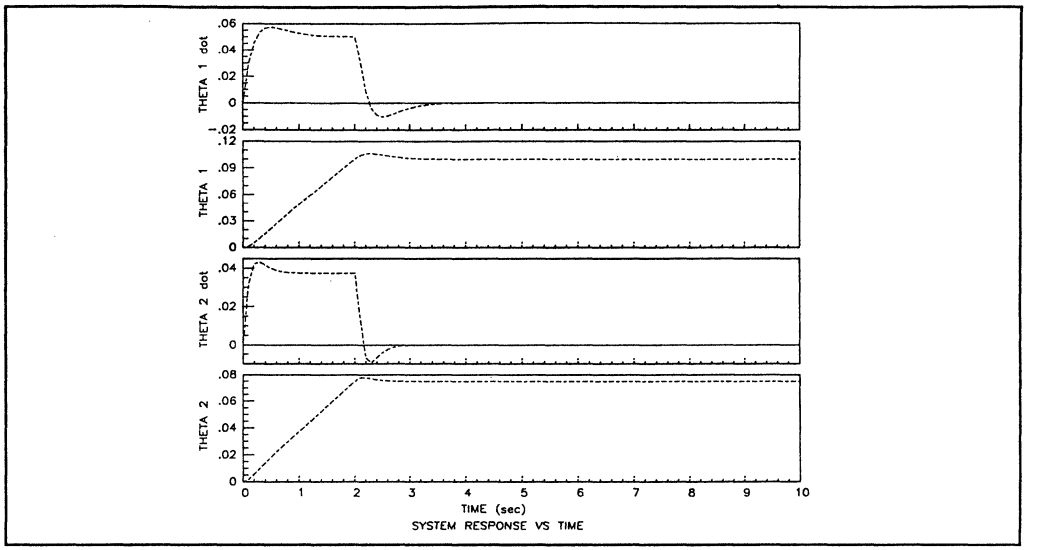


Figure 7

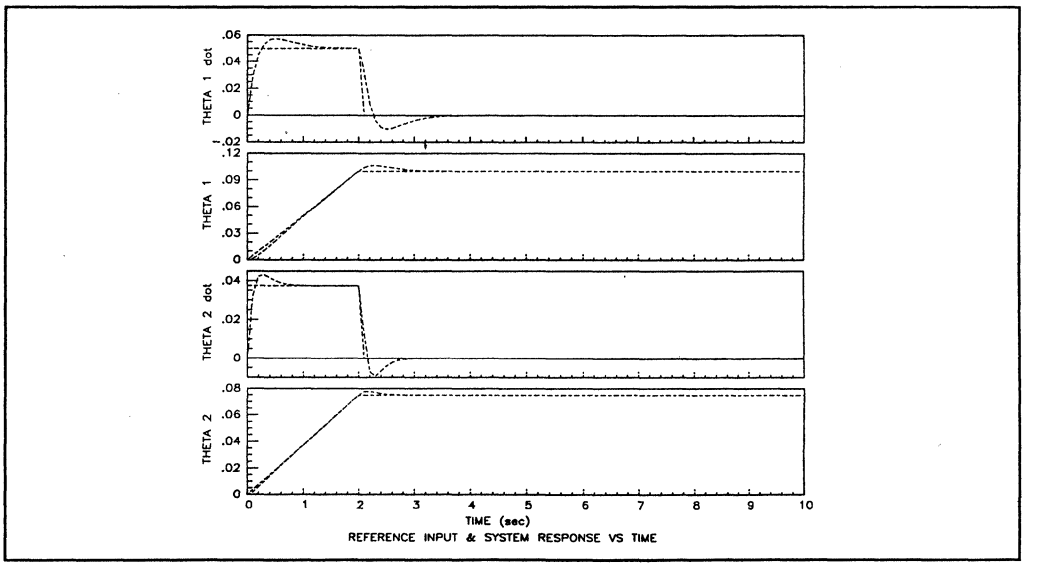


Figure 8

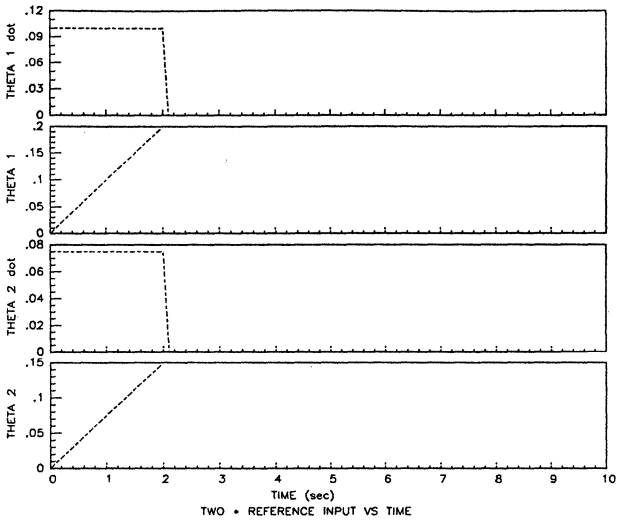


Figure 9

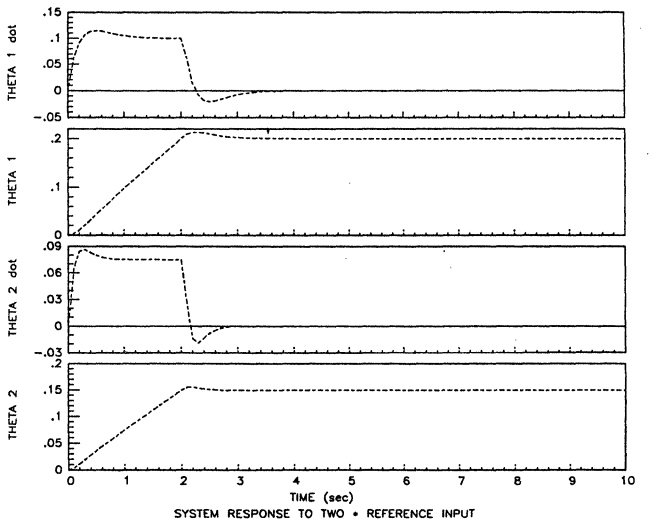


Figure 10

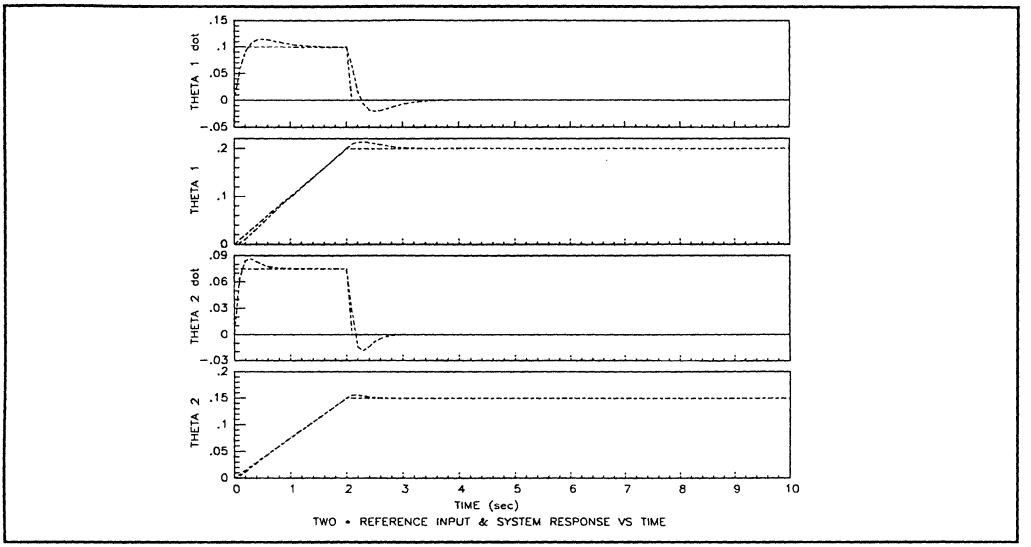


Figure 11

integrated systems inc.

Corporate Headquarters:
Integrated Systems Inc.
2500 Mission College Blvd.
Santa Clara, CA 95054-1233
Tel: (408) 980-1500

European Office:
Integrated Systems Inc Limited
274 Cambridge Science Park
Milton Road, Cambridge CB4 4WE
England
Tel: 0223 420999

AN0390 • 3/90

Simnon – A Simulation Language for Nonlinear Systems

Tomas Schönthal

Department of Automatic Control
Lund Institute of Technology
S-221 00 Lund, Sweden

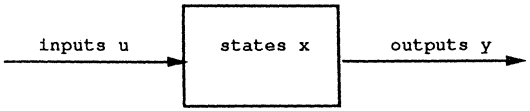
Abstract. This paper presents Simnon, an interactive simulation environment for nonlinear systems, developed by the Department of Automatic Control, Lund Institute of Technology, Lund Sweden. The following topics are covered: System descriptions, interactive facilities, examples, application areas and technical features.

1. Introduction

Simnon is a modular high level language for describing dynamical systems with continuous and/or discrete time. Equally important, it is an interactive command language, a "software laboratory", designed to organize and carry out simulation runs, vary circumstances (i.e. parameters, initial values or the models themselves) and display results graphically or numerically. A macro facilitity permits developers to pack models and command sequences into "turn-key" applications. The first version of Simnon appeared as the result of a master thesis in 1972. At that time digital simulation meant expensive batch runs on main frames, or writing your own dedicated Fortran programs, since there hardly existed any interactive systems with reasonably flexible input formats for the type of computers that a small research group could afford. Simnon soon became a standard tool at Automatic Control, Lund. In the years to follow Simnon went through several stages of development. Today Simnon is used worldwide by many universities for research and education in several disciplines and is equally popular in industry. Thanks to the MS-DOS version, Simnon is rapidly finding new users in both large and small organizations.

2. System Descriptions

The key concept is the system, which corresponds to a mathematical model of the reality being studied. In Simnon a system is a sequence of statements in a special modeling language. There are continuous systems (differential equations) and discrete systems (difference equations). A third type of system, connecting system, is used to form compound systems from continuous and discrete systems.



Continuous system:

$$\dot{x} = f(x, u, t)$$

$$y = g(x, u, t)$$

Discrete system:

$$x(t_{k+1}) = f(x(t_k), u(t_k), t_k)$$

$$y(t_k) = g(x(t_k), u(t_k), t_k)$$

$$t_{k+1} = h(x(t_k), u(t_k), t_k)$$

As we shall see later, describing a process (continuous system) controlled by a digital regulator (discrete system) is very natural in Simnon, but Simnon as such has no "built-in control theory". The approach is "open architecture", deferring all that is specific to a particular discipline to the user written models.
The statements of the system description language are: declarations (type of system, type of variable), assignments of variables, initial values and parameters. Variable assignment: variable = [IF condition THEN expression ELSE] expression. Expressions are formed by the common arithmetic operators and elementary functions. Random numbers, time delays, interpolation and the ability to drive a simulation by an external data file are also provided. Please refer to 'Technical Features, Compiler' for more details.

3. Interactive Facilities

Once a system has been written according to the rules of the system description language, the user can, with the aid of the command language, begin to experiment with it. First of all, the system has to be translated by Simnon's compiler. Then variables are selected for plotting, and a simulation over a selected time interval is started.
Simulation, in general, is very much a trial-and-error process. If the results differ from those expected, it is easy, with Simnon, to change a parameter, an initial value, or even an equation and repeat the simulation. In the meantime Simnon can accumulate raw material for a report. All this can be accomplished conveniently with only a few of Simnon's 43 commands. In addition to this, operating system commands may be executed from within Simnon.
The interaction mode is command driven, i.e. commands can be entered in arbitrary order, like when you communicate with conventional operating systems such as MS-DOS, Unix or VMS. In 'Technical Features, Macros' it is indicated how the user may influence this situation.

4. Examples

4.1 Chaos

In 1963 Lorenz derived a set of ordinary differential equations to approximate the behavior of atmospheric air currents:

$$\dot{x} = a(y - x)$$

$$\dot{y} = bx - y - xz$$

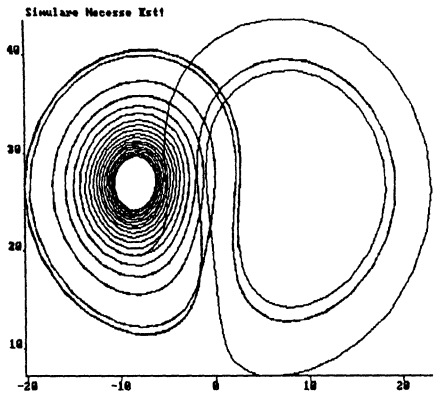
$$\dot{z} = xy - cz$$

These equations can be represented by the following Simnon system:

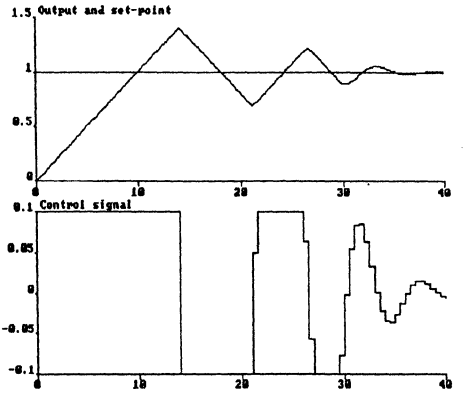
```
continuous system Lorenzeq
state x y z              States
der dx dy dz             Derivatives
dx=a*(y-x)               Computations
dy=b*x-y-x*z
dz=x*y-c*z
a: 10                    Parameters
b: 28
c: 2.667
x: -8                    Initial values
y: -8
z: 24
end
```

To solve the equations we type:

```
syst Lorenzeq            Translate the system
store x y z              Store the solution
error 1e-6               Demand higher accuracy
simu 0 20                Simulate
ashow z(y)               Plot z vs y
text 'Simulare Necesse Est!'
                         Add a title
hcopy                    Print the diagram
```

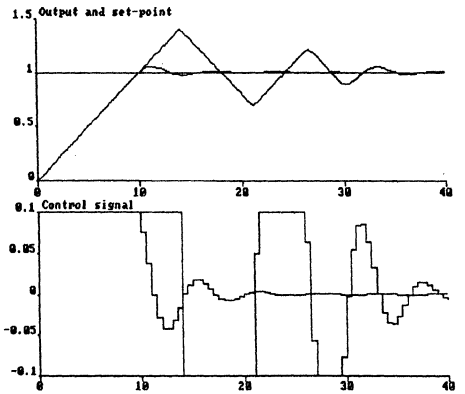


which produces:



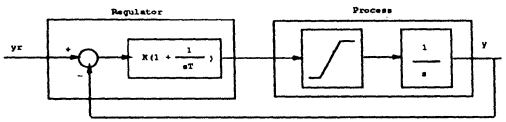
Now we can activate the anti-windup by setting the low and high values of the control. We then specify overplotting and repeat the simulation:

```
par ulow: -.1
par uhigh: .1
plot y[proc]:1 1 uclip:2 1
simu
```



4.2 Control

A simple example of a nonlinear control is one that respects the saturation limits of its regulator:



This model is represented in Simnon as a continuous process called proc, a discrete regulator called pireg and a connecting system called regsys. The discrete PI regulator has logic to limit saturation, or windup, on its integrator.
To simulate the model without anti-windup (default), type:

```
syst proc pireg regsys
store yr y[proc]
simu 0 40
split 2 1
ashow y yr
text 'Output and set-point'
ashow uclip
text 'Control signal'
```

This gives far better performance. If we instead wish to try an adaptive regulator in this environment, we could replace the module pireg with a "plug compatible" (i.e. having the same inputs and outputs) module adaptreg, and repeat the above commands, except, of course, that the parameter tunings would look different.

5. Application Areas

Simnon is used for education and research in

such diverse disciplines as automatic control, biology, chemical engineering, economics, electrical engineering, mathematics, mechanical engineering, etc. Typical problems include engine control, food processing, power systems, process control, robotics and ship steering.

6. Technical Features

6.1 Compiler

Before a model can be simulated, it has to be translated by Simnon's integrated compiler. The compiler not only checks for syntax errors, but also ensures that all quantities appearing to the right of an assignment have defined values. Thanks to the equation sorter, equations may be entered in arbitrary order, and algebraic loops will be detected. One kind of optimization is made: Time-invariant expressions are only evaluated once.
Numerical errors (e.g. zero divide) during simulation will be pinpointed in their source context. Since the models are compiled into machine code, the simulations will run as fast as Fortran programs. In contrast to conventional programming techniques, the turnaround times are neglible, allowing the users to modify their models "on the fly".
The MS-DOS version has dynamic memory allocation, which permits very large models.

6.2 Data Formats

Simnon is file oriented: System descriptions and macros are normal text files that can be prepared by any text editor. Time series are stored as binary files. These can be exported to printable ASCII (a time series then forms a column) and re-imported.
There exists a one-way path from PC-Matlab (The MathWorks, Inc, Sherborn, Mass.) to Simnon at the system description level: Included with Simnon is a preprocessor written as a Matlab function that takes as arguments a matrix set comprising a linear, time-invariant system and produces a complete Simnon system description. The command hcopy dumps the graphics part of the screen to a plotting device or to a file for further processing.

6.3 Documentation

Simnon comes with an English 180 page computer set tutorial and reference manual with many examples. The on-line help utility has over 100 entries.

6.4 Macros

Simnon usually takes commands from the keyboard, but a sequence of commands can be defined as a macro (for historical reasons the term 'macro' is used; perhaps a more adequate term is 'command procedure'). A macro can then be invoked by typing its name and any associated arguments. In this way the user may add extra commands to the Simnon vocabulary. There is provision for jumps and input/output just like in a programming language. Macros can be used to change Simnon's interaction mode from command driven to question and answer sequences, which may be utilised for demonstrations. Macros enable one person to develop and test a simulation model and someone else to use it. Macros have the feel of genuine Simnon commands, or they could act as "programs within the program". Typically, such a macro could present the user with a list of alternatives, then prompt the user for a choice (input wave form, PID-control or adaptive, etc.), or a numerical value.

6.5 System Requirements, MS-DOS version

- IBM PC, XT, AT, PS/2, 80386-based or compatible personal computer
- 8087, 80287 or 80387 maths coprocessor
- MS-DOS/PC-DOS version 2.00 (or later) or OS/2 with Compatibility Box
- 256 kB of RAM or more
- 3.5 or 5.25 inch diskette drive
- Hard disk (strongly recommended)
- One of these graphics systems (highest resolution used): CGA, EGA (enhanced or mono display), Ericsson PC, Hercules, Olivetti M24/AT&T, Toshiba PC or VGA/MCGA
- Recommended hard-copy devices:
  - Epson MX-80, IBM 5152 or compatible
  - HP LaserJet family
  - PostScript printers (e.g. Apple LaserWriter)

6.6 Prices, MS-DOS version

(July 1988, version 2.11) North American customers pay in US $. All other customers pay in Swedish currency (SEK). Swedish customers will be charged value added tax.

One copy of Simnon costs US $ 695 (SEK 5000). Quantity discounts:

3–4 copies	10 %
5–9 copies	15 %
10– copies	20 %

For universities and schools the following prices apply:

1 copy	US $ 345	(SEK 2500)
5 copies	US $ 1250	(SEK 9000)
10 copies	US $ 1750	(SEK 12500)
20 copies	US $ 2500	(SEK 18000)

Universities and schools may buy the *Classroom Kit* for US $ 500 (SEK 3500), provided that they (have) order(ed) at least one copy of regular Simnon. This reduced problem-size version of Simnon, which is intended for education only, comes with a license for 10 PCs.

7. References

For all of these works Simnon has been used extensively:

Åström, K. J., Bell, R. D. (1987): *Dynamic Models for Boiler-Turbine-Alternator Units: Data Logs and Parameter Estimation for a 160 MW Unit, CODEN LUTFD2/TFRT 3192*, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Simmon is a product of SSPA Systems,
PO Box 24001, S-400 22 Goteborg, Sweden.
Fax: +46 31 63 96 24, Phone: +46 31 63 95 00.

In North America, Simnon is provided exclusively by
Engineering Software Concepts, Inc.,
PO Box 66, Palo Alto, California 94301.
Fax: 415 325 0531, Phone: 800 325 1789
(in California 415 325 4321).

Olsson, G., Holmberg, U. and Wikström, A. (1985): *A Model Library for Dynamic Simulation of Activated Sludge Systems. Reprinted from Instrumentation and Control of Water and Wastewater Treatment and Transport Systems*, Pergamon Press, Oxford and New York.

Åström, K. J. and Wittenmark, B. (1984): *Computer Controlled Systems – Theory and Design*, Prentice Hall Inc, Englewood Cliffs, NJ.

Åström, K. J. and Wittenmark, B. (1988): *Adaptive Control*, Addison-Wesley, Reading, Mass.

Elmqvist, H., Åström, K. J. and Schönthal, T. (1986): *Simnon – User's guide for MS-DOS Computers*, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Mattsson, S. E. (1984): *Modelling and Control of Large Horizontal Axis Wind Power Plants, Ph.D. dissertation,*
CODEN: LUTFD2/TFRT-1026, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

PART III
Implementation of Digital Controllers

Implementing Digital Controllers

A lot of work has been done recently in the area of modern control theory, and many quite elegant theories have resulted. However, implementation has lagged substantially behind theory and idealized mathematical design. The outcome is that modern control theory is still limited somewhat to research labs, and most of the servo control applications in the industry utilize classical control techniques. This introduction discusses some of the issues in implementing digital controllers. It should be emphasized that there are no easy solutions — digital controllers still lag in the body of knowledge that is available for implementation. The introduction and the articles in this part may not provide canned solutions; however, they do highlight many pitfalls and problems of implementation and provide suggestions to minimize them.

The major issues in implementing digital controllers are the effects of finite word length, optimal controller structures, computational delays, and software development for microprocessors/DSPs. The most important issue in implementation is the effects of fixed-point arithmetic and finite word length. Some problems can be minimized by using floating-point processors; however, this may not always be possible. Before going into the effects of finite word length, section **Fixed-Point Versus Floating-Point** will review fixed-point and floating-point arithmetic formats.

Fixed-Point Versus Floating-Point

Floating-point processors have a very large dynamic range. In floating-point, a number is represented with a mantissa and an exponent. The mantissa represents the fraction, and the exponent represents the number of digits to the left of the decimal point. For example, assuming that a four-digit storage is available, 3740 can be written as 0.374×10^4. In floating-point, this can be represented as 4.374; where, exponent = 4 and mantissa = 374.

The largest floating-point number represented by four digits is 9.999 or $0.999 \times 10^9 = 999000000$. The largest fixed-point number represented by four digits is 9999.

Floating-point numbers thus allow a much larger dynamic range than fixed-point numbers. However, floating-point does not necessarily eliminate all finite word-length effects. Storage length is still limited, but with a larger dynamic range. There is also some loss of resolution. The number of significant digits in a mantissa determines the accuracy of the numerical value. However, the mantissa does not use all the storage capacity as some of the storage is taken up by the exponent. In practice, to minimize this loss of resolution, floating-point formats use 24 bits or greater to represent the mantissa. The TMS320 floating-point generations, TMS320C3x and TMS320C4x, have 32-bit architectures. Three floating-point formats are available: short format with a 12-bit mantissa and a 4-bit exponent, standard-precision format with a 24-bit mantissa and an 8-bit exponent, and extended-precision format with a 32-bit mantissa and an 8-bit exponent.

Floating-point processors are generally more expensive than fixed-point processors, and the cost may not be justified in some applications. Floating-point may be needed in applications where either gain coefficients are time varying or signals and gain coefficients have a large dynamic range. Other cases where floating-point can be justified is where development cost is more significant than component cost, and very low quantities are required. Floating-point processors usually allow code to be developed in high-level languages and reduce the need to fully identify the system's dynamic range.

Fixed-point processors generally are less expensive because less hardware is required on chip. In addition, they have smaller word length (typically 16 bits), and system cost is lower. However, more effort is required

to develop appropriate scaling factors to eliminate the effects of truncation or overflow during the intermediate and final states. Even in applications requiring use of floating-point for dynamic range requirements, it may be possible to use to use fixed-point processors. If gain coefficients have a large dynamic range but are constant, their dynamic range can usually be reduced by structure optimization techniques. If gain coefficients are time-varying and require adaptive control, a hybrid scheme can be used. Calculations for system identification typically have a slower update rate and can be performed with pseudo-floating-point format. The controller calculations, on the other hand, have a much faster rate and can be implemented in fixed-point arithmetic. Fixed-point processors can thus be used in most applications. The next section, **Binary Arithmetic,** will deal with fixed-point numbers only.

Binary Arithmetic

In binary format, a number can be represented in signed magnitude, where the left-most bit represents the sign and the remaining bits represent the magnitude:

+52 (decimal) = 34 (hex) is represented as 0011 0100 (binary)

–52 (decimal) = –34 (hex) is represented as 1011 0100 (binary)

Twos complement is an alternate form of representation used in most processors, including the TMS320. The representation of a positive number is the same in twos complement and in signed magnitude:

+52 (decimal) = 34 (hex) is represented as 0011 0100 (binary)

However, the representation of a negative number is different; as its name implies, the magnitude of a negative number is given in twos complement.

–52 (decimal) = –34 (hex) is represented by taking its twos complement, 1100 1100 (binary); i.e.,

Convert $+52_{10}$	0011 0100
Invert all bits to get ones complement	1100 1011
Add one to get twos complement	+ 1
Twos complement is	1100 1100

Therefore, –52 (decimal) = –34 (hex) is represented as 1100 1100

Adding 52 and (–52) gives

```
  0011 0100
+ 1100 1100
  0000 0000
```

as expected. The main advantage of twos complement is that only one adder is required to handle both positive and negative numbers. An addition will always give the correct result for both addition and subtraction. Also, if the final result is known to be within the processor's number range, an intermediate overflow can be ignored as the correct final result will still be produced. The largest positive number that can be represented with 8 bits is 7F (hex) or 127 (decimal), and the largest negative number represented with 8 bits is 80 (hex) or –128 (decimal).

The fixed-point binary representation does not have any binary point and does not represent fractions. However, it is sometimes advantageous to use an implied binary point to represent fractions. In signal processing, it is common to represent a number in fractions. For example, if 0.99 is the highest number that can be represented, the result of multiplying any two numbers will always be less than one — an overflow will never occur.

The location of the implied binary point affects neither the arithmetic unit nor the multiplier. It affects only the accuracy of the result and location from which that value will be read. For fractional arithmetic, the result is read from the upper 16 bits. For integer arithmetic, the result is read from the lower 16 bits (assuming no overflow). Fractional arithmetic loses accuracy but protects from overflows, while integer arithmetic

provides an exact result but offers no protection from overflow. In fractional arithmetic, an addition or a subtraction could produce an overflow, but a multiplication never causes one; generally, a single carry bit is sufficient to handle the overflow.

For TMS320 processors, numbers are typically represented in the Q15 format; where, the number following the letter Q represents the quantity of fractional bits. This implies that, in Q15, each number is represented by 1 sign bit, 15 fractional bits, and no integer bits. Likewise, a number in the Q13 format has 1 sign bit, 13 fractional bits, and 2 integer bits. The following shows both Q formats of eight decimal fractions and one integer:

decimal	**Q15**	**Q13**
+0.5	0.100 0000 0000 0000	000.1 0000 0000 0000
+0.25	0.010 0000 0000 0000	000.0 1000 0000 0000
+0.125	0.001 0000 0000 0000	000.0 0100 0000 0000
+0.875	0.111 0000 0000 0000	000.1 1100 0000 0000
–0.5	1.100 0000 0000 0000	100.1 0000 0000 0000
–0.25	1.110 0000 0000 0000	100.1 1000 0000 0000
–0.125	1.111 0000 0000 0000	100.1 1100 0000 0000
–0.875	1.001 0000 0000 0000	100.0 0010 0000 0000
–1.000	1.000 0000 0000 0000	100.0 0000 0000 0000

When two Q15 numbers are multiplied, the result is Q30 format and is also a fraction. The result has 30 fractional bits, 2 sign bits, and no integer bit.

```
  –0.5          1.100 0000 0000 0000
x +0.5        x 0.100 0000 0000 0000
  –0.25         11.11 0000 0000 0000 0000 0000 0000 0000
```

To store the result as a Q15 number, a left shift of one is performed to eliminate the extra sign bit, and the left-most significant 16 bits are stored. The result is stored as 1.110 0000 0000 0000.

Multiplication never gives an overflow in Q15 format, but successive additions may. If the final result is known to be within range, overflow in partial results will give correct results for the final sum. However, the saturation mode on the TMS320 must be turned off. For example,

```
  +0.875        1.100 0000 0000 0000    (Q15 format)
+ +0.50       + 0.100 0000 0000 0000
  +1.375        1.011 0000 0000 0000
+ –0.500      + 1.100 0000 0000 0000    (add twos complement to obtain result)
  +0.875        0.111 0000 0000 0000
```

Finite Word-Length Effects

Finite word-length effects are probably the most critical issue in implementing controllers. Most digital controllers use fixed-point processors. In a fixed-point processor, only a finite amount of storage length — for example, 4, 8, or 16 bits — is available to represent the signal and coefficients. Signals and coefficients must be scaled to fit in the dynamic range and word length of the processor. This limited storage capacity is referred to as the finite word-length issue. Finite word-length effects show up as noise in the system and may cause limit cycles or instability. But, it should be noted that finite word-length effects are somewhat forgiving in first- and second-order controllers. Finite word-length affects the controller in two ways: coefficient quantization and signal quantization.

Coefficient Quantization: Finite word length affects the representation of coefficients. The coefficients may need to be truncated or rounded to fit in that word length. If truncation or round-off is necessary, the process is called coefficient quantization. Coefficient quantization alters the transfer function of the system and changes the pole-zero locations and the gain of the system. Coefficient quantization is dependent upon the sampling rate as well as the word length. As the sampling rate gets higher, the poles tend to move toward and cluster around z=1, making the system very susceptible to coefficient quantization. Coefficient quantization can be minimized by using proper structures. Some of these structures make the system less susceptible to errors resulting from the effects of truncation/round-off. This is discussed in section **Controller Structures.**

Signal Quantization: Finite word length can also cause signal quantization. This can be divided into three different categories.

A/D and D/A Quantization Effects: One type of signal quantization occurs upon the conversion and representation of a continuous signal into discrete magnitude by an A/D or a D/A converter. The A/D and D/A word lengths are usually limited to 8–12 bits. A/D and D/A conversion also affects the controller by contributing to computational delay. This is discussed in section **Computational Delay .**

Most commercial A/D and D/A converters are available in the range of 8 to 16 bits with heavy premium on higher resolutions. An 8-bit A/D converter gives an accuracy of 1 in 256 or error of 0.4%, while a 10-bit A/D converter gives a resolution of a 1 in 1024 or an error of 0.1%. Unlike errors caused by the other quantization processes, errors in the processor's word length due to A/D and D/A effects are not recursively fed back into the control system. In most cases, signal conversion requires a smaller word length than the processor word length. Sensor accuracy must also be taken into account. If the sensor has a 5-mV noise in a 5-V system, then there is no point in having an A/D with greater than 10-bit resolution. Once the A/D is selected, the D/A is chosen to have the same or slightly higher resolution. Selection of A/Ds and D/As are usually not a major problem when implementing the controller. Too often, errors from numerical calculations (truncation or round-off) are mistaken as low resolution in the input/output signal.

If the controller is used in the servo mode and forced to follow a reference signal, the reference signal must then be represented correctly. If it is represented with a higher precision than the A/D's resolution, the error will never go to zero, causing a limit cycle.

Truncation and Round-Off Effects: The second kind of signal quantization appears when results of signal processing are truncated or rounded. As intermediate calculations are carried out, they need higher precision. For example, a 16 x 16 multiply requires a 32-bit register to store the result. If only 16 bits are available, the lower 16 bits are thrown away; this is known as truncation error. If the LSB is rounded before throwing away the lower 16 bits, this is known as round-off error. Since both of these errors are fed back recursively, they will accumulate as successive calculations are performed.

Truncation and round-off introduce bias and noise into the system, which may produce limit cycles because of nonlinearities. If q denotes the quantization step, μ denotes the mean of noise density, and ó denotes the variance of noise density, then

$\mu = q/2$ and $ó = q^2/12$ for truncating

$\mu = 0$ and $ó = q^2/12$ for rounding

These effects can be minimized by the proper selection of structures. For example, a fourth-order system becomes less sensitive to truncation and round-off errors if it is broken into lower-order parallel structures.

Overflow Effects: A third effect of signal quantization is overflow conditions. Successive calculations (i.e., addition) can cause registers to overflow even when fractional arithmetic is used. This, in return, will force the contents of associated registers to wrap around and change magnitude from most positive to most

negative numbers. This is equivalent to changing the direction of the control. To prevent this, a check for overflows must be continuously made during the intermediate and final stages. When twos complement arithmetic is used, intermediate overflows can sometimes be ignored if the final result is known to be within bounds. In the TMS320 architecture, a saturation mode is provided to prevent the contents of registers from wrapping around and changing sign when an overflow occurs. Overflow effects can be minimized by the proper selection of scaling factors and by leaving extra guard bits.

Scaling

Selection of a proper scaling factor is critical in minimizing the effects of finite word length. The scale factor should support the full dynamic range of signals and coefficients. A large scale factor may cause an overflow condition. Although overflow protection is built into the TMS320 architecture, it is advisable to minimize the possibility of overflows. To solve that problem, sometimes it may be necessary to choose a smaller (12–13 bits) scale factor. The small scale factor could, on the other hand, increase quantization noise.

Usually, there is little choice in handling the dynamic range of signals. If the dynamic range is too big, it may dictate selection of a floating-point instead of a fixed-point processor. Simulations are required to determine the dynamic range. In some cases, it may be possible to switch modes and change scale factors.

For proper scaling, a two-step approach is required. The first step requires optimization of the structure. Once the structure has been transformed into a suitable one for implementation, scaling can be carried out. If transfer functions are used, direct structures should be avoided and broken into smaller cascaded structures. If necessary, different scale factors can be chosen for each substructure. The scale factor is found by first calculating the worst-case response, H(z), of a system under maximum input signal conditions. Different techniques, l_p, l_1, and l_2 (described later in this section) may be used to find H(z). Next, H(z) must be scaled down in value to prevent an overflow during the intermediate and final stages. If fractional representation of a Q15 format is assumed, the scaled response, H'(z), must be less than unity. The scale factor, S_n, is finally found by satisfying the following relationship:

$$H'(z) = \frac{H(z)}{S_n}$$

where

$$\frac{H(z)}{S_n} < 1$$

For state space structures, diagonal scaling can be used. Again, before scaling, the first step requires the transformation process. Techniques like Schur transformation or Modal transformation can be used to optimize structures. These transformation techniques not only reduce the dynamic range of coefficients, but also reduce the number of nonzero elements in the structure. This minimizes the calculations that the processor must carry out.

The next step is to find the appropriate scale factors. The scaling factor must take into account the translation of proper input/output variables (i.e., voltage range of the A/D and D/A converters). In addition, it must prevent overflow or saturation during the intermediate states. Extensive simulations are usually necessary to ascertain the maximum and minimum values of states to provide the necessary scale factors. The scaling procedure can be broken into two different operations: input/output scaling and state vector scaling.

Input/output scaling transforms the internal fractional representation of numbers to external physical variables. Internal numbers within the range of +0.9999 to −1.0000 may have to be changed into external values of ±volts for the A/D and D/A converter. For example, given a system

$$\mathbf{x}_{n+1} = \mathbf{A}x_n + \mathbf{B}u_n$$
$$y_n = \mathbf{C}x_n + \mathbf{D}u_n$$

Then **B**, **C**, and **D** matrices must be scaled by the following relationship

$\mathbf{B}_s = \mathbf{B}[(S_u)^{-1}]$

$\mathbf{C}_s = [(S_y)^{-1}]\mathbf{C}$

$\mathbf{D}_s = [(S_y)^{-1}]\mathbf{D}[(S_u)^{-1}]$

where $(S_y)^{-1}$ and $(S_u)^{-1}$ are diagonal matrices.

For a system with an input/output physical range of ±10 V and a processor number range of ±1.0000, we have

$\mathbf{B}_s = 10\mathbf{B}$

$\mathbf{C}_s = 0.1\mathbf{C}$

$\mathbf{D}_s = \mathbf{D}$

For state vector scaling each, state variable must be scaled to keep it within the number range of the processor. Each state vector is divided by the following diagonal scale factor matrix.

$\mathbf{x}_s = [(S_x)^{-1}]\mathbf{x}$

The system can now be represented as

$\mathbf{x}_{s,n+1} = \mathbf{A}_s\mathbf{x}_{s,n} + \mathbf{B}_s\mathbf{u}_n$

$\mathbf{y}_n = \mathbf{C}_s\mathbf{x}_{s,n} + \mathbf{D}\mathbf{u}_n$

where

$\mathbf{A}_s = [(S_x)^{-1}]\mathbf{A}S_x$

$\mathbf{B}_s = [(S_x)^{-1}]\mathbf{B}$

$\mathbf{C}_s = \mathbf{C}S_x$

There are three different ways to calculate the scale factor matrices.

The first way to choose S_x is to simulate the closed loop under worst-case conditions and to check for overflow at each node or summation. The worst case is defined as when the largest absolute value of a state variable is selected for the calculation. This is know as l_p scaling.

Given

$S_{x,i} = \max[\text{abs}(x_{n,i})]$

then

$$S_x = \text{diag}\left(\frac{1}{S_{x,i}}\right)$$

The second approach is to statistically analyze for the probability of overflow at each node instead of doing actual simulations. This is known as l_2 scaling.

The third approach is to perform an analysis with certain bounded conditions of input signals. This is known as l_1 scaling. l_1 scaling can be applied only to stable systems.

Controller Structures

Selection of the proper control structure for digital controllers is a very critical issue, and its importance cannot be overemphasized. It is often the most overlooked aspect of implementation. Digital controllers can be described in terms of different, but equivalent structures. These structures have the same infinite

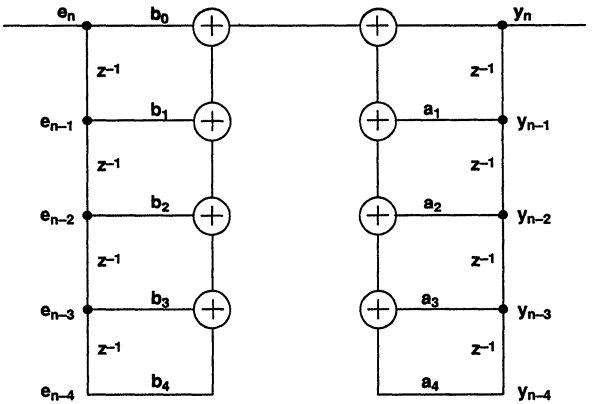
word-length behavior but different finite word-length behavior. The difference in finite word-length behavior results from the fact that some structures have coefficients that are less sensitive to coefficient truncation or that lie within a smaller numerical range, thus making it easier to scale. They may also produce lower-order equations.

Transfer Function Forms: Several different structures can represent systems when transfer functions are used. The simplest form is the direct structure shown in Figure 1.

Usually, in this structure, the coefficients have a wide range, depending upon the pole-zero locations. This makes the structure very susceptible to coefficient quantization, round-off error, and overflow. The structure can be represented in a transfer function form as

$$H(z) = \frac{b_0 + b_1z^{-1} + b_2z^{-2} + b_3z^{-3} + b_4z^{-4}}{1 + a_1z^{-1} + a_2z^{-2} + a_3z^{-3} + a_4z^{-4}}$$

Figure 1. Direct Structure



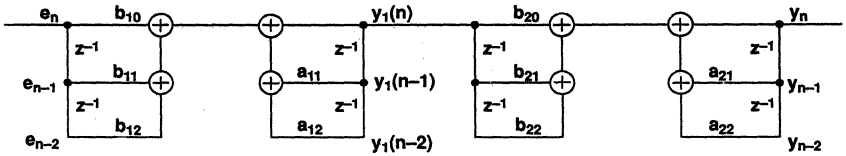
Another alternate structure is the cascaded structure as shown in Figure 2.

This can be represented in a transfer function given by

$$H(z) = \frac{(b_{10} + b_{11}z^{-1} + b_{12}z^{-2})(b_{20} + b_{21}z^{-1} + b_{22}z^{-2})}{(1 + a_{11}z^{-1} + a_{12}z^{-2})(1 + a_{21}z^{-1} + a_{22}z^{-2})}$$

The cascaded structure is somewhat less susceptible to round-off error and overflow than the direct structure. One advantage of this method is that poles and zeroes close to each other can be matched together. This will reduce the range of coefficients for each substructure. Different scale factors can then be chosen for them. A transfer function should be broken into first- or second-order cascaded functions to derive the greatest advantage from this structure.

Figure 2. Cascaded Structure



A parallel structure can also be chosen to represent a system. This is shown in Figure 3.

It is least susceptible to round-off errors and overflow problems. A parallel structure can be obtained by partial fraction expansion or division. This can be represented as

$$H(z) = \frac{(b_{10} + b_{11}z^{-1})}{(1 + a_{11}z^{-1})} + \frac{(b_{20} + b_{21}z^{-1})}{(1 + a_{21}z^{-1})} + \frac{(b_{30} + b_{31}z^{-1})}{(1 + a_{31}z^{-1})} + \frac{(b_{40} + b_{41}z^{-1})}{(1 + a_{41}z^{-1})}$$

For example, if the transfer function is given by

$$H(z) = \frac{z_2 - az_1 + bz_1}{z_2 - 1.9979 + 0.9979}$$

the poles of the systems are at $z_1 = 0.9988$ and $z_2 = 0.9991$.

If this system is represented with coefficient round-off, it becomes

$$H(z) = \frac{z_2 - az_1 + bz_1}{z_2 - 1.998 + 0.998}$$

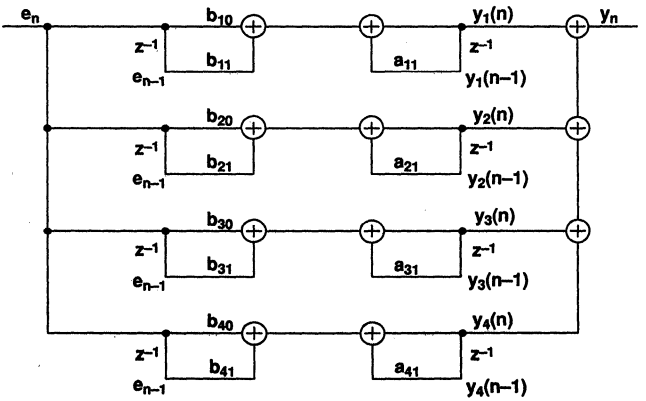
The new pole locations are now $z_1 = 0.9980$ and $z_2 = 1.0000$.

If this system had been represented as a cascade of two first-order substructures, the new structure after round-off would be

$$H(z) = \frac{(z - a_1)(z - a_2)}{(z - 0.998)(z - 0.999)}$$

Thus, the cascaded structure shows less sensitivity to coefficient round-off.

Figure 3. Parallel Structure



State Space Form: If the state space form is used, the controller can again be represented in different, but equivalent state space structures that can give better finite word-length behavior. Structure transformation techniques should also be employed to create structures that will have less numerical sensitivity. Structural forms like Modal or Schur can reduce the number of nonzero elements in the structure.

The Modal form of a matrix is a diagonal matrix with all its eigenvalues as the diagonal elements. If the eigenvalues are complex, then the diagonal elements are a 2-by-2 matrix. The Modal form requires that all eigenvalues be linearly independent. This is referred to as the diagonal canonical form. The Modal form is represented as follows:

$$\begin{bmatrix} r_1 & 0 & 0 & 0 & 0 \\ 0 & r_2 & 0 & 0 & 0 \\ 0 & 0 & r_3 & 0 & 0 \\ 0 & 0 & 0 & r_4 & 0 \\ 0 & 0 & 0 & 0 & r_5 \end{bmatrix}$$

The Schur representation of a matrix is the upper-right triangular portion of the matrix with its eigenvalues on the diagonal. If the eigenvalues are complex, then they are 2-by-2 blocks on the diagonal. The Schur representation is given as follows:

$$\begin{bmatrix} r_1 & x & x & x & x \\ 0 & r_2 & x & x & x \\ 0 & 0 & r_3 & x & x \\ 0 & 0 & 0 & r_4 & x \\ 0 & 0 & 0 & 0 & r_5 \end{bmatrix}$$

The following example shows the effects of structure transformation; complete implementation examples along with the TMS320C14 code are given in Appendix 1. The state controller and estimator that were developed in PART II's introduction are used here. The structure is transformed with the Schur method and Impex® software. The A matrix now represents

A – BK – LC

from the original matrices in order to satisfy the input requirements for the Impex® software. The original system is given by the following set of coefficients. The software uses extended-precision/floating-point format to represent the original that system. After structure optimization and scaling, the numbers are converted into 16-bit/fixed-point format for implementation and code generation. For illustrational purposes, the system will also be represented in 32-bit/fixed-point format to show the loss of resolution due to lack of structure optimization.

		Representation in 64-Bit Floating-Point	Representation in 32-Bit Fixed-Point
Matrix A	a(1,1)	–6.66408081842000E–02	–00.066408
	a(2,1)	–2.83492351899385E+02	–28.349235
	a(1,2)	9.26115172000000E–04	00.000926
	a(2,2)	8.52505649404000E–01	00.852505
Matrix B	b(1)	2.68531510600000E–05	00.000026
	b(2)	5.36066065964500E–02	00.053606
Matrix C	c(1)	1.00000000000000E+00	------

After the Schur transformation, the matrices are obtained as follows:

		Representation in 64-Bit Floating-Point	Representation in 32-Bit Fixed-Point
Matrix A	a(1,1)	–6.66408081842000E–02	–00.06640808
	a(2,1)	–5.53695999803486E–01	–00.55369559
	a(1,2)	4.74170968064000E–01	00.47417096
	a(2,2)	8.52505649404000E–01	00.85250564
Matrix B	b(1)	1.37488133427200E–02	00.01374881
	b(2)	5.36066065964500E–02	00.05360660
Matrix C	c(1)	1.95312500000000E–03	-------

Note that the Schur transformation has tremendously reduced the dynamic range of the coefficients, thus, making it easier to scale them. Matrix C is not treated since it can be scaled independently.

Computational Delay

Computational delay is a critical disadvantage to using digital controllers. It has prevented widespread use of microprocessors and microcomputers in digital controllers because the amount of computational delay that is produced by these processing elements is unacceptable. With the high-performance of DSPs computational delay becomes more manageable. Computational delay shows up as phase delay within the system and affects the phase margins of that system. The negative phase-shift contribution can be calculated as follows:

phase delay = (computational delay)(bandwidth frequency)(360 °)

For a system with a 1-kHz bandwidth, a 100-μs computational delay will produce a negative phase shift of 36 degrees.

Even when using DSPs, it is advisable to minimize the effect of computational delay. This may be done by adopting appropriate structures or signal flows. For example, a compensator is represented by the following difference equation:

$$u(n) = \mathbf{K}_1[u(n-2)] + \mathbf{K}_2[u(n-1)] + \mathbf{K}_3[y(n-2)] + \mathbf{K}_4[y(n-1)] + \mathbf{K}_5[y(n)]$$

Only the last element, $\mathbf{K}_5[y(n)]$, is dependent upon the latest measurement. The remaining elements can be precomputed and stored into memory. As soon as the measurement is made, the last element can be calculated and the control output u(n) sent to the actuator.

Similarly, a state estimator is expressed as

$$\hat{\mathbf{x}}(n+1) = \mathbf{A}[\hat{\mathbf{x}}(n)] + \mathbf{B}[\mathbf{u}(n)] + \mathbf{L}[y - \mathbf{C}\hat{\mathbf{x}}(n)]$$

$$y = \mathbf{C}[\hat{\mathbf{x}}(n)]$$

$$\mathbf{u} = -\mathbf{K}[\hat{\mathbf{x}}(n)]$$

These can be split up as follows:

$$\bar{\mathbf{x}}(n+1) = \mathbf{A}\hat{\mathbf{x}}(n) + \mathbf{B}[\mathbf{u}(n)]$$

$$\bar{y} = \mathbf{C}[\bar{\mathbf{x}}(n+1)]$$

As soon as the measurement y is made, the control can be calculated by the following:

$$\hat{\mathbf{x}}(n+1) = \bar{\mathbf{x}}(n+1) + \mathbf{L}(y - \bar{y})$$

$$\mathbf{u} = -\mathbf{K}[\hat{\mathbf{x}}(n+1)]$$

This structure is usually referred to as a current estimator.

Another aspect of computational delay is the contribution by the A/D and D/A converters; the A/D usually being the main factor. The A/D has some minimum conversion time, while the D/A requires settling time. The conversion delay of the A/D creates a negative phase shift and affects the phase margin and stability of the system. The ZOH hold action of the D/A converter produces a delay of one sample time. This delay is comprehended into the design when the plant is discretized. The A/D conversion and the D/A settling times, on the other hand, must be taken into account during the implementation.

Typical A/D converters available in the market today range in conversion time from 50 ns for video applications to 50 μs for data acquisition. There is often a trade-off between conversion time and resolution. Those A/Ds with fast conversion times usually have lower resolution. For most control systems, A/D converters are chosen with conversion time of 15 μs or less. However, the selection depends upon the bandwidth and phase margin of that system. The phase delay is given by

phase delay = (computational delay)(bandwidth frequency)(360 °)

For a system with a 1-kHz bandwidth and an A/D converter with a 10-μs conversion time, the A/D converter will contribute a negative phase shift of 3.6 degrees.

Sampling Rate Selection

Another important consideration is the selection of sampling rate. In signal processing, the sampling rate should be at least twice the bandwidth or twice the highest frequency component in the system. If lower sampling rates are selected, noise from the high-frequency components may be introduced into the system and would be indistinguishable from the signal. Antialiasing filters are installed before the controller so that high-frequency components can be attenuated. In control systems, the sampling rate is commonly chosen to be ten to twenty times the system's bandwidth. However, this refers to the closed-loop bandwidth for the controller. If the system has structural resonances so that notch filters are needed to cancel them, a sampling rate of two times the bandwidth or higher is sufficient for the filters.

Theoretically, a digital system should be equivalent to an analog system if the sampling rate is very high. However, in practice, when the sampling frequency becomes too high, the poles will cluster around z=1, making the system more susceptible to coefficient quantization. Modifying the structure may be necessary to minimize this effect.

Another factor that needs to be taken into account is stability. When a stability analysis is done by mapping pole locations or eigenvalues on the z-unit circle, it is true only for that sampling frequency. As the sampling frequency is changed, it creates a new mapping of eigenvalues on the unit circle.

Table 1 shows the pole locations for various sampling frequencies of a lead-notch controller, which is transformed into the z domain using the bilinear transformation. The lead-notch controller is given by

$$G_c(s) = \left[\frac{(s + 0.35)}{(s + 8)}\right]\left[\frac{(s^2 + 0.06s + 1.2)}{(s + 27)^2}\right]$$

In addition to the controller's sampling rate, the sensor's bandwidth needs to be considered. Sensors like encoders give digital outputs. At high sampling rates and low speeds, their outputs may be heavily quantized, causing large variations from sample to sample. Taking a moving average of the last few samples may be necessary to eliminate those variations. This essentially implements a low-pass filter for the input signal.

Table 1. Location of Poles for a Lead-Notch Controller

	Pole Locations		
Sampling Frequency (kHz)	*Pole 1*	*Pole 2*	*Pole 3*
0.1	0.7621145	0.7621145	0.9230769
0.5	0.9474196 ± *j*2.316677E–08		0.9841269
1	0.9733596 ± *j*1.411779E–08		0.9920318
5	0.9946145	0.9946145	0.9984012
10	0.9973036 ± *j*2.567057E–08		0.9992003
50	0.9994601	0.9994601	0.9998400
100	0.9997300 ± *j*2.106205E–08		0.9999200

Antialiasing Filters

In a digital signal processing system, a minimum sampling rate must be implemented to allow reconstruction of the information in the digital domain. According to the Nyquist criteria, the sampling frequency must be at least twice the highest frequency component in the signal. If a lower sampling frequency is used or if high-frequency noise is present, some of the information will be lost. This is known as aliasing. If unwanted high-frequency components are present, they must be removed through circuits known as antialiasing filters.

In control systems, antialiasing filters must be used carefully; they can cause phase delay, which also adds to the computational delay of the controller. A negative phase shift affects the phase margin of that system. Due to the oversampling intervals (10 – 20 times the frequency) in control systems, it is usually possible to avoid the usage of antialiasing filters. If antialiasing filters are used, they should be first-order filters with minimum phase delay. The negative phase-shift contribution of the filter should be taken into account along with the computational delay and A/D conversion delay.

Controller Design Tools

Analog controller implementation requires only hardware design. A digital controller implementation not only requires a hardware design, but also extensive software design. The hardware design of a digital controller is somewhat easier to accomplish, and standard forms of processor interface can be chosen independently of the type of controller structure selected. The burden of software design can be eased by the wide selection of CASE (Computer Aided Software Engineering) and code-generating tools that are available today. These tools tremendously increase the productivity of the control designer.

Algorithm Development

In control systems, extensive simulation of control algorithms is necessary before the design can be carried out. Simulations may also be necessary under worst-case conditions so that appropriate scaling factors can be obtained. Numerous software packages are available that allow not only simulation capability but also design capability. As mentioned earlier, some of the more popular packages are PC-Matlab, Matrix-X, and Simnon. The Impex® software package also has extensive simulation capabilities. It supports simulation with A/D and D/A converters and the effects of the converters' resolution and conversion delay. It can also comprehend computational delays and different levels of quantization on all or some of the states.

Software Development

Software development is another major concern in implementing digital controllers. The programmable approach to controllers allows easy upgrade and maintenance. It protects development investment but, at

the same, requires more initial development effort. Still, programming with DSPs requires slightly different techniques than programming with ordinary processors.

Typically, in control systems, processors are used for supervisory functions, and analog circuits are used for signal processing functions. When DSPs are used, they may be required to implement not only the signal processing functions but also the supervisory functions. With ordinary processors, there is usually a large reliance on lookup tables for math and other functions. With DSPs, it is more common to calculate the actual math functions or the algorithms. Functions like sine and cosine may be easily calculated using the expansion series. Due to the high speed of DSPs, it is very common to eliminate as much of the external hardware as possible and, instead, use on-chip processing for those functions. For example, low-cost sensors could be used; or, some of the sensors can be eliminated entirely, and on-chip processing can compensate for their removal.

DSPs have been designed for realtime signal processing and have very fast interrupt response. On earlier processors, the facilities for concurrently running multiple tasks were limited due to their smaller-sized hardware stacks, although larger software stacks were possible. This reduced the number of nested interrupts or subroutines that the processors could handle. Therefore, it is normally advisable to use macros and the straight-line code instead of repeated subroutine calls.

DSPs do not have a single-cycle divide instruction, so division should be avoided. If necessary, the first choice is a multiplication by an inverse procedure. Division can also be performed by repetitive executions of the SUBC instruction. Or, a limited division can be performed by right-shift operations.

Four different approaches to software development can be taken: high-level languages, assembly language, signal processing languages, and code generation software.

High-Level Languages: Using a high-level language (HLL) like C, Pascal, or FORTRAN can substantially cut development effort. Such languages are familiar to everybody and easy to program. Typically, high-level languages are used for initialization and nonrealtime code. They are not optimized with respect to signal processing functions and to particular processor architectures. Code compiled on a processor is always larger than handwritten assembly code and may be 2 to 4 times the size of assembly code; this is a high penalty trade-off for time-critical signal processing applications. In cases where a high-level language is necessary, it is beneficial to have a thorough knowledge of processor architecture to make the most efficient use of the special signal processing features.

Due to the general trend towards more usage of HLL in industry, new TMS320 architectures are also being optimized for HLL. Floating-point generations (TMS320C3x and TMS320C4x) of the TMS320 family have architectures especially designed for greater support of high-level language code and produce highly efficient assembly code. On the other hand, the fixed-point generations may require assembly coding for their time-critical routines.

Assembly Language: Assembly language produces the most efficient coding. When using a high-level language, it may be necessary to use assembly language for the more time-critical operations. Assembly language programming requires an intimate knowledge of the processor architecture. At the same time, the nature of performance requirements for some signal processing systems requires maximum code efficiency, leaving very little choice in the usage of assembly language. To give assembly language some resemblance of high-level language, macro libraries are often developed for more frequently used functions.

Signal Processing Languages: Signal processing languages can provide a middle ground between high-level language coding and assembly language coding. They can ease the development of standard high-level languages. At the same time, they offer code efficiency that is comparable to that of assembly

language because they are designed for specific signal processing applications. Digital signal processing language (DSPL) from dSPACE is one example. One disadvantage is that there is no standard for these languages, and none of the languages is widely known.

Code Generation Software: Code generation packages that will automatically generate assembly code for particular processors are becoming available. For example, the Impex® software package from dSPACE will generate TMS320 assembly code from a mathematical description of the controller. The DFDP (Digital Filter Design Package) from ASPI will generate assembly code for TMS320 processors from a description of a filter. These packages are becoming increasingly popular because they allow the control designer to focus on design issues instead of developing assembly language software.

Device Simulators

Another useful tool in designing software is the device simulator. Simulators for the TMS320 family run on common platforms like PC and VAX, which provide full simulation of the instruction set along with instruction timing. Such simulation of the controller software can fully check the effects of math operations on internal registers and memory without the need for off-chip hardware. In some cases, software simulators have features that are not available on hardware development tools. These include full access to and tracing of internal processor memory and registers and sometimes even internal pipeline operations. Also available are full breakpoint capabilities for the inspection of the processor's state at the required/desired instances.

Hardware Design

A wide variety of tools is available for designing the hardware for a controller. These include target systems and EVMs that plug into a PC or are stand-alone. The in-circuit emulators can be used for complete system debugging. The XDS/22 emulators from TI support complete in-circuit emulation along with extensive breakpoint and tracing capabilities. Also available are device behavioral models that can simulate the timing and bus behavior of a complete target system without additional hardware. Logic Automation provides behavioral models for most members of the TMS320 family that run on popular workstations. Manufacturers like HP and Tektronix produce logic analyzers that can be used for extensive tracing. These logic analyzers can debug code by disassembling captured data.

Figure 4 shows the typical block diagram of a digital controller. A digital controller normally requires a processor, a memory interface to the processor, and A/D and D/A converter interfaces. Figure 5 shows a typical interface of a TMS320 DSP with memory and A/D and D/A devices. Further information is available in the appropriate TMS320 user's guides.

Figure 4. Digital Controller

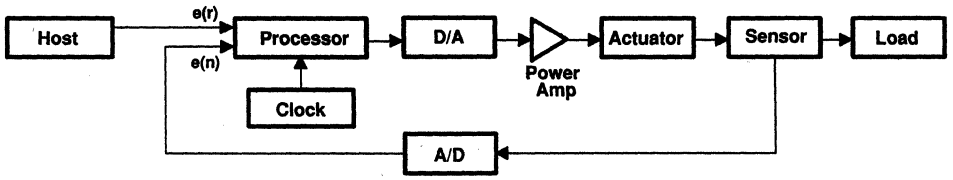
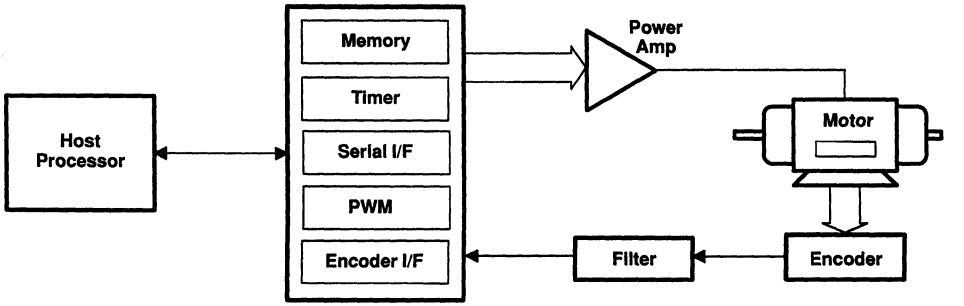


Figure 5. Controller Interface



Summary

Implementation of digital controllers is a relatively new area as the limited availability of information suggests. Most of the previous commercial implementations in industry were either first- or second-order systems. Typically, these are low-bandwidth systems like process control and do not take full advantage of the capabilities that modern control theory has to offer. Limitations of earlier processors had prevented widespread use of digital controllers in many segments of industry. DSPs are the first class of processors that have the right combination of architecture, performance, and cost to make it possible for implementing these advance concepts in practical everyday systems. This combination now allows people to implement advanced controllers in a wide variety of products and services and to solve the major problems in implementation of digital controllers. PART IV's introduction as well as articles describe many of these products and applications.

Digital controller implementation, however, is fundamentally different from analog controller implementation. Since natural analog processes are approximated, a fair amount of work must be done in preparing a controller design for implementation. This introduction highlighted some of the major problems that are usually encountered when implementing digital controllers. Undoubtedly, there are countless other problems that are unique to each application. However, minimizing these problems that are discussed here will provide a solid foundation for control system implementation. The use of CASE tools like Matrix-X, Impex, and DFDP is again recommended because they not only automate design and implementation processes but also represent years of experience by experts.

References

1. Maroney, P., *Issues in the Implementation of Digital Feedback Compensators*, The MIT Press, 1983.

APPENDIX 1

This shows an example of a design and implementation using CASE Tools. The controller was designed in the previous section using PC-Matlab. The pole locations were chosen to be z=0.90 and z=0.95. The following design parameters were obtained.

$$A = \begin{bmatrix} 1.00000000000000 & 0.00099444139773 \\ 0 & 0.98890343243454 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.00002685315106 \\ 0.05360660659645 \end{bmatrix}$$

$$C = [\ 1 \quad 0\ ]$$

$$D = [\ 0\ ]$$

$$K = [\ 93.27208561511948 \quad 2.54443979371671\ ]$$

$$L = 100\begin{bmatrix} 0.01063903432437 \\ 2.78492351899385 \end{bmatrix}$$

$$A - BK - LC = 100\begin{bmatrix} -0.00066408081842 & 0.00000926115172 \\ -2.83492351899385 & 0.00852504649404 \end{bmatrix}$$

The Impex software will be used for code generation that is suitable for implementation on the TMS320E14. The next sections of Appendix 1 show the different outputs of the software.

1a. This shows the original system derived from PC-Matlab and the input to the Impex software. The matrices A, B, K, and L have to be combined as shown above and will be referred to as the "**a**" matrix in the system. The remaining matrices will remain the same.

1b. This shows the effect of schur transformation in the system. The dynamic range of the coefficients has been significantly reduced.

1c. This shows the system after scaling and schur transformation. The C matrix is not scaled as this can be done via input output scaling or even with an external amplifier.

1d. This shows the realized system and the DSPL (Digital Signal Processing Language) code for the state controller/estimator.

1e. This shows the assembly language code for this controller on the TMS320E14 DSP. The code also shows the macros that will be used in the expansion. The code interface to a DS1101 (a TMS320E14 board developed by dSPACE). Initialization and peripheral addresses can be changed for other systems.

Appendix 1a

```
- This is the original system obtained from PC-Matlab
- Dynamic_matrix a represents A - BK - LC in the design.

basic_block is
  state controller/estimator

system_info_text is
  example for a second order state controller/estimator
  with one input and one output.
end system_info_text;

sampling_period := 0.001;

system_inputs is
  name => pos_err, unit => V,
    lower_bound => -1.00000000000000E+01, upper_bound =>
1.00000000000000E+01;
end system_inputs;

system_outputs is
  name => plant_con, unit => V,
    lower_bound => -1.00000000000000E+01, upper_bound =>
1.00000000000000E+01;
end system_outputs;

system_equations ssd is

  system_representation := PHYSICAL;

  system_states is
    name => state_x1;
    name => state_x2;
  end system_states;

  dynamic_matrix is
    a( 1, 1) := -6.66408081842000E-02;
    a( 2, 1) := -2.83492351899385E+02;
    a( 1, 2) :=  9.26115172000000E-04;
    a( 2, 2) :=  8.52505649404000E-01;
  end dynamic_matrix;

  column_input_matrix pos_err is
    b( 1) :=  2.68531510600000E-05;
    b( 2) :=  5.36066065964500E-02;
  end column_input_matrix;

  row_output_matrix plant_con is
    c( 1) :=  1.00000000000000E+00;
  end row_output_matrix;

  direct_link pos_err to plant_con is
    d       :=  0.00000000000000E+00;
  end direct_link;

end system_equations;

end basic_block;
```

```
- This shows the controller after performing schur
- transformation on it.

basic_block is
  state controller/estimator

system_info_text is
  example for a second order state controller/estimator
  with one input and one output.
end system_info_text;

sampling_period := 1.00000000000000E-03;

system_inputs is
  name => pos_err, unit => V,
    lower_bound => -1.00000000000000E+01, upper_bound =>
1.00000000000000E+01;
end system_inputs;

system_outputs is
  name => plant_con, unit => V,
    lower_bound => -1.00000000000000E+01, upper_bound =>
1.00000000000000E+01;
end system_outputs;

system_equations ssd is

  system_representation := SCHUR;

  system_states is
    name => state_x1_schur;
    name => state_x2_schur;
  end system_states;

  dynamic_matrix is
    a( 1, 1) := -6.66408081842000E-02;
    a( 2, 1) := -5.53695999803486E-01;
    a( 1, 2) :=  4.74170968064000E-01;
    a( 2, 2) :=  8.52505649404000E-01;
  end dynamic_matrix;

  column_input_matrix pos_err is
    b( 1) :=  1.37488133427200E-02;
    b( 2) :=  5.36066065964500E-02;
  end column_input_matrix;

  row_output_matrix plant_con is
    c( 1) :=  1.95312500000000E-03;
  end row_output_matrix;

end system_equations;

end basic_block;
```

```
- This shows the controller after performing schur transformation and
scaling on it

basic_block is
state controller/estimator

system_info_text is
  example for a second order state controller/estimator
  with one input and one output.
end system_info_text;

sampling_period := 1.00000000000000E-03;

system_inputs is
  name => pos_err_scaled,
    lower_bound => -1.00000000000000E+00, upper_bound =>
1.00000000000000E+00;
end system_inputs;

system_outputs is
  name => plant_con_scaled,
    lower_bound => -1.00000000000000E+00, upper_bound =>
1.00000000000000E+00;
end system_outputs;

system_equations ssd is

  system_representation := SCHUR;

  system_states is
    name => state_x1_schur_scaled;
    name => state_x2_schur_scaled;
  end system_states;

  dynamic_matrix is
    a( 1, 1) := -6.66408081842000E-02;
    a( 2, 1) := -3.05727555099513E-01;
    a( 1, 2) :=  8.58759911760409E-01;
    a( 2, 2) :=  8.52505649404000E-01;
  end dynamic_matrix;

  column_input_matrix pos_err_scaled is
    b( 1) :=  2.04659364615941E-01;
    b( 2) :=  4.40603476246127E-01;
  end column_input_matrix;

  row_output_matrix plant_con_scaled is
    c( 1) :=  1.31209002384973E-04;
  end row_output_matrix;

end system_equations;

end basic_block;
```

- This shows the realized system and the DSPL code to implement it.

```
system realization   linear_system is
  2nd order state controller/estimator
type fractional is
  fix'(bits => 16,
       fraction => 15,
       representation => twoscomplement);

scptype state1 is
  fix'(acculength => 32,
       round => on,
       scale => on,
       saturation => on);

scptype out1 is
  fix'(acculength => 32,
       round => on,
       scale => common,
       saturation => on);

a1 : scalable constant vector (2) of fractional
  := ( -6.665039062500E-002 ,
        8.587646484375E-001 );
a2 : scalable constant vector (2) of fractional
  := ( -3.057250976563E-001 ,
        8.525085449219E-001 );
b1 : scalable constant vector (1) of fractional
  := (  2.046508789063E-001 );
b2 : scalable constant vector (1) of fractional
  := (  4.406127929688E-001 );
c1 : scalable constant vector (2) of fractional
  := (  1.220703125000E-004 ,
        0.000000000000E+000 );

xk  : vector (2) of fractional;
xk1 : vector (2) of fractional;
u   : vector (1) of fractional;
input is u;
y   : vector (1) of fractional;
output is y;

begin
  every   1.000000000000E-003 do
    update (xk1, xk);
    input (u);
    output (y);
    accumulate scalpro (state1, 1.000000000000E+000 )
      xk1(1) := a1 * xk + b1 * u;
    end accumulate ;
    accumulate scalpro (state1, 1.000000000000E+000 )
      xk1(2) := a2 * xk + b2 * u;
    end accumulate ;
    accumulate scalpro (out1, 1.000000000000E+000 )
      y(1) := c1 * xk1;
    end accumulate ;
  end every ;
end linear_system;
```

```
; TMS320C14 assembly code for the controller/estimator
;
        .title "linear_system"
        .list                   ; enable listing
;
        .global RESET           ; user program entry
;
; ========================================================== ;
;                                                            ;
; code for DSPL's initialization                             ;
;                                                            ;
; standard version                                           ;
;                                                            ;
; version for DS1101 TMS 320C14 / E14 processor board        ;
;                                                            ;
; WARNING : no interrupt besides TIMINT1 must be used !!!    ;
;                                                            ;
; revision 2.01 / 09-Nov-1989                                ;
;                                                            ;
; (C) 1989 dSPACE GmbH                                       ;
;                                                            ;
; ========================================================== ;

init    $macro callno,blkno

TIMINT1_bit?  .set    16

bsr?          .set    7
ddr?          .set    1
if?           .set    4
im?           .set    5
fclr?         .set    6
adc0?         .set    8
adc1?         .set    9
strb?         .set    0AH
comreg?       .set    0EH

;
; initialize RESET vector and INT vector
;
        .asect "RESET", 0

        b    RESET              ; vector to user program entry
        b    ISR                ; vector to interrupt dispatcher
;
        .asect "DSPL"           ; return to DSPL compiler's code section

;
; define initial processor state
;
        dint                    ; disable interrupts
        rovm                    ; disable hardware overflow mode
;
; initialize constant because INIT is called before DSPL code
; transfers data to on-chip RAM
;
        lack  1
        sacl  one
```

```
;
; initialize interrupt system
;
       zac
       sacl   *
       out    *, bsr?            ; select BANK0
       out    *, ddr?            ; configure parallel port as input
       sub    one
       sacl   *
       out    *, im?             ; mask off all interrupts
       out    *, fclr?           ; clear all interrupt flags
;
; dummy read 12 bit ADCs to enable ADC operation
;
       lack   adc0?
       tblr   *
       lack   adc1?
       tblr   *
;
; dummy read communications port to reset rxfull flag
;
       lack   comreg?
       tblr   *
;
; clear incremental encoder counter registers
;
       lack   30H
       sacl   *
       lack   strb?
       tblw   *

       b      exit?              ; initialization complete

;
; interrupt service routine
;
ISR    in     *, if?             ; read interrupt flag register
       lack   TIMINT1_bit?
       and    *
       bz     no_TIMINT1?
       sacl   *, 0
       out    *, fclr?
       call   timint1
no_TIMINT1?
       eint
       ret

exit?

       .endm
;
```

```
; ================================================================== ;
;                                                                    ;
; code for DSPL's EVERY-statement (begin)                            ;
;                                                                    ;
; standard version                                                   ;
;                                                                    ;
; version for TMS 320C14 / E14 on-chip timer 1                       ;
;                                                                    ;
; formal parameter TIME passes requested sampling period in  s       ;
;                                                                    ;
; 160  ns <= sampling_period <= 10.4 ms, resolution 160 ns           ;
; 10.4 ms <  sampling_period <= 41.9 ms, resolution 640 ns           ;
; 41.9 ms <  sampling_period <= 65.5 ms, resolution 2.56  s          ;
;                                                                    ;
; revision 2.01 / 09-Nov-1989                                        ;
;                                                                    ;
; (C) 1989 dSPACE GmbH                                               ;
;                                                                    ;
; ================================================================== ;

evbeg  $macro callno,blkno,time

time?  .set  :time:
bsr?   .set  7
bank0? .set  0
im?    .set  5
fclr?  .set  6
bank2? .set  2
tcon?  .set  4
tpr1?  .set  1
t1int? .set  0010H

;
; on-chip timer setup TMR1
;

       lack  bank2?
       sacl  *, 0
       out   *, bsr?            ; select BANK2

       $if time? < 03333H
       lack  006H               ; prescale 0
       $else
       $if time? < 0CCCCH
       lack  002H               ; prescale 4
       $else
       lack  004H               ; prescale 16
       $endif
       $endif

       sacl  *, 0
       out   *, tcon?           ; update TCON

       lt    one
       mpyk  tpr?
       pac
       tblr  *                  ; load timer period value
       out   *, tpr1?           ; set TPR1

       lack  bank0?
```

```
        sacl   *, 0
        out    *, bsr?            ; select BANK0

        lt     one
        mpyk   imval?
        pac
        tblr   *
        out    *, im?             ; set IM register

        lack   t1int?
        sacl   *, 0
        out    *, fclr?           ; clear TMR1 interrupt flag bit

        eint                      ; enable interrupts
        b      $                  ; wait for interrupt

        $if time? < 03333H        ; if period < 13.107 ms
tpr?    .word time? * 5
        $else
        $if time? < 0CCCCH        ; if period < 52.428 ms
tpr?    .word time? * 5 / 4
        $else
tpr?    .word time? * 10 / 32
        $endif
        $endif

imval? .word ~t1int?

timint1

        .endm
;

; ================================================================== ;
;                                                                    ;
; code for DSPL's  EVERY-statement (end)                             ;
;                                                                    ;
; standard version                                                   ;
;                                                                    ;
; version for TMS 320C14 / E14 on-chip timer 1                       ;
;                                                                    ;
; revision 2.01 / 09-Nov-1989                                        ;
;                                                                    ;
; (C) 1989 dSPACE GmbH                                               ;
;                                                                    ;
; ================================================================== ;

evend   $macro callno,blkno,time

        ret

        .endm
;
```

```
; ================================================================ ;
;                                                                  ;
; code for DSPL's  INPUT-statement                                 ;
;                                                                  ;
; standard version                                                 ;
;                                                                  ;
; version for DS1101 on-board 12 bit ADCs                          ;
;                                                                  ;
; revision 2.01 / 09-Nov-1989                                      ;
;                                                                  ;
; (C) 1989 dSPACE GmbH                                             ;
;                                                                  ;
; ================================================================ ;

in12    $macro  callno,blkno,data,channel

iop?    .set    0

        lack    1 << (:channel:-8) ; setup busy test mask
wait?   in      *, iop?            ; get busy bit
        and     *                  ; test busy bit
        bnz     wait?              ; wait until adc ready

        lack    :channel:          ; read adc data
        tblr    :data:

        .endm
;

; ================================================================ ;
;                                                                  ;
; code for DSPL's  START macro                                     ;
;                                                                  ;
; version for DS1101 on-board 12 Bit ADCs                          ;
;                                                                  ;
; revision 2.01 / 31-Oct-1989                                      ;
;                                                                  ;
; (C) 1989 dSPACE GmbH                                             ;
;                                                                  ;
; ================================================================ ;

start   $macro callno,blkno

strb    .set   0AH

        lack   003H               ; strobe for ADC0 .. 1
        sacl   *
        lack   strb
        tblw   *                  ; start both ADCs

        .endm
;
```

```
; ================================================================ ;
;                                                                  ;
; code for DSPL's  OUTPUT-statement                                ;
;                                                                  ;
; version for DS1101 on-board 14 bit DACs                          ;
;                                                                  ;
; revision 2.01 / 31-Oct-1989                                      ;
;                                                                  ;
; (C) 1989 dSPACE GmbH                                             ;
;                                                                  ;
; ================================================================ ;

out14   $macro callno,blkno,data,channel

        lack    :channel:       ; write data to DAC
        tblw    :data:

        .endm
;
        .asect "DSPL", 00010h   ; program memory base address
;
; status register save location (data page 1)
_st     .set  000ffh
; predefined constants
_c1     .set  00000h            ; predefined constant
        .word 1
_c2     .set  00001h            ; predefined constant
        .word 32767
_c3     .set  00002h            ; predefined constant
        .word -32768
_c4     .set  00003h            ; predefined constant
        .word -1
; declarations for UPDATE variables
_v1     .set  00004h            ; xk1(1)
        .word 0
_v2     .set  00005h            ; xk(1)
        .word 0
_v3     .set  00006h            ; xk1(2)
        .word 0
_v4     .set  00007h            ; xk(2)
        .word 0
; declarations for variable vectors
_v5     .set  00008h            ; u(1)
        .word 0
_v6     .set  00009h            ; y(1)
        .word 0
; declarations for coefficients
_c5     .set  0000ah            ; a1(2)
        .word 28140
_c6     .set  0000bh            ; b1(1)
        .word 6706
_c7     .set  0000ch            ; a2(1)
        .word -10018
_c8     .set  0000dh            ; a2(2)
        .word 27935
_c9     .set  0000eh            ; b2(1)
        .word 14438
; declarations for external procedures
one     .set  0000fh            ; constant for procedure init
```

```
        .word 1
zero    .set  00010h            ; constant for procedure in12
        .word 0
_v7     .set  00011h            ; parameter for procedure in12
        .word 0
_v8     .set  00012h            ; parameter for procedure out14
        .word 0
;
; start of program
;
RESET
        lark  ar1, 000e7h       ; initialize software stack pointer
        larp  ar1               ; make stack accessible
        ldpk  000h              ; select data page
        init  0,1               ; call external procedure init
; perform data RAM initialization
        lark  ar1, 19           ; initialize counter
        lark  ar0, 00000h       ; initialize destination pointer
        lack  00010h            ; initialize source pointer
_l1
        larp  ar0               ; select destination pointer
        tblr  *+, ar1           ; transfer word, select counter
        add   _c1               ; increment source pointer
        banz  _l1               ; repeat until transfer complete
        lark  ar1, 000e7h       ; initialize software stack pointer
        larp  ar1               ; make stack accessible
; line 42
        evbeg 0,1,1000          ; begin block statement
;
; ---- 16 cycles
;
; line 43
        ldpk  000h              ; select data page
        dmov  _v1               ; xk1(1) --> xk(1)
        dmov  _v3               ; xk1(2) --> xk(2)
;
; ---- 3 cycles
;
; line 44
        start 0,1               ; initialize input
        in12  0,1,_v5,00008h    ; input u(1)
;
; ---- 56 cycles
;
; line 45
        out14 0,1,_v6,00008h    ; output y(1)
;
; ---- 4 cycles
;
; line 46
        zac
        lt    _v2               ; xk(1)
        mpyk  -2184             ; a1(1)
        lta   _v4               ; xk(2)
        mpy   _c5               ; a1(2)
        lta   _v5               ; u(1)
        mpy   _c6               ; b1(1)
        apac
        add   _c1, 14           ; perform rounding
; overflow test and rescaling 0 bit
```

```
        sach  *, 1              ; save result
        blz   _l2               ; branch if result negative
        sub   _c2, 15           ; positive limit
        blez  _l3               ; branch if no positive overflow
        lac   _c2, 0            ; use positive saturation
        b     _l4               ; update result
_l2
        sub   _c3, 15           ; negative limit
        bgez  _l3               ; branch if no negative overflow
        lac   _c3, 0            ; use negative saturation
        b     _l4               ; update result
_l3
        lac   *, 0              ; reload result
_l4
        sacl  _v1, 0            ; xk1(1)
;
; ---- 19 cycles
;
; line 49
        zac
        lt    _v2               ; xk(1)
        mpy   _c7               ; a2(1)
        lta   _v4               ; xk(2)
        mpy   _c8               ; a2(2)
        lta   _v5               ; u(1)
        mpy   _c9               ; b2(1)
        apac
        add   _c1, 14           ; perform rounding
; overflow test and rescaling 0 bit
        sach  *, 1              ; save result
        blz   _l5               ; branch if result negative
        sub   _c2, 15           ; positive limit
        blez  _l6               ; branch if no positive overflow
        lac   _c2, 0            ; use positive saturation
        b     _l7               ; update result
_l5
        sub   _c3, 15           ; negative limit
        bgez  _l6               ; branch if no negative overflow
        lac   _c3, 0            ; use negative saturation
        b     _l7               ; update result
_l6
        lac   *, 0              ; reload result
_l7
        sacl  _v3, 0            ; xk1(2)
;
; ---- 19 cycles
;
; line 52
        zac
        lt    _v1               ; xk1(1)
        mpyk  4                 ; c1(1)
        apac
        add   _c1, 14           ; perform rounding
; overflow test and rescaling 0 bit
        sach  *, 1              ; save result
        blz   _l8               ; branch if result negative
        sub   _c2, 15           ; positive limit
        blez  _l9               ; branch if no positive overflow
        lac   _c2, 0            ; use positive saturation
        b     _l10              ; update result
```

```
_18
        sub     _c3, 15             ; negative limit
        bgez    _19                 ; branch if no negative overflow
        lac     _c3, 0              ; use negative saturation
        b       _110                ; update result
_19
        lac     *, 0                ; reload result
_110
        sacl    _v6, 0              ; y(1)
;
; ---- 15 cycles
;
; line 55
        evend 0,1,1000              ; end block statement
;
; ---- 2 cycles
;
        b       $                   ; wait for interrupt
        .end
```

HARDWARE / SOFTWARE-ENVIRONMENT FOR DSP-BASED MULTIVARIABLE CONTROL

H. Hanselmann, H. Henrichfreise, H. Hostmann and A. Schwarte
dSPACE digital signal processing and control engineering GmbH
An der Schönen Aussicht 2, D-4790 Paderborn, Fed. Rep. Germany

Abstract

Single-chip Digital Signal Processors (DSP) are powerful candidates for the implementation of multivariable control for fast systems. We report briefly on several applications of DSP in the control of mechanical systems. The success of these applications was to a large extent due to a set of software and hardware tools for controller implementation. Building upon our experiences of these applications we derive requirements and concepts for a novel development system (hardware/software-environment) for DSP in multivariable control.

Current DSP

The reason for considering DSP for control is their computing speed. In most other respects DSP are inferior to other kinds of processors [1,2]. The speed of DSP comes mainly from the integrated hardware multiplier and accumulator, and from the multiple bus architecture. The latter is necessary in order to keep the fast arithmetic units busy, i.e. to allow the operand and result data transfers to keep up with the usually single-cycle arithmetic operations.

A detailed description of DSP architectures is not given here. Some comparisons of current DSP chip architectures can be found in [3,4]. A few benchmark results related to control are mentioned in [1,2] and some more are reported below in the applications section.

The spectrum of DSP has grown rather broad now. It is divided into two blocks: one with fixed point and one with floating point arithmetic hardware.

The low end is represented by low cost devices such as the Texas Instruments TMS32010 with 16 bit fixed point arithmetic (32 bit in the accumulator) and rather limited data memory address range (144 words on-chip), which needs 400 ns for a multiply-and-accumulate operation (mac). In the medium range are devices which also support 16 bit fixed point arithmetic but are about twice as fast, have increased addressing space, and have increased functionality (such as on-chip serial interfaces). One example is the TMS320C25. High end fixed point arithmetic chips are the AT&T DSP16 with its speed (75 ns per mac), and the Motorola DSP56000 with its extended wordlength (24 bit operands and 56 bit in the accumulator). For high volume industrial use versions with on-chip program EPROM (TMS320E15) or even EEPROM (General Instruments DSP320EE12) are particularly interesting.

A few floating point DSP have become available recently, most notably the NEC77230 and the AT&T DSP32. Both chips offer 32 bit arithmetic with 150 ns (NEC, pipelined) to 250 ns (AT&T) for a mac. So there is only a small time penalty for floating point arithmetic if these chips are used. Even faster will be the chips which are scheduled to be sampled in 1988/1989 such as the AT&T DSP32C (up to 80 ns per mac) and the Texas Instruments TMS32030 (60 ns per mac).

These chips will use 0.75 μm and 1 μm technology. The same technology will enable fixed point chips to be faster, but what is often more important for industrial use, the chip area saved by sticking to fixed point hardware can be used to increase the chip's functionality by integrating more timers, ports, interrupt control etc.. Microcontrollers like the Intel 8096 but with DSP core may be created that way. Using more conventional technology will on the other hand lower chip cost and thus open up high-volume applications. Fixed point DSP will have a place in industrial applications for years to come.

Applications

In this section we report briefly on some multivariable control applications using DSP. Unless otherwise stated these are applications we were involved in during our work at the Department of Automatic Control in Mechanical Engineering at the University of Paderborn.

Winchester disc-drive actuators

Modern high performance disc drives use fast voice coil actuators for the positioning of magnetic heads onto desired tracks and for keeping them on track against various disturbances by closed-loop control. Head positioning control comprises two tasks: (A) Positioning on a target track (maybe across many tracks), and (B) track following during read and write operations. Modern control techniques can be expected to improve control speed and accuracy for both tasks.

For task (A) state estimator techniques help to solve the problem of estimating the state of the fast moving actuator from the track error, which is the only measurement variable usually available. For task (B) controllers can be designed which achieve high control bandwidth and good disturbance rejection despite the complicated nature of the mechanical plant.

Using a simple low order model (double integrator) for the actuator an estimator-based controller was implemented on an Intel 8096 microcontroller by IBM [5]. Owing to the medium performance embedded servo technique, the crossover frequency (around 300 Hz) and the sampling rate (around 4 kHz) were not very high and the controller was relatively unambitious with respect to processor computing speed.

The computing power of a TMS32010 DSP was utilized in the track following control studies reported in [6,7]. A 9th order controller based on notch filter techniques (to compensate for structural resonance effects) was designed and implemented, running at about 30 kHz sampling rate [6]. A crossover frequency around 900 Hz was achieved. The crossover frequency was limited mostly by model uncertainty, but the high sampling rate was not a luxury because of strong resonances in the plant even at 10 kHz. This controller was for an 8 inch drive with dedicated servo and a rotary voice coil actuator. A disturbance observer with disturbance feedforward was added to the 9th order controller for improved disturbance rejection.

A different controller with excellent disturbance rejection based on lq (linear quadratic optimal) controller design for the same drive was also implemented and ran at 34 kHz sampling rate [7].

Tailoring positioning controllers to modelled disturbance dynamics, incorporating adaptive techniques, or increasing the usable frequency range of the mechanical construction (smaller drives and better construction) will further increase processor power demand. So disc drives are an interesting field for DSP application.

Active or semiactive vehicle suspension

Active vehicle suspension means total replacement of the conventional spring and shock absorber assemblies. Hydraulic cylinders driven by servovalves are used instead. The system relies fully on control [8].

The abovementioned group at the University of Paderborn has been working on this subject for years under contract with several groups of Daimler Benz AG. Multivariable control techniques are applied. Multivariable controllers with more than 10 sensor inputs, 4 actuator outputs, several diagnostic outputs, and orders above 20 are common. These controllers are mostly linear with some added lumped nonlinearities for the compensation of nonlinear hydraulic

flow phenomena. Fast dynamics of the hydraulic systems require sampling rates above 1 kHz. After single axis test-bed studies some years ago (already using DSP) an experimental off-road truck is currently being equipped to run tests in the field. A study for another type of vehicle is underway. TMS32010 systems were used until recently, and have now been replaced by TMS32020 systems now.

In preparation for the off-road truck test the cylinder construction was tested at the university lab in a hardware-in-the-loop simulation. The real cylinder, which is to replace the spring/absorber assembly, was used. The road and the vehicle body were simulated in a TMS32010, together with the suspension controller and the controller for a second cylinder generating the correct dynamic load such as the suspension cylinder would find in the real vehicle. The total system could have run at 7 kHz sampling rate, somewhat more than necessary.

A fully active system has also been designed and implemented for a race car at Lotus Co., UK, also using a TMS320 processor. 17 sensors are involved.

Semiactive vehicle suspension means replacement of the conventional shock absorber by an adjustable one. In contrast to existing slowly and / or discontinuously adjustable absorbers the actuator mechanism has servovalve characteristics in order to come close to an active system in performance. Such a system is under development in an industrial company which is advised by the abovementioned university. Again a TMS32020 system is used, which replaced a TMS32010 system recently.

Elastic Robot

With conventional control, the elastic movements in the drives and the flexibility of the arms of lightweight robots result in large vibrations of the hand, particularly during and after high acceleration intervals. A multivariable controller has been designed and implemented for a three-joint articulated robot driven by electrical servo-drives [9,10]. This controller removed the vibrations virtually completely without a speed penalty.

Each motor was equipped with a position encoder and a tacho-generator and the two arms carried two strain-gages each for curvature measurements in both deflection directions. The total number of sensors was thus 10. The reference trajectory was fed into the controller as 3 position, 3 velocity and 3 acceleration feedforward signals. The controller thus had 19 inputs and 3 outputs to the motors. The order of the controller was only 6 due to the special design technique and due to the fact that many sensors were been used (many static gains). The controller was implemented on a TMS32010 and the sampling rate used was 10 kHz. The sampling rate could however have been more than twice that, so there was considerable spare computing power for additional tasks to be performed by the processor.

Hydraulic Robot

For tasks requiring medium speed but very high acceleration (such as water jet cutting) a 5 degrees-of-freedom (6 drives) gantry robot is under construction at an industrial company. Hydraulic drives have been chosen because of their good torque-to-weight ratio. The construction is novel in many respects and makes use of very lightweight materials.

Two particular challenging requirements for control design and implementation have been: (a) to maintain tough trajectory control under maximum acceleration (i.e. max. error 0.2 mm at 30 m/s²), (b) to use no other sensors than the position encoders of each hydromotor (absolute minimum).

Requirement (a) necessitated nonlinear compensation to cope with the strong nonlinearities of hydraulic flow through the servovalve. Requirement (b) was met (in the axis designs completed at the time of writing) by relying on Kalman-Filters for estimating the plant state. This was successful because high resolution position encoders are used. The trolley position for example spans 2m and the associated encoder stepsize is about 8 µm.

The trolley controller consists of a 6th order linear Kalman-Filter plus state-feedback, a connected linear 4th order subsystem for nonlinear compensation, and the nonlinearity, which requires some simple operations and a squareroot performed via table-lookup.

All drive control of the whole robot is performed by two TMS32020 boards, the sampling rates being around 5 kHz. 16 bit fixed point arithmetic is used with the exception of a few concentrated simple operations on the larger position sensor words. To keep the high resolution (large word) information out of the linear controller computations a special technique has been developed [11].

At the time of writing the controllers for some of the 6 drives have been designed and implemented up to simulations. One of the axis controllers (for the trolley) has also been tested experimentally. It worked as predicted.

Development System Requirements

In this section we specify what a development system oriented towards DSP control should comprise. For the projects of the previous section (these are not the only ones) we used several tools which have been developed over years to facilitate and in some cases even automate the implementation of nontrivial controllers on DSP. Recent relevant papers are [1,2,12,13]. A new generation of DSP control development tools is now in the making at dSPACE GmbH, building upon past experience.

General Considerations

The main line is to support controller implementation as well as is usually expected for controller design and simulation, and to do this in terms accessible to the control designer. Often not much consideration is given to implementation during design, mostly because design specialists are rarely implementation specialists as well and work is traditionally split between control theory / design people on one side and processor / electronics / programming people on the other side. It proved highly beneficial for control designers to be able to study implementation issues themselves during design, to produce DSP programs (via automatic code generators), and to carry out experiments without delegating responsibility at any stage of this process. The value of *direct feedback between design, implementation, and experiment* cannot be overestimated.

A second aspect concerns the choice of a target hardware system. For preliminary studies of implementation issues and to check feasability of the control system there should be no forced dependence on specific processors, their software, or specific target hardware. Most of all, it should not be necessary to build hardware before knowing what hardware is actually needed and sufficient. Thus it should be possible to study implementation based on flexible models of the target processor hard- and software.

When experimental evaluation is about to begin, it should still not be necessary to build special hardware in every case. A set of ready-to-use hardware components (boards fitting into a PC-AT computer for instance) is preferable whenever possible. Only after experimental validation of the design should tailoring of hardware for low cost etc. be made. We frequently observed in industry that early decisions on target hardware were made and then much engineering resources were wasted in squeezing code (e.g. to meet speed requirements problems) and dealing with secondary limitations although the controller design was not yet settled. It is much better to have a quick validation of the controller design, with the lowest implementation effort possible, and then to investigate possibilities of downsizing (memory, processor version etc.) the target hardware afterwards. This may eventually lead to custom chips with DSP cores.

Hardware

Our approach is to provide a set of processor boards all compatible with the same set of peripheral boards (ADC, DAC, decoders) so that, if desired, one may start with a floating point DSP implementation (which is the simplest), then move to a fast fixed point DSP with a large memory of the same family, then move to a lowest-cost device with more restrictions. All these steps would be carried out with the same ready-to-use peripheral boards. The last step may be to move to custom hardware if the standard boards do not meet space or economy requirements.

We choose to host the hardware on PC-AT and compatibles. This provides a convenient development environment and, by making use of industrial AT computers, an AT-host can even be useful for final products such as robot control systems.

The AT-bus is of course not used for DSP-I/O. We provide up to 32 bit wide data transfer. This is useful for the next generation of DSP and accommodates for instance the wide data words delivered

by high resolution position sensors (absolute encoders or incremental encoders with counters) used in robot control.

Processor boards: Different application fields require different types of processors . If the focus is on the experimental validation of a control concept, then high performance floating point DSP will normally be the first choice. If the focus is on producing prototypes for final products, quite different DSP may be used. In a high-volume disk drive application, for instance, the goal will be to find the lowest-cost device which is just sufficient. So there should be a number of processor boards which, as far as possible, are similar from the host-side, and which fit a single set of peripherals. Our choice is the Texas Instruments TMS family, which covers all types of DSP of interest.

Fast host-to-DSP communication is provided by means of true dual-port-RAM. On the host side DMA can be used. The DSP does not need to be halted during host access. This feature is not necessary for the development of stand-alone DSP applications, but is useful for applications such as robot control with trajectory data delivered by the host.

DSP usually have very limited interrupt control facilities. In order to allow peripherals to request service or flag their state (e.g. ADC-ready), some hardware is necessary to allow the DSP program to find out the interrupt source and its priority quickly.

Peripherals: Depending on the application fields the requirements are rather different. What is needed is a broad range of boards such as the one available on the general data acquisition market. However, our control field requirements differ in some respects. For example, in contrast to common data acquisition tasks, we sometimes need random access to input as well as output channels, and we cannot tolerate significant delays.

Random access is desired for instance because sampling rates on channels of the same board may be required to be different (multi-rate control), and even controller state-driven (instead of time-scheduled) access to channels may be necessary. Output of control signals to the actuator occurs preferably as soon as these signals are computed in order to minimize the delay introduced into the control loop. These arguments rule out FIFO-based (transient-recorder like) architectures and external constant-frequency sampling control.

For analog sensor signals 12 bit ADCs will, in our experience, in virtually all cases be sufficient for control, as well as 12 bit DACs for analog output. If the final product has to have lower resolution converters for economy reasons, it is easy to round off to any desired number of bits by very small pieces of code in experiments. Higher resolution is frequently necessary in position control, but in this case digital sensors are normally used (encoders). It would nevertheless be fine to have up to 16 bit converters available.

Successive approximation ADCs usually should have a sample/hold-circuit (SHC) at the analog side, but in control applications it is sometimes beneficial with respect to loop delay not to use the SHC [2]. If experiments prove that the SHC can indeed be omitted this may in addition decrease final product cost considerably (good and fast SHCs are not cheap). Bypassing the SHC should therefore be possible under host control.

It may sometimes be necessary to place anti-aliasing filters (AAF) in front of ADCs. In all of the applications we have ever carried out, there was only one single occasion when we needed an AAF, and this was a very simple one (first order). In contrast to many data acquisition tasks, we are reluctant to put sharp filters into the control loop because of their strong adverse effect on loop frequency response phase [2]. We consider it best to keep AAFs out of the ADC boards, and to provide optional extra boards, with programmable active filters. The most flexible architecture allows for the filters to be programmed and bypassed both under DSP and host control. Note that it is necessary to make accurate AAF frequency response models available to the controller design software, because filter dynamics must usually be taken into account in the design.

Digital sensor signals provided by absolute position encoders (multi-turn) must be accommodated (robotics). They are often wider than 16 bits and frequently supply data via special fast serial interfaces. Conversion from Gray to binary code might be done in software, but a hardware decoding facility for optional use should be available on the peripheral board.

Incremental position encoder signals should be decoded on a peripheral board. The width of the counter should not be below 24 bits. Reset by detection of a reference pulse must be possible, either by hardware or by the action of the DSP or host after it has been given notice of the reference pulse transition.

In our experiences it would be very useful for experiments on the real plant to have means of monitoring (graphic display) the sensor and control signals from the host through the same ADC or digital channels as the DSP. It is already of great help to be able to do this before the DSP is started, but monitoring sensor and control signals while the DSP is running is even better.

Commonly a set of separate measurement devices is used for such monitoring, but this is inferior to our approach for various reasons: (a) the bit patterns seen by the DSP are not recovered precisely, (b) LSB-flipping of ADCs cannot be observed, (c) possible offsets of ADCs remain undetected, (d) sampling instants are different, (e) digital sensors signals (from encoders/decoders) are usually not accommodated by measurement devices.

These deficiencies can to some extent be remedied by passing the words received by the DSP to additional monitoring outputs. However, this not only requires additional output channels but also makes necessary additions to the DSP program, which have nothing to do with the control task. With simpler DSP such as the TMS3201x family in particular, any software extension of this kind may make large software changes necessary, for instance because on-chip data memory may be only sufficient as long as no extensions are made. It is much better if the DSP program can remain undisturbed.

Software Tools

The tasks to perform when a multivariable controller such as those of the applications discussed above is to be implemented can be divided into two main blocks: (A) the preparation of a designed controller for matching the capacities of target hardware, (B) programming. Only some brief considerations can be given here. A detailed discussion of these issues is given in [12], which presents a basis for a controller implementation oriented software system.

Preparation: It is particularly with fixed point DSP that (A) is not trivial. Because arithmetic is limited in range and resolution, it is very important to select appropriate realization structures, to decide if and where extended precision arithmetic (costly) is necessary, and to scale state variables and intermediate results. It is crucial to have good methods (in tool form) for the tasks mentioned. The desire to have 32 bit floating point arithmetic is frequently due to lack of methods and tools for doing the same with 16 bit fixed point arithmetic. Note that in all our applications the latter was entirely sufficient. Many of the tasks can be automated or almost automated for linear control systems so that controller implementation becomes easy.

The situation changes with controllers with many nonlinear operations, where selection of the computational structure and scaling cannot be supported so strongly by methods and algorithms as in the linear case. But note that the hydraulic robot as well as the vehicle suspension applications had nonlinear controllers. The nonlinearities had been isolated from the larger linear parts and linear methods were applicable for preparation of the linear controller subsystems.

With 32 bit floating point DSP scaling is no longer a problem. Structure selection remains an issue but is less critical. A direct form or parallel form realization structure, for instance, which may be selected at first may still fail. For example, we encountered a case where 32 bit floating point coefficients were not sufficient to represent a 3rd order controller subsystem, but 16 bit fixed point sufficed when a parallel form was used, and we encountered just the opposite too (floating point parallel form failed and fixed point direct form was good for a 3rd order subsystem).

As described in [12] there are three main issues associated with computer assisted or automated preparation of a designed controller for implementation: (a) the representation of digital controller models, (b) model management, and (c) the tools acting upon the models.

The representation of digital controller models should be such that every piece of information about the controller is incorporated. This means for example that, in contrast to some theory oriented CACSD packages, a digital controller is much more than a collection

of z-transfer matrices or state variable matrix coefficients. Things like sampling delays (skewed sampling), ADC range and resolution, and descriptions of the type of arithmetic performed must be integrated into a model.

In [2,12] hierarchical multi-rate controllers models are considered. Our applications so far have been single-rate. But if multi-rate controllers had been supported, in one case a problem with a slow controller subsystem could have been dealt with that way instead of letting it run at the single (high) rate and using extended precision (slow subsystems run at too high rate are likely to pose precision problems). Regarding hierarchical models, describing a controller as a connection of subsystems on one level should be the minimum supported (one level listing subsystems and connections, and the level of individual subsystems descriptions).

A model management facility (database) is also necessary. Imagine a continuous controller as a starting point. Then, during implementation process iterations, several differently discretized controllers may be generated. Each of them may be transformed into several realization structures for trade-off studies, and each of these may be scaled several times under different assumptions. Then for some of these different arithmetic type and wordlength specifications may be investigated. It is clear that "management by filenames" is not sufficient here. Adding to all this, we consider it necessary that a tool creating one controller model derivate from another (e.g. scaled from unscaled) records all information which allows the tool's function to be retrieved completely later on. Such records must be logically connected to the generated model. Such requirements can only be met by some kind of specialized database.

The preparation tools should as an absolute minimum comprise the following tasks for linear controllers or controller subsystems: (1) discretization, (2) structure selection, and (3) scaling (for fixed point DSP). Some tools to analyse the effects of discretization, sampling and computational delays, coefficient quantization, and signal quantization are also very helpful. Appropriate methods have been discussed in [2,12].

And last, but not least, a simulation facility is desirable. We had very positive results with a preliminary tool which was able to simulate nonlinear plants with linear or nonlinear digital controllers, delays, ADCs and DACs, and processor arithmetic. Given complete digital controller models as outlined above and in [12], all necessary information can be derived from the model by the simulation tool. It is not necessary to have code or even know the target processor for simulation. The arithmetic can be specified. So such a simulation can be said to work with an "abstract processor model".

For our applications to date this has mostly been sufficient, because we could rely on error-free target processor code produced by our automatic code generators. Once the abstract model worked, the final real code worked too. When hand-coded parts are mixed into generated code the situation is different. It would be a great enhancement if abstract model simulation were complemented by code simulation. Common code simulators unfortunately are not designed to be operated within a closed-loop control system simulation. We think it very worthwhile to produce a simulation program which allows both abstract models as well as processor-plus-code models.

Programming: For a long time assembly language (ASM) programming was the only choice for DSP. ASM programming is generally undesirable for quick control implementation as outlined in the general considerations subsection. Some DSP have architectures and instruction sets which are less easy to use than those of general microprocessors, and, more important, there are severe restrictions with some DSP [1]. This makes ASM programming particularly unattractive for our purpose. High level language compilers have emerged, but they have difficulties in dealing with restrictions and it is likely that they produce far less optimal code than an experienced ASM programmer. This is backed by benchmark results given in [13].

After years of good experience with task-specific automatic code generators we consider it best to automatically generate DSP code from a controller model as far as possible but to provide for linking with parts which cannot be produced automatically and are hand-coded, or which are not critical and are produced by a HLL compiler. A code generator can perform optimizations which are virtually out of reach for a general purpose HLL and its compiler [13]. Such a code generator may proceed in two steps: first an intermediate language representation is derived from the model, and then this representation is compiled in ASM code. A generator / compiler following this approach is under development (almost completed) at the time of writing. The philosophy behind it as well as preliminary results are described in [13]. The intermediate language is tailored to (not restricted to) fixed point DSP and has flexible mechanisms for taking peripheral I/O into account.

For the latest floating point DSP the programming task is easier with respect to arithmetics, and for some it is also easier due to more regular architecture and instruction sets. HLL compiler writers may find it easier to produce good code then. However, floating point arithmetic does not as such mean easy ASM programming. Pipelining effects and difficult instruction sets (NEC 77230) may still make ASM programming awkward. A good compromise would probably be to have an intermediate language which is close to a general purpose HLL (as commercial HLLs for DSP already are) but to have additional language constructs which enable the compiler to produce better code by optimizations under a global viewpoint and by making use of the very special instruction constructs found in ASM instruction sets of DSP.

References

(1) H. Hanselmann, "Using Digital Signal Processors for Control" presented at IEEE Industrial Electronics Conf. IECON'86, Milwaukee, Wisconsin, Sept. 29 - Oct. 3, 1986.

(2) H. Hanselmann, "Implementation of Digital Controllers - A Survey", Automatica, Vol. 23, pp. 7 - 32, January 1987.

(3) J. Titus, EDN, pp. 163 - 176, Oct. 16, 1986.

(4) R. Gluth, "Integrierte Signalprozessoren", Elektronik, Vol. 18, pp. 112 - 125, Sept. 5, 1986.

(5) M. C. Stich, "Digital Servo Algorithm for Disk Actuator Control" in Proc. Conference on Applied Motion Control CAMC87, Minneapolis, Minnesota, June 16 - 18, 1987.

(6) H. Hanselmann and W. Moritz, "High Bandwidth Control of the Head Positioning Mechanism in a Winchester Disk Drive", IEEE Control Systems Magazine, pp. 15 - 19, Oct. 1987.

(7) H. Hanselmann and A. Engelke, "LQG-Control of a Highly Resonant Disk Drive Head Positioning Actuator", IEEE Transactions on Industrial Electronics, scheduled for February issue, 1988.

(8) J. Lückel, R. Kasper and K. Jäker, "A Practical Concept for the Active Suspension of Road Vehicles" in Preprints of 10th IFAC World Congress, Munich, Vol. 3, pp. 178 -183, 1987.

(9) H. Henrichfreise, W. Moritz and H. Siemensmeyer, "Control of a Light, Elastic Manipulation Device" in Proc. Conference on Applied Motion Control CAMC87, Minneapolis, Minnesota, pp. 57 -66, June 16 - 18, 1987.

(10) H Henrichfreise, "The Control of an elastic Manipulation Device Using DSP", will be presented at American Control Conf. ACC, Atlanta, Georgia, June 15 - 17, 1988.

(11) H. Hanselmann, "Low Resolution Implementation of High Resolution Position Control", IEEE Transactions on Automatic Control, scheduled for August issue, 1988.

(12) H. Hanselmann, "A Concept for Mostly Automatic Implementation of Control Algorithms", presented at IEEE Computer Aided Control System Design, Arlington, Virginia, sept. 24 - 26, 1986.

(13) H. Hanselmann and A. Schwarte, "Generation of Fast Target Processor Code From High Level Controller Descriptions", in Preprints of 10th IFAC World Congress, Munich, Vol. 4, pp. 90 -95, 1987.

Automatica, Vol. 23, No. 1, pp. 7-32, 1987
Printed in Great Britain.

0005-1098/87 $3.00+0.00
Pergamon Journals Ltd.

Survey Paper

Implementation of Digital Controllers—A Survey*

H. HANSELMANN†

Key Words—Digital control; microprocessor control.

Abstract—Stimulated by microprocessor technology there is increasing interest in the issues of digital control implementation. This paper reviews these issues, from algorithms through current hardware up to the various problems arising with non-ideal behaviour of digital controllers.

1. *Introduction*

For many years, theorists in the control engineering field have claimed that due to advances in microelectronics, their new algorithms could easily be implemented. Talking to the people who have to perform the implementation actually quite often reveals that nothing is that easy.

This applies even in the simplest cases of linear control, if the implementation is to be carried out under difficult conditions, e.g. without the possibility of resorting to minicomputers programmed in a high-level language using high-precision floating-point arithmetic, with plenty of speed. There are still comparatively few publications dealing with the problems of controller implementation in difficult conditions, and most of these are either from the sixties, or quite recent, stimulated by the increasing availability of microprocessors.

The situation has always been different in the related field of general digital signal processing, particularly digital filtering. The problems plaguing the implementer when he has to use fixed-point arithmetic with small wordlength were attacked by theorists persistently and systematically from the early days of digital filtering onwards. Much can be learned from this field for controller implementation, although modifications and additional research have been or are still necessary. This has been pointed out particularly in the work of Moroney, Willsky and Houpt (Willsky, 1979; Moroney *et al.*, 1980, 1981; Moroney, 1983).

The main problems with digital controller implementation as considered in this paper arise from: (a) quantization of signals and coefficients, particularly in the case of fixed-point arithmetic; (b) serial computation in a processor; (c) lack of computing speed in critical applications; and (d) lack of programming support in cases where high-level language programming is not adequate.

Because microprocessor technology is advancing rapidly, it could be argued that most of these problems are going to lose importance anyway, but there will always be implementation tasks either with demands exceeding the current capabilities of common microprocessor hardware, or with constraints that involve expending more engineering effort to get a more efficient product tailored to the application. Thus for instance fixed-point arithmetic may be attractive and sufficient if dealt with appropriately, even when fast floating-point hardware abounds.

The above list of problem sources already indicates that in this paper the scope of the term "implementation" will not be restricted to the more theoretical questions but will also include the selection and evaluation of current possible hardware. Some consequences for software tools for computer-assisted implementation within a Computer-Aided Control Engineering (CACE) environment are also discussed. On the other hand the type of control to be implemented will be restricted to a certain class, i.e. the focus is on implementation of mostly linear, time-invariant control. This is felt to be justified because even with this restricted class there are many problems to discuss and such controllers form the kernel of many control tasks. This is also the case when algorithms such as adaptation mechanisms, gain schedules and the like surround this kernel in more complicated control systems.

The organization of this paper has been chosen to reflect a quite common situation for control engineers:

—these are some control algorithms I want to realize,
—and that is promising hardware,
—but what are the issues in between? What steps must be taken in order to make use of the hardware?

Therefore, after the discussion of control algorithms and the issue of discretizing continuous controllers in Section 2 a review of current hardware is given in Section 3. Various classes of digital processors are reviewed, particularly with respect to speed and architecture. Whereas general microprocessors might allow for comfortable floating-point arithmetic, there are quite often restrictions dictating the use of short wordlength fixed-point arithmetic. There might be speed reasons for this, or the chosen hardware might even not allow for anything else. This entails many consequences, hence some basics on arithmetic are discussed in Section 4, including some "exotic" types of arithmetic.

Chronologically, once the discrete or discretized controller is known, the first step of the implementation procedure is to choose a structure for the controller, suitable for implementation with the available arithmetic. With fixed-point arithmetic at least, it is in most cases crucial to transform a discrete controller description into another description which is input–output equivalent, but exhibits better behaviour, for instance with respect to limited wordlength coefficient sensitivity. This issue is discussed in Section 5. Determination of "good" structures has long been a main issue in the digital filter field, and work still continues on this. Transformation into a well-behaved structure may also be necessary with floating-point arithmetic in critical cases. Such cases have indeed been encountered in practice.

In the case of fixed-point arithmetic the next step must be scaling (Section 6). Fixed-point numbers have a much more limited dynamic range than floating-point numbers with comparable wordlength. In order to avoid overflow but at the same time minimize quantization effects, the variables of the controller must be scaled. Scaling also influences the coefficients of the controller which have to fit into the coefficient number range available.

Finally, the target processor program can be written. This should be quite an easy task if a high-level language can be used and if the control algorithms are straightforward, but

* Received 6 September 1985; Revised 17 June 1986. The original version of this paper was presented at the 9th IFAC World Congress on A Bridge Between Control Science and Technology which was held in Budapest, Hungary during July 1984. The published proceedings of this IFAC Meeting may be ordered from Pergamon Books Limited, Headington Hill Hall, Oxford, OX3 0BW England. This paper was recommended for publication in revised form by Associate Editor B. Wittenmark under the direction of Editor K. J. Åström.

† University of Paderborn, Department of Automatic Control in Mechanical Engineering, Pohlweg 55, D-4790 Paderborn, F.R.G.

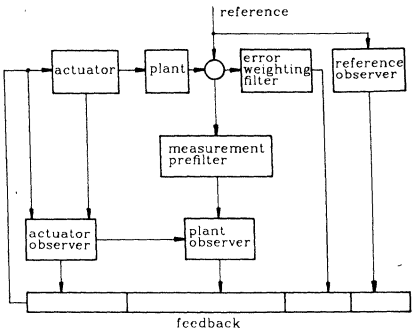


FIG. 1. Example of a structured control system.

programming can be more challenging under less convenient circumstances. Some relevant points are discussed in Section 7.

It is always advisable to carry out analysis and simulation in parallel to the steps of the implementation procedure, checking for the effects of discretization, skewed sampling, finite word-length arithmetic-effects and the like. Although appropriate analysis tools apart from simulation are valuable, the final word at least should come from a full-blown simulation of the whole control system. This task is not as trivial as it might seem, and deserves some discussion in Section 8.

2. *Control algorithms*

Before discussion of implementation issues it is useful to have a look at some of the algorithms which are possibly to be implemented and at some implications of these algorithms with respect to implementation.

It is common practice to design linear control systems and to apply them to the usually non-linear plants. Thus linear controllers are the main focus. However, linear controllers are sometimes augmented by specific non-linearities, such as non-linear friction-compensating terms, non-linear command or reference generating models and the like. This should be taken into account at least when it comes to software, both for aiding in the implementation process and for target processor program development.

A complex control system is usually composed of subsystems. For an example see Fig. 1. Such a subsystem structure may be imposed by the design process, but there are usually also technical reasons for the structuring. It is often appropriate to preserve this structure in the controller implementation, although it may be easy to merge everything together into a single system description with the set of inputs comprising all measurements from the plant as well as external inputs, and with the variables acting on the plant as outputs. Such a global controller description may facilitate handling of the implementation tasks, because CACE software then only has to deal with simple single-system descriptions. This, however, is usually the only advantage of using a global description. In terms of maintenance, modifiability and self-documentation it is certainly better to keep the subsystem structure throughout, up to and including the final target processor program. Variables with physical meaning sometimes need to be preserved and may be lost in the global description, at least after transformations, which occur in the implementation process, have been performed. A description which reflects the modular structure of the system is also adequate if the controller is composed of subsystems running at different sampling frequencies, i.e. it is a multi-rate system, and if there are non-linear subsystems a structured description is almost mandatory anyway.

2.1. *Some basic types of discrete control algorithms.* In this subsection a discussion of basic control algorithm types is given with regard to the typical individual subsystem types.

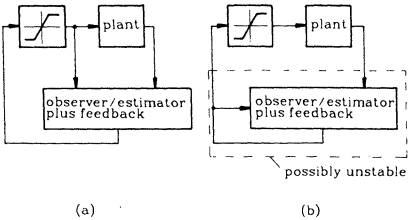


FIG. 2. Implementation with actuator saturation.

2.1.1. *Observer/estimator and state feedback.* The observed (or estimated) state vector $\hat{x}$ is computed via

$$\hat{x}_{k+1} = \Phi\hat{x}_k + \Gamma u_{p,k} + K[y_{p,k} - H\hat{x}_k] \tag{1}$$

(Franklin and Powell, 1980; Åström and Wittenmark, 1984), where u_p is the vector of control inputs to the plant and y_p may contain plant measurement variables as well as reference inputs or measured external disturbances, in the case of reference and disturbance modelling. The observed state vector is then used in

$$u_{p,k} = -L\hat{x}_k, \tag{2}$$

where L is a constant state feedback matrix, possibly including columns for feedforward of observed reference or disturbance model states. In (1) there could additionally be input terms separate from the control input term in the case of additional measurable external plant input signals. The term in brackets could be augmented by $-Du_{p,k}$, when the discrete plant state space description contains it as a direct feedthrough term. This occurs for instance when dealing with computational delay of the control processor using the approach given by Kwakernaak and Sivan (1972). The state observer/estimator may also come in another version, slightly different from (1):

$$\hat{x}_{k+1} = \Phi\hat{x}_k + \Gamma u_{p,k} + K[y_{p,k+1} - H\Phi\hat{x}_k]. \tag{3}$$

This version is called "current" estimator by Franklin and Powell (1980). Åström and Wittenmark (1984) distinguish the predictor version given by (1) from the filter version given by (3). The presence of $y_{p,k+1}$ has implications with respect to non-zero computation time (see Subsection 2.2).

Because $u_{p,k}$ which is computed via (2) also appears on the right-hand sides of (1) and (3) it is sometimes argued that (2) could just as easily be included in (1) or (3), yielding for instance

$$\hat{x}_{k+1} = (\Phi - \Gamma L)\hat{x}_k + K[y_{p,k+1} - H\Phi\hat{x}_k] \tag{4}$$

in the case of (3), along with (2) for computing the control input to the plant. This could however be dangerous when u_p as input to the plant saturates (Åström and Wittenmark, 1984). The versions (1) and (3) still work (Fig. 2a) but in the case of (4) the control system is broken up due to saturation into the plant and a system whose eigenvalues are those of $\Phi - \Gamma L - KH\Phi$, which are not even guaranteed to be stable (Fig. 2b). The control system may never again regain stable operation after saturation has occurred. Astonishingly, this simple fact has frequently been ignored in the literature. Note that this problem ties in with the loop transfer recovery issue of continuous control (Doyle and Stein, 1981) as well as with antiwindup compensation (Åström and Wittenmark, 1984). From the author's own experience designs are not unlikely to end up with an unstable system (4). In such cases at least, the control inputs to the plant should also be explicit inputs to the controller. If saturation occurs only at the DA-converter, an internal feedback of u_p under saturation in the control processor to the right-hand side of (1) or (3) may suffice, otherwise the inputs to the plant should be measured. Note that even when (4) is stable, the dynamics may be very unsatisfactory. If a continuous controller is designed and afterwards discretized, the discrete controller with feedback

as in Fig. 2b may be unstable if actuator saturation occurs, even if the continuous controller remains stable.

In (1) and (3) there is an explicit computation of the observation/estimation error (the bracketed terms). The term depending on $\hat{x}_k$ could however be omitted if $\Phi - KH$ in (1) or $\Phi - KH\Phi$ in (3) are used for Φ instead. The controller (1), (2) can then be reduced to a standard state space form

$$\hat{x}_{k+1} = (\Phi - KH)\hat{x}_k + (\Gamma, K)\begin{bmatrix} u_{p,k} \\ y_{p,k} \end{bmatrix} \tag{5}$$
$$u_{p,k} = -L\hat{x}_k$$

which is equivalent to (1), (2) with infinite arithmetic precision. With short wordlength arithmetic there may however be cases where the representation of $(\Phi - KH)$ and K in the processor causes observation/estimation errors.

The reduction of (3) and (2) to standard state-space form is prevented by the presence of $y_{p,k+1}$ in (3). An input/output equivalent standard state-space form could be found (see below) but $\hat{x}$ would not be preserved.

2.1.2. *Standard state-space systems.* If the controller design method does not yield a specific algorithmic structure such as (1) and (2), but just a discrete dynamic system with some inputs and some outputs, or in cases where the structure is not required to be preserved, the standard state-space description may be adequate:

$$x_{k+1} = Ax_k + Bu_k \tag{6}$$
$$y_k = Cx_k + Du_k.$$

Such a system may also appear as a subsystem in a complex controller. Its input thus does not necessarily coincide with the plant measurement, reference and measured disturbance vectors as in (1), and its output is not necessarily the control input vector to the plant. The usual convention of u being the input and y being the output of this system has therefore been adopted, and will be used in similar cases below. It is important to include the direct feedthrough terms in (6) because controllers frequently have such a term (think of simple P, PI, PD, PID type controllers).

If (6) describes an unstable controller/compensator with u_k which does not contain the actuator control variables (as opposed to (5)), the same problems in the case of actuator saturation arise as discussed above. The closed-loop system of course should be stable but breaking of the loop because of actuator saturation is likely to have disastrous consequences (due to possibly only "conditional stability" in Bode's terminology). Åström and Wittenmark (1984) suggest a neat way of circumventing such problems by implementing the system (instead of (6))

$$x_{k+1} = (A - MC)x_k + (B - MD)u_k + My_k, \tag{7}$$
$$y_k = Cx_k + Du_k,$$

which is equivalent to (6) as long as everything is linear. The point is that (7) is a feedback system because y_k appears as y_{k-1} in the computation of x_k. Assume now y to be the control input to the plant, and u to be the plant output. If y_k now saturates, not only the controller/plant loop is broken, but also the loop in (7) (see again Fig. 2 with (7) replacing the observer/estimator/feedback system there). Thus one is left with a system the dynamics of which are determined by $A - MC$ instead of A, and $A - MC$ may have more desirable eigenvalues because M can be chosen freely.

2.1.3. *State space system with "current" term.* Frequently the system description contains a "current" term, which means that x_k depends not only on u_{k-1} but also on the currently sampled u_k or

$$x_{k+1} = Ax_k + B_1u_{k+1} + B_0u_k \tag{8}$$
$$y_k = Cx_k + Du_k.$$

This occurs if certain methods are used to discretize an analog controller. But the simple PID controller given by

$$u_{P,k} = \alpha e_k$$
$$u_{I,k} = u_{I,k-1} + \beta e_k \tag{9}$$
$$u_{D,k} = \gamma u_{D,k-1} + \theta(e_k - e_{k-1})$$
$$u_k = u_{P,k} + u_{I,k} + u_{D,k}$$

also yields a description of the form (8), if the integral part u_I and the differential part u_D ared chosen as state variables. If it is not necessary to preserve the state variables, (8) can be translated into (6) using the substitution (Hanselmann, 1984)

$$x_k^* = x_k - B_1u_k \tag{10}$$

resulting in

$$x_{k+1}^* = Ax_k^* + (AB_1 + B_0)u_k \tag{11}$$
$$y_k = Cx_k^* + (CB_1 + D)u_k.$$

2.1.4. *Transfer functions.* Controllers or controller subsystems are often given in transfer function form if they are SISO, MISO or SIMO systems. In the case of MIMO systems the transfer matrix description is not directly appropriate for implementation purposes because of the underlying minimal realization problem. For this reason and because state-space models are more easily amenable to numerical treatment, basing CACE tools on state space descriptions might be preferred, with some important extensions as given in Subsection 5.4.

In the SISO case it is quite natural to derive an implementable difference equation directly from the z-transfer function in polynomial form:

$$\frac{Y(z)}{U(z)} = G(z) = \frac{b_0 + b_1z^{-1} + \cdots + b_mz^{-m}}{1 + a_1z^{-1} + \cdots + a_nz^{-n}} \tag{12}$$

could be implemented as

$$y_k = -a_1y_{k-1} - \cdots - a_ny_{k-n} + b_0u_k + \cdots + b_mu_{k-m}. \tag{13}$$

This is only the simplest equation, requiring more storage elements than necessary. There are various other structures also involving the polynomial coefficients of (12) more or less directly (see for example Phillips and Nagle, 1984). The problem is that such an implementation is very likely to fail with finite precision arithmetic even in low order cases, so transfer functions are usually realized in different, more appropriate forms (see Section 5).

If an observer/feedback controller is given in transfer function form in the case of a SISO plant, it has at least two inputs and one output. The two minimal inputs are the control input to the plant as measured and the plant's output variable. Additional inputs for the command or reference signals and measured disturbances may be present. So such a controller is always MISO. It may be tempting to eliminate the input of the plant's actuating variable into the controller which computed that variable. The problem associated with actuator saturation discussed above in the state-space context then also arises. If, originally, the controller is a compensator without this actuating variable feedback, and it is unstable or exhibits unsatisfactory dynamics, it is also possible to remedy this in transfer function form (Åström and Wittenmark, 1984), corresponding to the modification shown in (7).

2.1.5. *Finite impulse response filters.* Finite impulse response (FIR) filters are known from digital filter theory (Oppenheim and Schafer, 1975). They are commonly realized as non-recursive

systems, i.e. the difference equation has only input terms on the right-hand side

$$y_k = \sum_{i=0}^{n} b_i u_{k-i}, \tag{14}$$

but note that recursive realization is also possible, an example being the common recursive realization of a moving average filter. In a control system context, FIR filters may appear as subsystems for filtering purposes. They may also be used directly as controllers in certain settings (Fromme and Haverland, 1983; Widrow and Walach, 1983).

2.1.6. *Non-linearities.* All the controllers or subsystems discussed above only require simple scalar product operations involving coefficient vectors (matrix rows) and data (signal) vectors. This computation of sums of products, which requires only multiplications and additions, is the type of operation predominant in general digital signal processing, for instance in digital filtering or correlation computations. Thus processor architectures suited to the strong market of general digital signal processing are usually also well suited to controller implementation (see Section 3).

Practical control systems, however, frequently need extensions of the simple linear time-invariant systems discussed. Examples are: compensation of state-dependent non-viscous friction in mechanical systems (Henrichfreise, 1985; Walrath, 1984), non-linear command or reference generators (Broussard *et al.*, 1985), compensations of kinematic non-linearities in robot control, or adaptive mechanisms (Åström, 1983). Computations introducing operations such as decision making, divisions, table lookup, interpolation, polynomial evaluation, and computation of non-linear functions may give rise to problems with processors which are intended for linear digital filtering.

2.2. *Implications of computational delay.* In the difference equations discussed above the subscript k of input or output variables expresses time instants where sampling or output occurs respectively. Thus u_k in (6) means $u(kT)$ and y_k means $y(kT)$. Sampling and output must therefore be performed exactly simultaneously. Note that the state vector may have a meaning with respect to time instants too as in the case of (1) or (3), where $\hat{x}$ is the observed plant state but this depends on the design method which yielded the controller. It is in any case irrelevant at what time instant the state vector is computed, as long as it is computed before it has to be used for the computation of the output.

If there is no direct feedthrough from input to output and there is no current term in the state update equation (as in (1), (2), (5) and (6) if $D = 0$) then the output can be readily computed before the input is sampled and sampling and output can be simultaneous in reality. Otherwise, there is inevitable delay because—take (6) for instance—Du_k at least has to be computed and added to Cx_k, which might already have been computed because x_k does not depend on u_k. If Cx_k is precomputed, delay is minimized. The control processor program can easily be organized that way (Franklin and Powell, 1980; Hanselmann, 1982; Åström and Wittenmark, 1984). Similar arguments apply to the observer/estimator described by (3) with (2), where, in order to compute $u_{p,k}$, $y_{p,k}$ must be available and the computational effort is at least the addition of $Ky_{p,k}$ to the precomputable part of $\hat{x}_k$, and finally the computation of $u_{p,k} = -L\hat{x}_k$.

If the minimized delay is not negligible, it should be taken into account in the controller design. How this is done depends on the design method. With classical Bode diagram design for instance the delay introduces additional negative phase which could be assigned to the plant for this purpose. With direct discrete design the delay may also be assigned to the plant and design is then based on a discrete description for the plant with input delay. This description is computed either in the z-domain using modified z-transforms (Franklin and Powell, 1980; Åström and Wittenmark, 1984; Phillips and Nagle, 1984), or in state space (Franklin and Powell, 1980; Åström and Wittenmark, 1984; Wittenmark, 1985). In all these cases the delay shows up in the design of the controller.

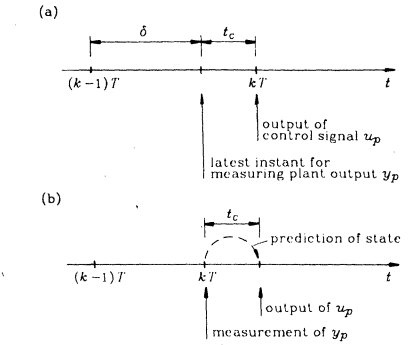


FIG. 3. Computational delay.

With observers/estimators there are more elegant possibilities which compensate for the delay. In the approach given by Kwakernaak and Sivan (1972), the time grid is fixed to the time when output of the control signal $u_{p,k}$ to the plant occurs, i.e. $u_{p,k}$ means $u_p(kT)$ (Fig. 3a). With the requirement of simultaneous sampling and output the latest measurement usable to compute $u_p(kT)$ would be $y_p(k-1)T$). If skewed (non-simultaneous) sampling were used, the latest measurement could however preferably be $y_p((k-1)T+\delta)$, where $\delta = T - t_c$, and t_c means the computational delay. Thus an observer/estimator design based on a plant description with output $y_p(kT+\delta)$ instead of $y_p(kT)$ would compensate for the computational delay.

In the approach given by Meisinger and Lange (1976), the time grid is fixed to the sampling of $y_p(kT)$ but the computation of $u_{p,k}$ is based on a predicted plant state $\hat{x}(kT+t_c)$ (Fig. 3b). The prediction is easily incorporated into the observer/estimator equations with no additional computational overhead. Similar ideas are used by Mita (1985). Meisinger and Lange's approach appears different from that of Kwakernaak and Sivan, and no reference to the latter is given. In fact, the equations describing the estimator can be shown to be equivalent. The difference is that Meisinger and Lange express the estimator gain matrix in terms of the "no delay" gain matrix assumed to be computed first.

The observation that direct feedthrough terms, or current terms which map into direct feedthrough, cannot be implemented exactly with finite speed processors, has led to the exclusion of such systems in the whole work of Moroney *et al.* (1980, 1981, 1983) and Moroney (1983). However it seems reasonable not to exclude such systems as models, firstly for cases where delay can indeed be neglected, secondly for cases where delay is assigned to the plant during design, and finally because such systems may be series-connected to others which do not have direct feedthrough, so that the input and output operations of the series connection visible from outside may well occur at the correct time instants.

2.3. *Discretization of continuous controllers.*

2.3.1. *Motivation.* Although the common design methods are available in discrete form, it is quite common to carry out continuous design first, so that discretization can be assigned to the implementation task. Discretization of continuous designs is sometimes ruled out as being inefficient with respect to necessary sampling rates, giving up some possibilities present only in discrete design (such as deadbeat behaviour), and being simply imprecise because discretized control never behaves like the continuous design. Experience shows however that it is far from uncommon for none of these arguments to be of significant relevance in practice, and there may be several reasons why the indirect way via continuous design may be the better choice.

One possible reason is that in order to exploit the exactness of discrete design there must be early decisions on sampling

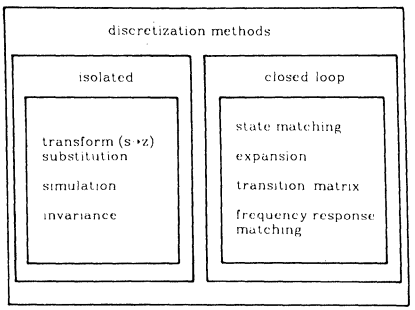


FIG. 4. Discretization methods.

frequency, and possible sampling skew (non-simultaneous sampling of all inputs) or computational delay must be known in advance. But all this depends on what shows up to be computed, what the numerical data are, and which processor and which data format will be used. If inadequate estimates have been used initially, the control system has to be redesigned.

2.3.2. *Methods.* There are so many methods available for translating a linear time-invariant controller into a discrete "equivalent" system (which in fact can never be completely equivalent), that this topic could be the subject of a survey in itself. In the following, not much more than a classification and a bibliography are given, plus a short discussion of two methods.

The discretization methods available can be classified as indicated in Fig. 4. There are two main groups. The first comprises methods which do not take into account the fact that the controller will be connected to the plant and will operate in closed loop. At most there are a few assumptions about the input signals. In the second group, discretization is carried out considering the closed-loop use of the controller.

Among the contributions to the second group are those published by Kuo (1980), Kuo *et al.*, (1973), Yackel *et al.*, (1974), Singh *et al.*, (1974) and Miller (1985). They consider the redesign of continuous system state feedback and reference feedforward matrices for the discrete case with the objective of matching the state or parts of the state of the discrete control system to those of the continuous system in closed-loop operation.

Also connected with state feedback and reference feedforward matrix redesign is another approach given by Kuo *et al.*, (1973) and Kuo and Peterson (1973) (also in Kuo, 1980) based on a Taylor expansion of those matrices about $T = 0$. These methods have been reviewed and further discussed by Kleinman and Rao (1977), who also give a so-called average gain method with the objective of approximating control signals instead of states. Closed-loop redesign is also the objective with the methods proposed by Rattan and Yeh (1978), Rattan (1981, 1982, 1984) and Shieh *et al.*, (1982), which are based on frequency response curve fitting.

The group of methods for "isolated" discretization, where only the system to be discretized is considered without taking its later connection to the other systems into account, is the largest. The most widely described methods within this group assume that the s-transfer function $G(s)$ of the continuous system is given. With the most prominent method, the so-called bilinear transform, (see for instance Oppenheim and Willsky, 1983) the recipe is: substitute s by $2(z-1)/T(z+1)$. A z-transfer function $G_z(z)$ is thus achieved. This transformation is also known as Tustin's method and relates to discrete integration, i.e. to simulation. It has the nice property of never generating unstable z-poles as long as the s-poles are stable. Another property is that the frequency response of $G(s)$ is exactly replicated in the frequency response of the discrete system (more precisely $G_z(e^{j\omega t})$, i.e. without hold device) but unfortunately with a warped frequency axis. The response of the continuous system shrinks to the range $0 \ldots \omega_s/2$, where ω_s is the angular sampling frequency.

The bilinear transform is widely in use, and tests on numerical examples (Katz, 1981; Hanselmann, 1984) indicate that this is not a bad choice. It is also quite simple to formulate this method in state space for multivariable systems. Given the continuous system

$$\dot{x} = A_c x + B_c u \qquad (15)$$
$$y = Cx + Du$$

the discrete system is of the form of (8) (Haberland and Rao, 1973; Hanselmann, 1984), with

$$A = \left[I - A_c\frac{T}{2}\right]^{-1}\left[I + A_c\frac{T}{2}\right], \qquad (16)$$
$$B_1 = B_0 = \left[I - A_c\frac{T}{2}\right]^{-1}\frac{T}{2}B_c,$$

where I means identity matrix. Note that A is a first-order Padé approximation for the transition matrix $\exp(A_cT)$.

The formulation in state space directly translates into a simple computer program. The calculations based on the transfer functions can however also be mechanized (Ahmed and Natarjan, 1983; Bose, 1983; Pei, 1985). Bilinear transformation is not the only method from the transform or substitution class. More can be found for instance in Katz (1981) and Franklin and Powell (1980) along with some comparisons by examples, and in Rosko (1972) and Smith (1977). A "small T" root and frequency response error (continuous/discrete) analysis for the bilinear transformation is given by Howe (1982).

Since determination of a discrete system equivalent to a continuous one is related to simulation, methods from that field may also be of interest here. In fact, the bilinear transformation already corresponds to a simulation of an equivalent continuous state space system via implicit trapezoidal integration. Hanselmann (1984) also derived discrete systems from Heun's simulation method and one of the Runge-Kutta type and compared them to other methods. Experience showed no general advantage over for instance bilinear transformation and over the ramp-invariance method described below. One method which seems very interesting and also has some connection with simulation has recently been published by Forsythe (1983, 1985). It is given for SISO systems and is based on expressing the samples of the input and output variables via Taylor series expansion of the continuous functions. Results are shown which are clearly superior to those of the bilinear transform in a large frequency range, although at the expense of increased gain in the high frequency region. This could be dangerous in a closed-loop control system.

The last class of methods is based on assumptions on test input signals applied both to the continuous system and to the discrete one to be determined. The objective is to achieve agreement of both outputs at sampling instants. Assumption of a step input leads to a step-invariant and to a ramp input to a ramp-invariant discretization, occasionally called "zero order" and "first order hold equivalence" methods, respectively. The step-invariant discretization is just what has to be performed in order to describe a continuous plant driven by a zero-order hold (ZOH). A table of step-invariant transfer functions can be found in Neuman and Baradello (1979). The ramp-invariant discretization is also easy to achieve, either via transfer function calculation, i.e.

$$G_z(z) = \frac{Y^c(z)}{R(z)} = \frac{(z-1)^2}{Tz}Z\left\{G(s)\frac{1}{s^2}\right\}, \qquad (17)$$

or in state space. The assumption of a ramp input between

sampling instants leads to the state-space equation solution (continuous system (15) assumed)

$$x_{k+1} = \exp(A_c T)x_k + \int_0^T \exp[A(T-\tau)] B_c \left[u_k + \frac{u_{k+1} - u_k}{T}\tau \right] d\tau \quad (18)$$
$$= Ax_k + Hu_k + H_s(u_{k+1} - u_k)/T$$
$$= Ax_k + B_1 u_{k+1} + B_0 u_k.$$

The transition matrix and the input matrices H and H_s can be computed simultaneously via a single transition matrix calculation (Hanselmann, 1984), but also by other means. The power series expression of $\exp(A_c T)$, for instance, which is sometimes used as a basis for computation of A and H, also leads to algorithms for computing H_s. Schittke and Dettinger (1975) used this (unfortunately there is an error in the series given in their equation (15)). The approach given by Källström (1973) for computation of A and H based on one single series calculation can also be extended*. The series to be summed is

$$\psi = \sum_{i=0}^{\infty} (A_c T)^i/(i+2)! \quad (19)$$

then

$$A = I + A_c T + T^2 A_c^2 \psi \quad (20)$$

$$\mathrm{H} = (\mathrm{T\,I} + \mathrm{A_c T}^2 \psi)\mathrm{B_c} \quad (21)$$

$$H_s = T^2 \psi B_c. \quad (22)$$

Definition and derivation of the ramp-invariance method in state space has already been found in a paper by Haberland and Rao (1973). The expressions given there for B_1 and B_0 can be derived from (20)–(22). A small T study concerning scalar transfer function zeros generated via impulse-, step-, or ramp-invariance has been carried out by Bondarko (1984), whose step-invariance results relate to those of Åström *et al.*, (1984).

The author's experiences with the ramp-invariance method are very good, particularly in critical cases where there are continuous system eigenfrequencies near $\omega_s/2$. The step-invariant results, however, showed bad frequency responses compared to those of the continuous systems in practically every application. Sampling frequency could have been lowered by a factor of five using ramp invariance instead of step invariance with a high-order controller for a hydraulic system (Hanselmann, 1984). So the unsatisfactory experiences with discretized continuous designs, compared to discrete designs, which are sometimes reported may well be due to inappropriate discretization.

2.3.3. *Influence of zero-order hold.* A general problem with discretized controllers is that the ZOH at the outputs introduces considerable phase lag. Thus discretized controller frequency responses are likely either to show more negative phase compared to the continuous controller, or to show increased gain in the higher frequency region, which stems from the attempt to lift phase. Stability and damping problems could occur. In applications carried out by the author, sampling frequency had to be from a factor of 3 to 10 higher than crossover frequency, in order to preserve reasonably the behaviour of the continuous system. From aliasing and roughness of control signal considerations, which often also dictate sampling frequencies in that range, such a ratio does not seem to be excessive.

With some of the discretization methods the phase lag of the ZOH can be taken into account directly. This applies naturally to the closed-loop discretization method class. The "isolated discretization" method of Forsythe (1985) is also able to do this, and furthermore to compensate somewhat for possible computational delay. The price of delay compensation however is again an increased high frequency gain. The same applies to what might be called "post-filters", which are digital filters connected between the controller difference equation output and the ZOH. Such filters have been described by Yekutiel (1980) and Beliczynski and Kozinski (1984). They lift phase but must be handled with care unless a rapid gain rolloff beyond the crossover frequency is guaranteed.

3. *Implementation hardware*

The author is well aware of the fact that any discussion of hardware is doomed to be obsolete within a very short time. So this survey gives only a snapshot of current implementation hardware, but there are some points which might be relevant for a few years.

3.1. *Spectrum of current hardware.* The range of possible hardware for implementation of algorithms as discussed in Section 2 is very broad. A rough overview is given in Table 1.

3.1.1. *Special machines for rapid experimenting.* At the upper end in terms of cost as well as computational power there are high-speed computers specifically designed for real time data acquisition and computation. The AD10 from Applied Dynamics International, Ann Arbor, Michigan, is capable of 30 million arithmetic operations^{-1} and 10 kHz data acquisition on 32 A/D channels simultaneously (Powers, 1985; Kerckhoffs *et al.* 1985), but costs are in the US $200,000 range. Advanced versions recently available are also capable of floating-point computation (Fadden, 1984), but at even greater cost. Such systems are attractive for experimental work in the early stages of a control system design and implementation project, in order to obtain feedback from real experiments as early as possible, and as easily as possible, with the convenience of floating-point arithmetic, flexible programming, and plenty of speed. Common minicomputers backed up with array processors may also be used with similar power but also at high cost (Jacklin *et al.* 1985). Without array processors, the speed of minicomputers is usually rather modest. A less costly system which is marketed specifically for experimental linear control system evaluation is the PC 1000 from Systolic Systems Inc., San Jose, California, starting at US $25,000. It is rated at 200 ns multiply as well as addition time with 32-bit floating-point numbers, and 2 kHz maximum sampling rate. Controllers of type (6) with up to 32 states and 16 inputs and outputs can be accommodated under the control of a personal host computer with download facility.

3.1.2. *Fast floating-point chips.* Roughly the same computation speed as described above will be possible with systems based on so-called word-slice chip sets from Advanced Micro Devices (Flaherty, 1985; Quong and Perlman, 1984) and Analog Devices (Windsor, 1985; Taetow, 1984). They evolved from the more traditional bit-slice concepts and now comprize all necessary building blocks to develop microprogrammed high-speed signal processing systems with just a few chips, among which are special purpose arithmetic chips, i.e. separate chips solely for accumulating or multiplying floating-point numbers.

Floating-point computation in the same speed range as with word-slice devices is possible using arithmetic chips from Weitek Corporation. Separate 32-bit floating-point adder and multiplier chips along with 32-word register file devices form a powerful numerical processor. Control of the devices must be derived from microcode memory and control logic. About 2 MFLOPs (mega floating-point operations per second) can be achieved in low latency flowthrough mode, which means that the result of a single arithmetic operation is available as soon as possible. If pipelining can be used 10 MFLOPs are achievable, but results are then not immediately usable in subsequent operations.

Another two-chip set for floating-point arithmetic is available from TRW (Eldon and Winter, 1983), which is, however, restricted to a 16-bit mantissa 6-bit exponent format. Note that division is not as directly performed as accumulation (addition and subtraction) or multiplication with these chips, nor is it with the above-mentioned word-slice devices. Division must be

*Thanks to Prof. K.-J. Åström who brought this to my attention.

TABLE 1. IMPLEMENTATION HARDWARE

	Experimental use in the laboratory			Dedicated	Dedicated
	High cost	Medium cost	Low cost	low volume	high volume
High speed	AD-10 minicomp. and array processors Systolic Systems PC1000	word-slice floating-point chips	VLSI signal processors		
Medium speed	minicomputer	microprocessor with numerical coprocessors			custom VLSI
		Microcontrollers			

performed using table-lookup methods to yield rough estimates which are then improved via additional operations, or it is performed totally iteratively. This means that division and any other function computation involving division is performed much more slowly than the elementary scalar product operation acc: = acc + coefficient* variable. Within the Weitek register file there is an integrated lookup table for computing $1/x$ and $\sqrt{x}$.

3.1.3. *Microprocessors.* Easy implementation and testing of controllers at much lower cost and effort is of course possible using standard personal computers or microcomputer board level systems, equipped with process interfaces, and speeded up by numerical coprocessors, such as the Intel 80286/80287 or the National Semiconductor NS 32016/32081 combinations. Such systems are easy to program in high-level languages and deliver medium speed (see Subsection 3.3), sufficient for implementing even complex process control in many cases, but frequently not fast enough for control of fast systems such as mechanical ones. Attaching fast hardware multipliers to general microprocessors may also seem to be an alternative. They are available in abundance from many companies, up to (24 × 24)-bit fixed-point format at 200 ns multiply speed or (16 × 16)-bit in 35 ns. But data transfer from and to such a chip via a microprocessor is much too slow, so the multiplier would be idle most of the time. Avoiding this would necessitate not only using a hardware multiplier, but surrounding it with a lot of hardware to achieve more independent operation on local data memory, under local sequencing control.

3.1.4. *Microcontrollers.* The term microcontroller is used commonly for single-chip microprocessors which are designed to be used as dedicated processors. But control is meant here in a much broader sense than considered in this paper, including sequencing control, pulse-width or pulse-frequency modulation control, and so on. Microcontrollers stand somewhere between traditional single-chip microcomputers and general purpose microprocessors. Three powerful 16-bit devices shall be named here, the Motorola MK 68200, the Nippon Electric NEC μPD 78312, and the Intel 8096. Typically, the arithmetic computation speed is not much higher than with general 16/32-bit microprocessors for fixed-point arithmetic. But there are features like on-chip AD-converters or timers and modulators which make such processors attractive for developing products. It is interesting to note that the 8096 evolved from a chip originally designed according to requirements specifications made by Ford Motor Company for control applications in an automobile (Powers, 1985; Breitzman, 1985; Simmers and Arnett, 1985).

3.1.5. *Signal processors.* Very attractive computation speed is achieved with a number of VLSI signal processors at microprocessor level cost (Hanselmann and Loges, 1983, 1984; Hanselmann 1986). Present devices of that kind that seem to be useful for control implementation and are available to the public are the Nippon Electric NEC 7720 (Nishitani *et al.* 1981), the Texas Instruments TMS 32010 (McDonough *et al.* 1982), the Fujitsu MB 8784 (Gambe *et al.* 1983), the STC DSP 128 (Pickvance, 1985), and the Texas Instruments TMS 32020 (Magar *et al.*, 1985; Essig *et al.*, 1986). Some descriptions can also be found in Quarmby (1984), Marrin (1985), and of some recently announced processors in Marrin (1986).

The signal processors mentioned are off-the-shelf products. The class of only mask-programmable signal processors has been excluded. They are not of course useful for the average control implementation task. There is great activity in the development of signal processors. Several companies have announced such devices.

For medium to high volume applications, custom chips may be the choice. Custom design is advancing in supplying quite complex building blocks such as multipliers, arithmetic units and memory (Cole, 1985). Furthermore, there is considerable effort towards fully automated chip design (Cappello, 1984). Pope *et al.* (1984) and Rabaey *et al.* (1985) for instance describe a silicon compiler which starts with some high-level descriptions of what the signal processor chip is expected to perform. The software then chooses optimal parameters of a parameterized architecture and finally outputs a complete chip layout. Combining building blocks into a freely designed architecture is another approach (Glesner *et al.*, 1986).

VLSI signal processors make implementation of non-trivial controllers at high sampling rate feasible at reasonable cost, and particularly the TMS 32010 has already been used in many control applications, as described for example by Slivinski and Borninski (1985), Kanade and Schmitz (1985), Hanselmann (1986). The power of signal processors is due to their architecture, not to exotic silicon process technology. It may therefore be interesting to have some general discussion of architectural features in the next subsection.

3.2. *Architectural issues.* When a chip or chip set is to be selected for controller implementation, there are many criteria which might be relevant. Their priority depends mostly on the type of application intended. Building a tool for flexible lab experimentation sets priorities other than looking for a medium volume dedicated industrial instrumentation system.

3.2.1. *General considerations.* How general purpose 16/32-bit microprocessors, a typical microcontroller, and the current VLSI signal processors meet some of the relevant criteria is shown in Table 2 (for a survey of microprocessors see Gupta and Toong, 1983, 1984). The 8096 has been chosen as representative of a trend in microcontrollers. Note the amount of input/output support right up to multi-channel on-chip AD-conversion. Microcontrollers are particularly well suited to industrial applications, where control of the type discussed in Section 2 is frequently only one task among many others, including sequencing, complex timing, interrupt processing and communication. Computing speed is however not as high as with signal processors. Apart from the special input/output features, the architecture of the 8096 is much like that of traditional general microprocessors, with the exception of an increased number of on-chip registers forming a so-called register file. There are 232 bytes free to the user to be referenced as byte, word or double word registers. This is an important feature, because such an on-chip register file can be accessed more quickly than external memory. It is large enough to carry out large portions of the task locally and also helps speed up context switching during processing interrupts.

When so many functions as a microcontroller has are integrated on a single chip, something must be sacrificed in comparison with general purpose 16/32-bit microprocessors. One of the features of the latter, missing in a microcontroller,

TABLE 2. PROCESSOR COMPARISON

	Microprocesors	Microcontroller 8096	Signal processors
Floating point	slow, medium with coprocessor ($\approx$5–15 μs mult. or add)	slow	impossible or slow
Speed 16 × 16 fixed-point mult.	5-12 μs	7 μs	0.1–0.3 μs
ALU wordsize	16–32	16	16–35
Program address space	>1 MB	64 kB	1.5–128 kB
Data address space	same	same	128–588 × 16 for onchip R A M (external extension possible with newest proc.)
On-chip AD/DA	—	4–8 AD channels, 10 bit	—
Special I/O	—	pulse width mod., timer, counter, watchdog, ports	—
On-chip ROM	—	8 kB	all but one
Memory speed required	medium	medium	25–150 ns
Interrupts	flexible via interrupt controllers	7 sources internal, 1 external	0–3
Multiprocessor capability	ext. logic	—	newest proc.
Program language support	best	some	asm only in most cases; high level language support for one processor
Chip count	high	low	medium to high

is the large address space, which is in fact not necessary for control implementation. A small address space saves much room on the chip, because the address space is reflected in all registers and logic related to effective address computation, as well as in the bus interface. Provisions for memory management can also be dispensed with.

Other savings stem from reduced instruction decoding circuitry due to a simpler instruction set, excluding advanced high-level language-like instructions as for instance incorporated in the VAX-like instruction set of the NS 32016 general microprocessor. The reduction in instruction decoding and processing logic due to a simpler instruction set is also a general line of development with advanced supermicroprocessors for general purposes. These processors are said to be of the RISC type (RISC means reduced instruction set computer) (Wallich, 1985). They are characterized by an instruction set which includes only the most used instructions and by executing one instruction every machine cycle. Operations are performed on operands in large register files, not on memory, which is accessed only by load and store operations. Among the digital VLSI signal processors there are also some which are RISC-like, particularly the TMS 32010 and the DSP 128 signal processors.

3.2.2. *Specifics of signal processors.* Whereas the 8096 microcontroller discussed above appears, from outside the chip, to be of the traditional "von Neumann" computer type, internally instructions go their own way separately from the data. It is a well-known bottleneck of traditional processors (von Neumann type) that instructions and data travel on the same bus. This architecture must be abandoned if data transfer between registers, data memory, and arithmetic units is to be fast for maximum throughput. One step away is the so-called 'Harvard' architecture. In this architecture the instruction bus is separated from the data bus so that instruction fetch and data transfer do not interfere with each other. Some signal processors exhibit even more data paths. For illustration, a sketch of the core architecture of a hypothetical but typical signal processor is given in Fig. 5, showing the data manipulation part (instruction bus and control unit are separate). There are two 16-bit data buses, each connected to a block of data memory and to the hardware multiplier inputs. Factors can thus be routed to the multiplier without bus conflicts. The arithmetic/logic unit (ALU) gets operands either from the accumulator, from memory, or from the multiplier, converted to 32-bit where necessary. Typical components are the shifters, particularly the barrel shifter. It allows the shifting of an operand by multiple bits within a single data transport operation.

Besides the multiple bus and data path structure, the most significant difference between signal processors and general microprocessors or microcontrollers is the integrated parallel hardware multiplier. This multiplier produces a (16 × 16)-bit product in every machine cycle (see discussion of speed in Subsection 3.3), which is afterwards directly fed through the ALU into the accumulator in the next cycle in order to perform the basic operation

$$\text{acc} := \text{acc} + \text{coeff} * \text{variable}.$$

With a hardware multiplier the multiplications no longer dominate execution times as usual. They are as fast as additions or logic operations. It is however important not only to have a hardware multiplier, but also to have a powerful data path structure. Otherwise the precious arithmetic units cannot be kept busy all the time. Note that these components consume

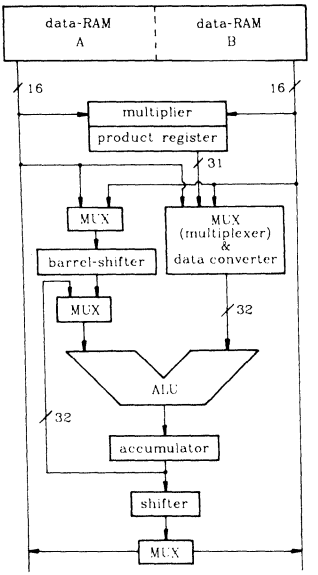


FIG. 5. Typical signal processor core.

large parts of the chip area (see photographs in Cushman, 1982). With the TMS 32010 for instance, a scalar product computation $a = c^T x$ proceeds as follows:

```
.
.
.
LTA x(i)
MPY c(i)
LTA x(i + 1)
MPY c(i + 1)
.
.
.
```

where the LTA instruction loads one operand in one of the multiplier's input registers, but at the same time performs accumulation of the previously computed product. The MPY instruction loads the second operand into the second multiplier input register and in the same cycle the multiplication is performed, the result of which is accumulated during the next LTA. The operands travel to the multiplier over a single data bus, so loading takes two cycles. With processors having split memory (as in Fig. 5) the coefficients (of c^T in the example) and variables representing signals (x in the example) could be stored separately and loaded simultaneously, so single-cycle operation is possible. This can frequently be found in signal processor architectures.

The integrated hardware multiplier, along with an appropriate data path structure connecting the arithmetic units (ALU, multiplier, shifters) and memory are the keys to the high speed of VLSI signal processors. There are however quite a number of miscellaneous features which also contribute to speed, mostly by devoting hardware to tasks traditionally performed by software. The VLSI signal processors are currently acknowledged as being attractive candidates for control implementation, not only in the sense of Section 2. They are also well-suited to performing arithmetic subtasks as a slave to a general microprocessor host within a control system (Schumacher and Leonhard, 1983; Rojek and Wetzel, 1984; Leonhard, 1986). They cannot however directly compete with microcontrollers in terms of functionality.

3.2.3. *Arithmetic and data formats.* A last important point of discussion is the arithmetic data format supported by the different processors. This point can be as crucial as speed. In many cases floating-point arithmetic is desired, be it because the dynamic range required is indeed large, or because the implementer does not want to deal with the problems of fixed-point arithmetic. With general microprocessors as well as microcontrollers, floating-point arithmetic in a common format (IEEE standard 754, 32-bit) is easy to achieve through subroutine libraries or floating-point coprocessors, providing considerable speed.

With present VLSI signal processors, however, floating-point arithmetic is not easily achievable. There has been an effort to perform floating-point arithmetic on a TMS 32010 (Blasco, 1983) and on the TMS 32020 (Crowell, 1985), but speed results are rather disappointing compared to general microprocessor/co-processor combinations. No effort to implement floating-point arithmetic on the other fixed-point signal processors has been reported. There is one VLSI signal processor, the Hitachi HD 61810 (Hagiwara *et al.*, 1983), which is specifically designed for a particular kind of floating-point arithmetic, but it is only available with mask programmed ROM, and floating-point accuracy is limited by a (12 × 12)-bit multiplier. There are some known developments of signal processors with full 32-bit floating-point hardware on the chip (from Bell Labs, Nippon Electric and Texas Instruments), but the first is not available to the public, the second has just been announced, and the third still seems to be in the design stage. Thus with present VLSI signal processors one must deal with fixed-point arithmetic and all the associated problems.

Within this group of fixed-point processors there are still differences in the useful data formats, which stem from architecture design decisions. The main differences are in the processing of products from the multiplier, and in the format of the accumulator. With the exception of MB 8764 all processors provide at least 32 bits for accumulation of (16 × 16)-bit products, so that full precision is preserved until storage of a final scalar product result (see Section 4). At this point rounding or truncation is usually performed to obtain the most significant 16 bits of the result, although more precision is possible with most processors, at the cost of more complicated code and slower execution.

3.2.4. *New architectures.* In addition to the more conventional architectures just discussed, there are other developments which are already having an impact on signal processing and also beginning to have one on control. The transputer concept (Taylor, 1984), systolic architectures (Kung, 1984; Jover and Kailath, 1986), and data flow processor concepts (Chong, 1984; Hartimo *et al.*, 1986) should be mentioned here.

3.3. *Speed.* Although there are usually many aspects of processor selection other than speed, it is nevertheless often the most pressing factor in controller implementation. This is typical of the field of controlling mechanical devices via fast electromechanical or servohydraulic actuators. Eigenfrequencies from 100 Hz up to 10 kHz are not uncommon, and higher order controllers are often necessary to cope with structural resonance effects (see for example Slivinski and Borninski, 1985; Kanade and Schmitz, 1985; Hanselmann, 1986).

A speed comparison for general microprocessors for the task of digital filter implementation, which is many respects similar to controller implementation, has been given by Nagle and Nelson (1981), also published in Phillips and Nagle (1984) (note that some of the programs originally published have been corrected in the latter publication). Speed comparisons on instruction and routine level using general data processing benchmarks have been published by Gupta and Toong (1983) and Toong and Gupta (1982).

TABLE 3. SAMPLING FREQUENCIES WITH AN EXAMPLE CONTROLLER USING FIXED-POINT ARITHMETIC

Microprocessor	Clock (MHz)	f_s(kHz)
8086	8	<2
Z8000	5	<2
68000	10	<4
32016	10	<5
TMS 32010 signal processor		31

If floating-point arithmetic is required the current signal processors can be excluded from the comparison. Their fixed-point speed is about the same as the floating-point speed of the fast word-slice and floating-point chips from Section 3.1.2. The fastest chip set (using the AMD 29325) achieves computation of a length n scalar product in about $n * 200$ ns, with full 32-bit IEEE standard data format. This should be compared with the often "thought to be fast" microprocessor/coprocessor combinations such as the Intel 80286/80287 or the faster National Semiconductor 32016/32081. The latter require about $n * 20\,\mu$s for the same thing (at 10 MHz clock, slave processor protocol execution included, from measurements by the author).

Roughly the same speed as with microprocessor/coprocessor combinations can be achieved with the microprocessors alone if floating-point arithmetic is dispensed with. Compared to adds and subtracts or miscellaneous operations, the fixed-point multiplications are the most time-consuming ones. A typical execution time is $6\,\mu$s for a 10 MHz 32016 processor (operands in memory).

With VLSI signal processors the execution times of add/subtract as well as multiply operations are in the range 100–300 ns. Multiplication is no longer the most time-consuming operation. Remember that in the example of a scalar product computation with the TMS 32010 in Subsection 3.2 only two instructions provide computation of a product (16 × 16 bit) and its accumulation (32 bit). This takes just 400 ns.

In Table 3 a comparison is made between some microprocessors and a signal processor (Hanselmann and Loges, 1984; Hanselmann, 1986). The comparison is based on the implementation of a 9th order controller with only one input and one output. This controller arose in an industrial application with a very fast electromechanical positioning system. Since with general microprocessors the multiply operation mainly determines the execution time, an upper bound for the achievable sampling rate can be given based only on the total number of multiplications. This upper bound is given in the rightmost column. The controller had 33 non-zero and non-one coefficients, i.e. 33 (16 × 16)-bit multiply operations had to be performed per sampling interval. Since there are also additions and data transfer operations to be performed the sampling frequency actually achievable would be somewhat lower. A comparison of the estimate with actual experimental results was carried out on a filter (from Phillips and Nagle, 1984), and on the controller on which Table 3 is based. The target was a 68000 system running at 10 MHz, programmed in assembly language. Actual sampling rates turned out to be about 50% of the upper bound estimate in the filter case, where subroutines and loops were used, and about 70% in the controller case with fast subroutine- and loop-less code.

The same controller was also implemented on a TMS 32010 signal processor and ran at 31 kHz sampling frequency. Thus the signal processor is an order of magnitude faster. Roughly the same applies to the other signal processors mentioned, and this compares quite well with the 17 kHz achieved in what seems to be a similar situation using an AD10 machine (Howe, 1982).

3.4. *Processors with special architecture related to control.* The average control engineer still only has access to off-the-shelf processors such as general purpose microprocessors or signal processors. Custom processor design, however, is already beginning to play a part. In the general digital signal processing field there is much going on in that direction (Cappello, 1984). Since there are many relationships between general signal processing and control these efforts also have an impact on this field. Proposals for processor architectures directly related to control were made years ago by Tabak and Lipovsky (1980) and recently by Jaswa *et al.* (1985). Proposals for processors using non-standard arithmetic such as that given by Lang (1984) or Tan and McInnis (1982) should also be mentioned here; the arithmetic issue will however be discussed in Section 4.

3.5. *Interfacing to the plant.* It is not the intention to go into the details of analog and digital interfacing techniques here, but there are some points which seem to be worth making.

A typical analog-to-digital interface consists of an analog prefilter for each channel, a multiplexer if an analog-to-digital converter (ADC) is to be shared among several inputs, a sample-hold circuit, and the ADC. The purpose of prefilters is to avoid aliasing due to spectral components of the input signal above $f_s/2$, where f_s is the sampling frequency. Clearly such filters have to be chosen carefully in control applications, because generally the sharper the cutoff in the magnitude frequency response, the lower the phase introduced into the loop. For instance even a simple second order low pass (damping $1/\sqrt{2}$) designed to give a mere 20 dB attenuation at $f_s/2$ still introduces about 25 degrees negative phase at $0.05 f_s$, where the crossover frequency might be. Most often it will be necessary to include prefilter dynamics in the control design (Åström and Wittenmark, 1984).

Measurement noise effects under variation of prefilter bandwidth and sampling rate have been studied by Peled and Powell (1978). The results are also given in Franklin and Powell (1980). It is shown that good noise attenuation at quite low sampling rates can be achieved with prefilter bandwidth only about twice the control bandwidth, provided that appropriate digital lead compensation is introduced to counteract the prefilter lag.

The purpose of a sample-hold (SH) circuit in front of an ADC is to provide a constant input signal to the normal successive approximation ADC during conversion (Davies, 1985; Jaeger, 1982). SH circuits in front of the multiplexer are necessary if simultaneous sampling of several channels is desired, sharing only a single ADC. It is always taken for granted that a successive approximation ADC must be preceded by a SH. Otherwise changes of the input signal during conversion may be reflected in the binary conversion result. This is considered to be erroneous since the value at the definite sampling time, i.e. at start-of-conversion time, is expected to be converted. To prevent such a change of the input signal, its amplitude and/or frequency must be very low or a SH must be inserted (Jaeger, 1982; Shoreys, 1982). In the control application it may however sometimes be reasonable to omit the SH, because in that case changes in the input signal occurring during conversion influence the conversion result so that it can be nearer to the input signal value at the end of the conversion than in the SH case. Thus reduced effective conversion delay can be expected. Experiments by the author showed delay reduction of a factor of up to 4. This factor is even higher if the acquisition time of the SH is significant. The effective delay reduction is however dependent on signal amplitude and spectrum, so some dynamic nonlinearity is introduced.

At the analog outputs of a controller there are commonly digital-to-analog convertors (DAC). Standard components are fast enough for conversion time to be neglected. But spectrum shaping may be of interest to smooth the staircase output signal or correspondingly to remove the extra high frequency components introduced by the zero-order hold device. Analog reconstruction or low pass filters for that purpose are often used in general digital signal processing or signal generation. With control systems such output filters are introduced more reluctantly, because of effects on system dynamics similar to those of prefilters. Reducing actuator wear as well as preventing excitation of high frequency structural modes in mechanical systems might however require output filtering.

The last point to be discussed is the sequencing of inputs and outputs. In the usual "near-theory" case there will be simultaneous sampling and simultaneous output, possibly with delay between the two, but non-simultaneous sampling may be dictated by processor hardware, or may be deliberately introduced to include the latest measurements in the computation. For example, numerical processing of channel 1 input may take considerable time before channel 2 is involved. It might then be reasonable to delay sampling of the latter. The same applies if ADCs with quite different speeds are used.

Non-simultaneous output may also occur for similar reasons. Although such cases do not fit well to common control design software, they do exist, and should be considered, at least in simulation.

4. *Arithmetics and their implications*

Basically, there are several choices of arithmetic which could be used to implement a controller. The most well known are floating- and fixed-point binary arithmetic and they are the ones supported by standard processors. Fixed-point arithmetic is mainly used because of the high speed which can be achieved with relatively simple arithmetic units. In speed, space, or cost-critical applications fixed-point arithmetic will most likely be chosen. In the following some main issues concerning fixed-point arithmetic will be reviewed. Floating-point arithmetic will be discussed only briefly as well as some other possible candidates. Unfortunately, the chapters on arithmetic found in most texts on digital filters or digital control are quite rudimentary. There are, however, some texts on computer arithmetic covering the material needed to understand the principles and problems of the mechanics of binary (and other types of) arithmetic, such as Flores (1963), Hwang (1979), Waser and Flynn (1982). Classical original papers on arithmetic as reprinted in Swartzlander (1980) are also quite instructive.

4.1. *Fixed-point arithmetic*

4.1.1. *Basics.* The usual fixed-point data formats in digital signal processing make use of two's complement representation. Here, the decimal value of a number is

$$r = 2^{-B}\left[-b_{l-1}2^{l-1} + \sum_{j=0}^{l-2} b_j 2^j\right], \quad b_i \in 0,1, \tag{23}$$

where the $b_j, j = 0, \ldots, l-2$ represent the binary digits, i.e. bits, b_{l-1} carries the sign information, l is the total wordlength, and B determines the location of the binary point. Two special cases are $B = 0$, which means r is an integer, and $B = l-1$, which means r is a fractional number. With floating-point number representation B could be different for each number whereas with fixed-point numbers B is fixed throughout.

The reason why the representation (23) is called two's complement becomes obvious in the important case of fractions, where $B = l-1$ and thus

$$r = -b_{l-1} + \sum_{j=0}^{l-2} b_j 2^{j-(l-1)}. \tag{24}$$

If $r < 0$ but the binary representation bit pattern of $|r|$ is known, then the binary representation bit pattern of the positive two's complement number $2 - |r|$ yields the b_i in (24) exactly, because $2 - |r| - 2 = -|r| = r$ and subtracting 2 has the same effect as changing the weight of the b_{l-1} bit from $+1$ to -1, as is done in (24). No bit is altered from the bit pattern representing $2 - |r|$, only the interpretation as decimal value is affected by changing the weight of b_{l-1}. A 4-bit fractional two's complement representation for example is

0.875	0.111
⋮	
0.125	0.001
0	0.000
−0.125	1.111
⋮	
−1	1.000

and for instance the bit pattern for -0.125 is that of the binary representation of $2 - 0.125 = 1.875$. The example also illustrates that the number range is unsymmetrical, i.e.

$$-1.0 \leqslant r \leqslant 1.0 - 2^{-(l-1)} \tag{25}$$

in the fraction case. An implication of this is that the product $-1.0 * -1.0 = +1.0$ (all decimal) can never be represented. In fact, processors usually yield the wrong result -1.0 in this case. In consideration of the dynamic range of data in connection with scaling (Section 6) the upper limit is simply approximated by 1.0 to simplify discussion.

The main advantage of two's complement representation compared to other candidates lies in the simplicity of hardware for adding or subtracting (Shaw, 1950). No distinctions need to be made as to what the signs and magnitudes of operands are and a single adder unit plus a simple complementer circuit is sufficient to perform addition and subtraction.

Another advantage is that a sequence of two's complement additions or subtractions, as encountered in the scalar product computation, always produces the correct result as long as this is in the number range. Intermediate overflows of partial sums thus do not matter and can be ignored. This nice property however is only useful if the result is indeed known to be in the number range. Where it is not, it is even impossible to detect this and to supply a maximum or minimum value. Sometimes arithmetic units have an extended accumulator to accomodate overflowing bits up to the moment where the result is going to be stored away. Then a check can be made on whether the result is valid or should be replaced by max or min values.

Although multiplication of two's complement numbers may seem complicated at first due to the negative weight of b_{l-1}, it can be carried out quite easily, for instance by performing appropriate sign extensions on negative number representations, or using Booth's algorithm or modifications of it (Booth, 1951; MacSorley, 1961; Rubinfield, 1975; Cappellini). These algorithms work for any combination of signs of the factors and at the same time speed is gained as compared to the simple "shift and add" technique. They are incorporated for instance within the hardware multipliers of signal processors.

The basic idea behind such algorithms is based on the observation that a string of ones in a binary number could be replaced by only two non-zero digits, if negative weights (denoted by bar) are allowed, for example

$$0111011110 \triangleq 01111000\bar{1}0 \triangleq 1000\bar{1}000\bar{1}0. \tag{26}$$

Thus if the leftmost binary pattern represents a factor in a multiplication, the right-hand side of (26) shows that the product can be computed with one addition and two subtractions, along with appropriate shifts. This compares to seven additions with shifts necessary originally. See for instance Peled and Liu (1976) for a short but instructive discussion. Translation of a binary number into this so-called canonical signed digit code (CSD) can easily be mechanized in an iterative process.

Multiplication based on CSD code has also found a number of applications in signal processors, which execute the shifts and adds or subs under program control, saving a hardware multiplier. A well-known chip of this kind was the now outdated Intel 2920 signal processor, (Hanselmann, 1982) but it is not the only one. In the design of chip area-effective custom signal-processing devices this kind of multiplication aroused (for instance Schmidt, 1978) and still arouses interest (Gaszi and Güllüoglu, 1983; Steinlechner *et al.*, 1983; Pope *et al.*, 1984).

The product of two l-bit numbers is a $(2l\text{-}1)$-bit number. This is because there is a sign bit in each factor, but the product needs only one. It is important to understand that a multiplier device is not usually concerned with the binary point location. It can multiply integers as well as fractions because the interpretation of the bit pattern of a number representation only takes place when the $(2l\text{-}1)$-bit product is stored away, see Fig. 6 for a 16-bit example. Here a 32-bit product register or accumulator is assumed, and the product bit pattern is right justified. So if the factor bit patterns were meant to represent integers, the result (assuming it should be 16 bits long) would be found in the lower (right) half of the register. If the factors were however meant to represent fractional numbers the result would be found in bits 15 through 30. Note that with some processors the output of the multiplier is aligned differently by hardware: to be specific, a fraction result could be left justified so that the store operation does not overlap into the lower half of the $2l$-bit accumulator. Note also that rounding could be performed before storing the truncated 16-bit result away by adding, prior to storing, a 1

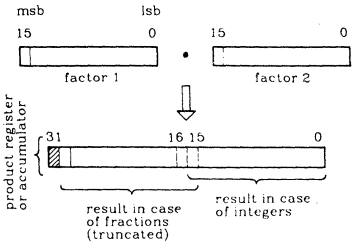


FIG. 6. Fixed-point arithmetic product.

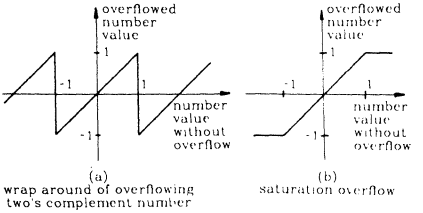


FIG. 7. Arithmetic overflow (with fractional numbers).

into the most significant of the bits which will be discarded, i.e. into bit 14 in Fig. 6 in the fractional case.

The reason for preferring fractions in digital signal processing or control is that products, or accumulated products with scalar product computation, can easily be cut down to the size of the factors for storage and further processing by dropping the least significant l-1 bits. Fractional fixed-point arithmetic thus trades precision for number growth. Integer arithmetic on the other hand would not allow for this. It is always exact but at the price of excessive risk of overflow. Overflow of course can also happen with fractional arithmetic in add or subtract operations, but not with multiplication. Sometimes implementors of digital filters or controllers claim to use "integer arithmetic". A closer look however shows that indeed processor instructions for integer arithmetic are used, but there is "scaling", "shifting" and the like. In fact, fraction arithmetic or something close to it is actually performed.

4.1.2. *Overflow.* Because of the limited number range with usual wordlength, say 16 bits, care must be taken that data, for instance controller states, and coefficients fit well into this range. Numbers should not exceed the range, but at the same time should not be so small that the quantization has undesirable effects. Controller scaling and realization structure selection are the major means to achieve this. These are considered in Sections 5 and 6.

In the case of scalar product computation, which is the basic operation with the controller equations, the partial sum overflows can be ignored with two's complement arithmetic, as mentioned above, provided the final result is guaranteed to be in range, but there may be quantities to be computed during evaluation of the controller equations which cannot be guaranteed never to overflow, so there may be results not guaranteed to be in range. This is very likely the case for controller outputs, i.e. actuating signals, but may also apply to state variables.

Two's complement arithmetic then suffers from "wrap-around". For instance adding binary 0.010 (0.25 decimal) and 0.110 (0.75 decimal) yields binary 1.000, which would erroneously be interpreted as -1 decimal in two's complement fractional arithmetic, whereas the saturated binary value 0.111 (4-bit arithmetic assumed) would be preferable. This means that the desired saturation (Fig. 7) must be provided by code (Loges, 1985).

Signal processors sometimes incorporate optional saturation hardware intended for such cases, but the problem is that intermediate results, i.e. partial sums, are better not saturated because this would destroy an otherwise possibly non-overflowing result. The decision about whether the final result is in overflow and with which sign can only be made if there are enough spare bits in the accumulator to the left of the leftmost bit of the result to be stored away (Fig. 6). Perhaps the processor's accumulator provides a few bits for this purpose, but they may be too few for long scalar products, or the processor provides none at all. Overflow processing then requires computation of a downscaled scalar product which does not overflow, and a rescaling operation preceded by overflow checking, i.e. first $a' := c'^T x$ is computed instead of $a := c^T x$, with $c'^T = 2^{-p} c^T$, $p \geqslant 1$. Then the content of the accumulator (a') is either left shifted p positions under saturation, if the processor provides for this at enough speed, or the result is read out of the accumulator displaced by p bits, see Fig. 8 for an example. Both operations are equivalent to multiplying a' by 2^p, correcting for the downscaling of c^T.

4.1.3. *Signal quantization.* As discussed above, products are of almost double length and thus must usually be cut down to the size of the factors. If the processor's accumulator is double length, which is quite often the case, the products are accumulated in full length and the truncate or round operation is performed only with the final result. In any case truncation or rounding introduces a quantization error into the computations. Note that additions and subtractions are exact as long as there are no overflow problems.

Discussion of the influence of the quantization error was always an issue in the digital filter field, and can be found in most textbooks (for instance Oppenheim and Schafer, 1975), but there were also early papers in the control field (Bertram, 1958; Slaughter, 1964; Johnson, 1965, 1966; Knowles and Edwards, 1965a, 1965b, 1966; Lack, 1966; Curry, 1967), and the issue is now also to some extent dealt with in digital control textbooks, particularly in Katz (1981), Franklin and Powell (1980), and Jacquot (1981). Quantization (of variables or signals; for coefficients see Section 5) introduces three effects: bias, noise, and limit cycles. Bias is introduced with truncation, because in two's complement trunc $(x) < x$ for x positive as well as negative. It is better to use rounding, which is quite easily achieved, as mentioned above.

4.1.3.1. *Noise model.* The noise model of quantization is widely used and replaces the quantizer by a purely linear gain block followed by an injection of an additive white noise sequence, uncorrelated with the input. Two's complement arithmetic with truncation or rounding is assumed here, otherwise there could be correlation (Claasen *et al.*, 1975). If the quantization step is described by q, which is equal to 2^{-B} according to (23), then the noise statistics are taken as follows:

$$\begin{array}{lll} \text{variance} & \sigma^2 = q^2/12 & \\ \text{mean} & \mu = -q/2 & \text{for truncation} \\ & \mu = 0 & \text{for rounding.} \end{array} \qquad (27)$$

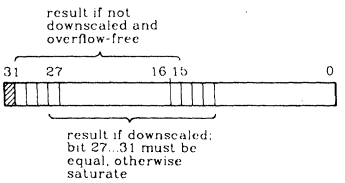


FIG. 8. Scalar product scaling example (fractional numbers, $p = 3$, TMS 32010 processor).

The expressions for σ^2 and μ follow from the assumption of uniform quantization error distribution in the q interval. As has been shown by Widrow (1956, 1961), Katzenelson (1962), Sripad and Snyder (1977), and Boite (1983), this assumption is valid under some conditions, particularly if the amplitudes of the signal to be quantized are not too low. A Gaussian signal, for instance, with variance a few times greater than $q^2/12$ already renders the model very near to what has been evaluated analytically and experimentally.

This classical noise model is, however, based on the assumption of a continuous amplitude input to the quantizer. This is the situation of AD-converter quantization, but within the digital computations the quantizer input is not continuous. In fact, with the rounding of product of a coefficient and a variable (state variable for instance) which is already quantized, the model predicts noise variance less accurately (Halyo and McAlpine, 1971; Sjoding, 1973; Eckhardt, 1975; Boite, 1983). Then there are pecularities leading to coefficient-dependent noise variance, and additionally correlation of error and signal may become significant even with larger signal amplitudes (Barnes *et al.*, 1985).

The noise model of quantization can easily be exploited to compute the total noise contribution to every variable of interest within a control system using standard covariance computation techniques for linear systems (Franklin and Powell, 1980; Moroney *et al.*, 1983). Transfer function-based variance computation is also possible, for instance via the simplified methods given by Patney and Dutta Roy (1980) and Mitra *et al.* (1974).

4.1.3.2. *Limit cycles.* Of course, the noise model of quantization is only an approximation. If signal variations are small compared to q, such as near the steady state of a control system, the non-linear nature of quantization shows up. The result may be limit cycles. Limit cycles observed in practical control systems are often due to the quantization of AD- and DA-conversion, but may as easily be caused by arithmetic. Since there are many quantizers at the same time, analysis of limit cycles in a closed loop control system is difficult. Much has been published on limit cycles in digital filters operated open loop, but the results are of little significance in a closed loop control system. This has been clearly pointed out by Moroney (1983), who gives a comprehensive discussion of the approaches in the digital filter field and their relevance to control.

Some discussion of limit cycle existence for SISO systems and some techniques for bounding their amplitudes (whether they exist or not) are also given by Ahmed and Belanger (1984b). The basic idea of such bounding techniques is to exploit the boundedness of the quantization errors and to check which signal amplitudes can be generated from those error sources. Absolute (Long and Trick, 1983) as well as rms (Sandberg and Kaiser, 1972) bounds, partly exploiting the periodicity of a limit cycle, have been derived for filters, and have been used for control by Ahmed and Belanger (1984b). They also demonstrate that for low external input (reference or disturbance) signal amplitudes limit cycles may be dominant in the output, but for increasing amplitudes the noise model of quantization comes into play and limit cycles may be quenched off, resulting in less output noise than for low input signal amplitudes.

The value of the available techniques for limit cycle bounding for higher order multivariable control systems seems however to be limited. Since they have to be carried out numerically for given parameters, it is probably more attractive to check the effects directly via simulation in practice, taking into account realistic input signals. Note that even slight measurement noise may already quench off limit cycles in the critical "steady state" situation. This is the same effect which sometimes leads to the deliberate introduction of dither signals in non-linear systems. On the same lines is the technique of random rounding known from digital filters (Callahan, 1976; Büttner, 1977).

4.1.3.3. *Double precision arithmetic and error feedback.* In Fig. 6 it has been assumed that accumulation in scalar products is carried out with the full length partial products. Quantization occurs only when the result is stored away and (assuming fractionals) the least significant bits are discarded. To compensate somewhat for the discarded residues thus produced they could be stored too, and included in some simple way in the next sample computation. This technique, called "error feedback", plays some part in the digital filter field (a recent paper is by Vaidyanathan, 1985), and has also recently been proposed for Kalman filter implementation by Williamson (1985). Such techniques are however not far away from performing double precision arithmetic (on signals, not coefficients), as has been pointed out by Mullis and Roberts (1982).

A special technique for performing almost double precision scalar product computation in an efficient way has been described by Loges (1985) for a signal processor. Even if both coefficients and signals are desired to have extended precision this technique leads only to a four-fold increase in processing time. This is quite good because doing anything other than performing the arithmetic the processor is designed for (16-bit in this case) is difficult and normally costs a lot of instructions.

4.2. *Floating-point arithmetic.* If standard wordlength floating-point arithmetic can be used, there is usually no reason to worry about accuracy and dynamic range, provided that the numerical values of data are in a reasonable range and computation of small differences of large numbers is taken care of. The usual single precision format (standard IEEE 754) consists of the mantissa's sign bit, an 8-bit biased exponent e, and 23 mantissa bits for the fraction f. The decimal value is given by

$$(-1)^s \cdot [2^{e-127}] \cdot (1+f). \tag{28}$$

The dynamic range spans $2^{-126} \approx 10^{-38}$ up to $2^{+128} \approx 3 \cdot 10^{38}$, and the accuracy according to 2^{-23} as value of the least significant bit in f corresponds to about seven decimal places.

If much shorter wordlength floating-point formats were used it might however be necessary to introduce scaling to keep data in the dynamic range, as discussed in Section 6, and quantization effects might become significant. Note that a fundamental difference from fixed-point quantization is that there the error is an absolute one, i.e. the noise model may assume noise injection to be independent of the signals, but with floating-point arithmetic the error is a relative one, dependent on the signal amplitude.

Studies of quantization errors for floating-point arithmetic operations and the resulting signal to noise ratio decrease effects in digital filters go back to the end of the sixties (Sandberg, 1967; Weinstein and Oppenheim, 1969; Liu and Kaneko, 1969; Kaneko and Liu, 1973; Fettweis, 1974). There are also studies concerning digital control. Rink and Chong (1979a) derived an upper bound for the variances of the plant state in a state feedback plus observer regulator control system in a stochastic setting. The bound can be quite loose, however. More accurate analysis is possible by computing covariances directly (Rink and Chong, 1979b). Van Wingerden and de Koning (1984) studied the increase of the cost function due to roundoff noise from mantissa rounding when an LQG state feedback is implemented using floating-point arithmetic. Some examples indicate good agreement between roundoff analysis and simulation. Emphasis is placed on derivation of approximate expressions for means and variances of errors in floating-point addition and multiplication by improved modelling of quantization. Phillips (1980) proposed a simulation scheme for evaluating the variance of the error between a control system output in the infinite and finite wordlength cases under the assumption of a deterministic input (reference or disturbance). This approach is however not far away from dispensing with analysis and checking for wordlength directly with simulation. Generally, the value of existing roundoff analysis for practical purposes seems limited. Results can perhaps more easily and more significantly be found by simulation, which is also more easily adaptable to complicated situations, for example if different wordlengths are to be used at different points in a controller.

4.3. *Non-standard arithmetic.* Apart from the common fixed- and floating-point binary data formats and arithmetic, there are at least two other candidates, logarithmic and residue arithmetic.

Logarithmic number representation might seem to be particularly well suited to control. Let the value to be represented be v, and fractional number range be assumed, i.e. $|v| < 1$, then ϕ in

$$v' = v + \Delta v = \text{sign}(v) \cdot D^{\phi}, 0 < D < 1 \qquad (29)$$

could be stored as a conventional binary number in the processor, representing v' which is the quantized version of v, with Δv as quantization error. Practical values of D would be close to 1. The interesting property of this representation is that the quantized values are unevenly spaced. With fixed-point numbers, spacing is equal and quantization error is absolute. With logarithmic number representation closest spacing is achieved in the low magnitude range. If control system trajectories for large state transitions are not required to be very close to the infinite precision ones, the higher quantization errors resulting from large signal magnitudes may be tolerable. If in steady state operation the signals (controller states, outputs, partial sums) are of low magnitude, the increased resolution in that range may be beneficial, leading to lower quantization noise or less limit cycle amplitude. Interfaces to the plant should however also be logarithmic. This is non-standard but possible for AD- as well as DA-conversion, for instance via switched attenuator networks. The arithmetic computations inside a logarithmic number processor are obviously simple in the case of multiplication. Addition and subtraction require logarithm computation but this can be replaced by table lookup (Kingsbury and Rayner, 1971; Swartzlander and Alexopoulos, 1975; Etzel, 1983; Frey and Taylor, 1985).

The use of logarithmic number representation for digital filtering has been proposed by Hall *et al.* (1970) and Kingsbury and Rayner (1971), preceded by yet earlier proposals motivated by construction of calculators, and has been discussed in several papers since then. The digital control application has already been mentioned in Lee and Edgar (1977) and Edgar and Lee (1979). They proposed a number system with an integer and a fractional part. The representation corresponding to (29) has recently been proposed as a basis for a special-purpose control processor by Lang (1984).

For control there seem to be two main problems. The first is that the controller coefficients are not likely to be of low magnitude, thus they are quantized relatively coarsely and possibly this is detrimental to the control system's behaviour. The second is that with practical control systems pure logarithmic signal representation will frequently be simply inadequate. Imagine, for instance, a position control system involving high resolution shaft encoders, where the position values are required to be represented with equal absolute accuracy over the entire range. The assumption that near steady-state operation leads to near-zero signals will often be unjustified, for instance when measurement signals are to be processed or preprocessed separately from reference signals, instead of taking differences first.

The last number system to be mentioned here is the residue number system (Waser and Flynn, 1982). It was proposed long ago for arithmetic unit construction and digital filtering. It also showed up in control-related publications (Tan and McInnis, 1982; Pei and Ho, 1984). The main advantage is that very fast computation is possible because operations are on digits instead of whole numbers. There are no carries, possibly propagating through all digits, thus slowing down the hardware. A high degree of parallelism is possible in principle. It may well be that residue arithmetic will gain ground in special purpose processor designs.

5. *Structures*

5.1. *Basic issues.* Frequently, a state space description of a controller or controller subsystem is derived in a manner motivated by design theory. An example is the observer/state feedback controller (1), (2), where the state has a physical meaning (assuming that the plant state had one) and corresponding matrices are involved. If it is not necessary to preserve the state meaning, but achieving the desired closed-loop control is the only objective, then any system with equivalent i/o behaviour from input to output will do the job. There may be i/o equivalent systems which are preferable to the original one in the following respects:

- number of storage elements;
- number of non-zero non-one coefficients;
- computational delay;
- multi-input/output capability;
- state space description possible or not;
- coefficient range;
- coefficient sensitivity;
- round/truncate noise.

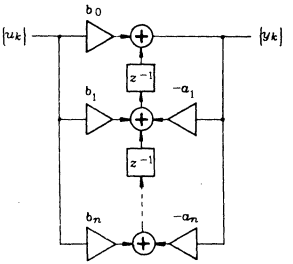


FIG. 9. A direct structure.

If transfer functions are the starting point there may be seemingly natural choices for obtaining programmable difference equations, such as (13) for (12), but other i/o equivalent equations may be preferable. Traditionally, specific organizations of the difference equation computation are depicted in block diagrams involving the z^{-1} or delay element, as in Fig. 9, so that the structure becomes visible. The term "structure" (or synonymous "form") is also used generally, for instance when one state-space description (6) is transformed into another by a similarity transformation

$$\begin{aligned} \tilde{A} &= T^{-1}AT \\ \tilde{B} &= T^{-1}B \\ \tilde{C} &= CT \end{aligned} \qquad (30)$$

yielding new matrices with different zero/non-zero entry patterns, or at least new numerical values.

Determination of "good" structures has always been a main issue in digital filtering. It seems to be quite reasonable to adopt for control purposes structures which proved to be useful in this field. However, some aspects are usually not addressed in digital filtering, namely computational delay, MIMO capability, and the influence of the closed-loop operation. In the following some basic structures are discussed without taking the closed control loop into account; work on this is reviewed in the penultimate subsection. All discussions are on system (6).

5.1.1. *Direct structures.* The simplest case to consider is realization of a SISO transfer function $G(z)$ from (12). In (13) a corresponding structure is given in terms of its difference equation. This structure belongs to the class of so-called direct forms or structures because the polynomials appear directly as coefficients in the difference equation or block diagram. As given in (13), $n + m$ delay or storage elements would be needed, but this can be remedied. Various direct structures can be derived and a few of these at least can be found in any textbook on digital filtering or control, for instance in Phillips and Nagle (1984), Oppenheim and Schafer (1975). In Fig. 9 one of the direct structures is shown, assuming $m = n$ for convenience. This structure can easily be extended to the MISO case.

It is well known that direct structures suffer from various drawbacks. First, the coefficients can easily be spread over a large number range, causing problems with number representation and arithmetic. This is because, according to Vieta's theorem, sums, products, and sums of products of polynomial roots form the coefficients, and roots can be anywhere from the origin even to outside the unit circle in the z-plane with

controllers. This is somewhat in contrast to digital filters, where poles and zeros are usually positioned well off the origin. Second, the sensitivity of roots or coefficient errors can be up to infinity. Such errors are introduced by the quantizing of coefficients to represent them in the processor. If λv denotes a root of

$$p(z) = z^n + \beta_{n-1}z^{n-1} + \cdots + \beta_0 \tag{31}$$

and β_i is perturbed by $\Delta\beta_i$, then λ_v is shifted by $\Delta\lambda_v$ and $\Delta\lambda_v$ is given (to first order, denoted by $\doteq$) by

$$\Delta\lambda_v \doteq -\frac{\lambda_v^i}{\sum_{\substack{j=1 \\ j\neq v}}^{n} [\lambda_v - \lambda_j]}\Delta\beta_i, \tag{32}$$

which clearly indicates high root sensitivity for clustered roots (Kaiser, 1966; Oppenheim and Schafer, 1975).

The situation becomes particularly bad when unclustered "slow eigenvalues" in the s-plane generate clusters near 1 in the z-plane. There is some remedy to this case by means of "delay replacement" (Agarwal and Burrus, 1975; Nishimura *et al.*, 1981; Orlandi and Martinelli, 1984; Goodwin, 1985; Middleton and Goodwin, 1985). One version of this is to replace the z^{-1} blocks by so-called δ^{-1} blocks. A δ^{-1} block realizes the z-transfer function $T/(z-1)$ and thus represents a discrete integrator. Implementing a δ^{-1}-block requires the operation

$$\alpha_k = \delta^{-1}\beta_k = \alpha_{k-1} + T\beta_{k-1} \tag{33}$$

(α output variable, β input variable of the δ^{-1}-block), instead of the z^{-1} shift operation. A z-transfer function then transforms into a δ-transfer function, which can be realized using any suitable structure known for z-transfer functions, but now involving δ^{-1}-blocks instead of z^{-1}-blocks. The advantage over the z^{-1}-block based realization is that the corresponding z-poles can be orders of magnitude less sensitive to errors in δ-polynomial coefficients, just in the case of pole clusters near $z = 1$ as introduced with relatively fast sampling.

The first order root sensitivity of (32) is not always of great importance, but sensitivities of impulse response (Knowles and Olcayto, 1968) and frequency response (Crochiere and Oppenheim, 1975) are high as well with direct structures. Another related drawback is potentially high gain sensitivity. Assuming a stable transfer function $G(z)$ with notation of (12) for simplicity, the final value of the output after a unit step input is

$$y_k|_{k\to\infty} = \frac{\sum_{i=0}^{m} b_i}{1 + \sum_{i=1}^{n} a_i}. \tag{34}$$

A direct structure, directly involving the quantized versions of b_i and a_i, is now likely to introduce inaccurate small differences of large numbers in (34), because the coefficients are frequently of large absolute value with alternating signs. Finally, direct structures suffer from particularly high signal quantization noise, which relates to high coefficient sensitivity (Fettweis, 1972, 1973; Jackson, 1976). The conclusion drawn from all this is to recommend direct structures only for low order systems or subsections of higher order systems, and to use them with care.

5.1.2. *Cascade structure.* A more reliable SISO structure, which is well accepted in digital filtering, is the cascade structure, where $G(z)$ is implemented in factorized form as a series connection of low order blocks, usually of first or second order (Oppenheim and Schafer, 1975), see Fig. 10 for an example. This structure offers possibilities of optimal distribution of poles and zeros among the blocks, and internal block structures can be chosen optimally. However, there are drawbacks for control application. First, the structure introduces increased computational delay in the common case of $G(z)$ having direct feedthrough, i.e. $b_0 \neq 0$ in (12). This is because output appears only after computation in every block is finished, unless direct feedthrough is bypassed directly from input to output, which

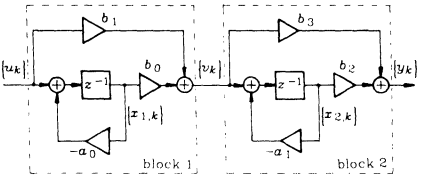


FIG. 10. Cascade structure example.

means departing from pure cascade structure. The second drawback is that this structure is limited to the SISO case, so that it may be valuable for SISO subsystems in a complex structured controller, but not for a complete MIMO controller.

5.1.3. *Parallel structure.* A structure which is widely regarded to be as good a candidate as the cascade structure is the parallel structure (Jackson, 1970b). It corresponds to implementing $G(z)$ in a partial fraction expansion form (Gold and Rader, 1969). The partial fraction blocks are commonly chosen to be of first and second order and can be implemented by suitable structures. Special cases of the parallel form have received much attention in digital filtering as being suboptimal in some respect to certain optimal structures, as discussed below (Jackson *et al.*, 1979; Mullis and Roberts, 1976). An advantage of parallel structures is that they can be used in MIMO cases.

5.1.4. *Other structures.* The above discussion does not cover all types of structure. There are several additional structures of practical importance known in digital filtering, such as wave digital filters or ladder structures. For an overview and bibliography see the recent paper of Fettweis (1984). However, such structures have not yet appeared in control applications.

5.1.5. *Relevance of non-state-space structures.* That a structure can be described by standard state-space models might be taken for granted by control engineers who are used to thinking in state-space terms, but there are many structures which cannot be represented by a single standard state-space model (Willsky, 1979; Moroney, 1983). This is because a state-space structure places restrictions on what nodes may be present. Take the cascade structure of Fig. 10 for example, where signal v occurs at a node not accounted for in a state-space model. If the cascade structure is restructured to map into a state-space structure, such as in Katz (1981), other coefficients are involved and intermediate signals are no longer represented. This is only irrelevant under infinite precision arithmetic. A useful description solving this representation problem is discussed in the last subsection.

5.2. *State-space structures.* Given a state-space description (6) (treatment of types other than (6) is obvious), an infinite number of structures can be derived via similarity transformation (30). The control engineer may be tempted to pick well-known canonical forms first, such as a control canonical form. This and related forms, however, involve transfer-function polynomial coefficients more or less directly and thus suffer from the problems discussed above. The only real advantage of such structures is their minimum coefficient count. For lower order systems or subsystems, they may be used in general without problems, although the author has encountered practical applications where canonical forms of only third order caused accuracy problems even with 32-bit floating-point arithmetic.

More promising for higher order controllers are the parallel structures. A typical one has a block diagonal $\tilde{A}$

$$\tilde{A} = \begin{bmatrix} \lambda_1 & & & & 0 \\ & \ddots & & & \\ & & \lambda_r & & \\ & & & \bar{A}_{r+1} & \\ 0 & & & & \ddots \\ & & & & & \bar{A}_l \end{bmatrix} \tag{35}$$

with (2,2) submatrices $\bar{A}_j$ accommodating complex eigenvalues, and $\bar{B}, \bar{C}$ possibly non-zero everywhere. This or related structures play an important role in digital filtering. It is a special case of that devised by Mullis and Roberts (1976) with special $\bar{A}_j$ as suboptimal quantization noise structure. The latter leads to dense $\bar{A}$, requiring much computational effort, and has thus been considered unattractive. Several authors took the block-optimal structure as a starting point and then focussed on the second order structures, i.e. on the $\bar{A}_j$ and the associated parts of B and C (Jackson *et al.*, 1979, Barnes, 1979, 1984; Mills *et al.*, 1981; Bomar, 1985). The second order substructure also attracted authors because results on overflow stability and limit cycles could be derived (Mills *et al.*, 1978; Jackson, 1979).

Various structures can be chosen for the second order subsystems which accomodate complex eigenvalues $\sigma \pm j\omega$ and the selection may be guided by the sensitivity, quantization noise, or limit cycling considerations discussed in the literature quoted above. The coefficient number range and the number of coefficients contributing to the computation time may also influence the decision. If, for instance, the number of non-zero non-unity coefficients is to be minimized, control canonical or observer canonical forms may be of interest, e.g.

$$\bar{A}_j = \begin{bmatrix} 0 & -\sigma^2 - \omega^2 \\ 1 & 2\sigma \end{bmatrix} \tag{36}$$

$$\bar{c}_j = (0, 1)$$

for a MISO controller. Another choice is

$$\bar{A}_j = \begin{bmatrix} \sigma & \omega \\ -\omega & \sigma \end{bmatrix} \tag{37}$$

with no special pattern in $\bar{B}_j$, $\bar{C}_j$. The resulting matrix $\bar{A}$ is well known in control-related algebra as a real valued version of the diagonal form. Stable controllers always have $|\sigma| < 1, |\omega| < 1$, thus $\bar{A}_j$ is well suited to fractional arithmetic. Transformation of any state-space model of the controller into the real diagonal form (35), (37) can easily be achieved using standard EISPACK software, provided the eigenvectors of A are sufficiently linearly independent in a numerical sense. A successful transformation using CAD software does not however guarantee that the resulting state-space model can be implemented with sufficient accuracy with shorter wordlength arithmetic on the target processor. Problems with large numbers in $\bar{B}$ and $\bar{C}$ can be expected. They correspond to large residues in a partial fraction expansion of the transfer functions, where contributions of terms are likely to almost cancel, thus producing large errors. The author encountered a case in a practical application where three real eigenvalues spaced 5% from each other caused such accuracy problems even with 24-bit mantissa floating-point arithmetic.

The same problems may occur with any attempt to force a model into any parallel structure in cases where there are clustered eigenvalues requiring a series connection representation instead of a parallel one. The obvious way to treat such cases is to introduce parallel blocks of higher order with appropriate internal structure. Clustered eigenvalues could then be accommodated within a Jordan block or a companion form block. But this should not simply be done after the observation of eigenvalue clusters without checking the residues, because clusters with inherent parallel block structure also occur. Additionally, it is with clustered eigenvalues that companion forms suffer from high eigenvalue sensitivity. The problems just mentioned have astonishingly not been an issue in digital filtering. The Jordan form played some part in Barnes and Fam (1977) but not with respect to the residue problems mentioned.

Particular types of state-space structures which have received a lot of attention are the minimum roundoff-noise structures proposed by Mullis and Roberts (1976) and Hwang (1977). They minimize signal quantization noise arising from the state update computation in (6) while retaining scaling of the state vector. Scaling is performed in such a way that the overflow probability is made equal for every state variable assuming a white noise input signal. The reasoning behind the optimal minimum roundoff realization is based on the derivation of a lower bound on the variance of the output noise generated by roundoff in the state vector computation. A lower bound exists because the scaling constraint has to be met. Attaining the lower bound is possible, and a corresponding transformation matrix T can be constructed.

The optimal realizations suffer however from the fact that $\bar{A}$ generally has no specific structure. All coefficients can be non-zero and non-unity. This has always been considered unattractive. But with a digital signal processor as a target the computation of long scalar products is not so time consuming in relation to other operations such as overflow management. If for example an optimal realization enabled single-word arithmetic to be used, whereas a structure with sparse $\bar{A}$ demanded multi-word arithmetic, the former might lead to the faster solution. Considering optimal structures in control applications thus seems worthwhile. Moroney (1983) adapted the theory to closed-loop operation, but focussed on the block-optimal case. From his numerical example, as well as from open-loop filter examples by Jackson *et al.* (1979), there is some indication that non-optimal parallel structures with second order blocks perform quite closely to corresponding block-optimal ones.

5.3. *Closed-loop considerations.* It is quite useful to have a collection of "known-to-be-good" structures and guidelines from which to select under given conditions. In most cases such a selection without closed-loop optimization will be sufficient. Given a 16-bit target processor, for instance, it does not matter much whether the minimum wordlength necessary to achieve satisfactory closed-loop operation is 8- or 10-bit, because 16-bit will be the increment. The situation changes, however, at the boundary, and in cases where wordlength is not fixed, as in custom VLSI processor design. Methods of structure selection or optimization considering the closed-loop operation, which optimize with respect to roundoff noise from signal quantization as well as with respect to coefficient quantization effects, should be useful in such cases. These issues have been studied by Moroney (1983), Moroney *et al.* (1980, 1983), and Sasahara *et al.* (1984). All assume a stochastic setting in an LQG context.

As mentioned above, Moroney *et al.* adapted the theory of Mullis, Roberts and Hwang to the closed-loop SISO case and additionally devised an iterative structure optimization technique for minimizing roundoff noise, which could be augmented to extend optimization to coefficient wordlength effects. The objective is to minimize the increase of an LQG cost function. The influence of coefficient wordlength is introduced via a statistical wordlength technique. The idea of statistical wordlength estimation is already found in Knowles and Olcayto (1968) and was later used by Avenhaus (1972) and Crochiere (1975), who estimated filter frequency response errors by assuming coefficient quantization errors to be independent random variables, leading to a variance estimate on frequency response. This is not as pessimistic as the equally possible worst-case bound, which is based on the assumption that individual coefficient errors are maximum in absolute value with signs opposed to the corresponding sensitivity of the response to the coefficient. But Crochiere's examples show that the statistical estimate is likely to be still somewhat pessimistic.

The statistical wordlength concept has been applied by Moroney (1983) to estimation of LQG cost function degradation. Second order sensitivities are involved because first order sensitivities are zero owing to LQG design. The structure optimization technique of Moroney allows for constraints in the structure, so the matrices of the controller description can be kept sparse, if so wished. Furthermore, the class of structures considered is wide, because everything is done for a generalized state-space structure discussed in the next subsection.

Sasahara *et al.* (1984) also minimize cost function degradation (for digital filters see also Kawamata and Higuchi, 1985). They derive a transformation matrix T for a Kalman filter, plus state feedback controller, which minimizes degradation due to signal quantization noise. So far this is also an adaptation of the Mullis, Roberts and Hwang theory to closed-loop control. From statistical modelling of coefficient quantization errors they then conclude that this approximately minimizes coefficient quantization degradation too. This conclusion is in line with results from digital filtering (Fettweis, 1973; Jackson, 1976;

Jackson *et al.*, 1979; Antoniou *et al.*, 1983) also showing close relationships between minimal noise and minimal sensitivity. An example given by Sasahara *et al.* shows large improvements in cost function degradation using the optimized structure compared to a direct form, and improvement on an unfortunately not specified canonical form is also considerable. Agreement between analysis and simulation appears to be very good for roundoff noise, but less so for coefficient quantization.

Since LQG cost function degradation is not always a suitable objective in practical applications, other means of analysis and optimization should also be developed. Quite effective tools could probably be derived from closed-loop eigenvalue sensitivity analysis. Closed loop frequency response sensitivity might also be interesting, possibly exploiting non-approximate large-change sensitivity expressions as discussed for digital filters by Jain *et al.* (1985).

5.4. *Serialism.* As mentioned in Subsection 5.1.4, the cascade structure of Fig. 10 cannot be described as a standard state-space model. Obviously, variable v between the first order blocks cannot be represented because it is neither a state nor an output and these are the only variables, i.e. network nodes, available in state-space formulation.

From another viewpoint, the example possesses serialism, whereas in a state-space structure all state vector components could be updated in parallel from the "old" state and the input vector. In order to describe more general structures (the cascade is only one example), it is necessary to account for precedence.

Crochiere and Oppenheim (1975) distinguish node precedence from multiplier precedence. In the structure of Fig. 10 there are two node precedence levels: first node signal v_k must be computed, then y_k. There are also two multiplier precedence levels: multiplications involving a_0, b_0, b_1, a_1, b_2 for instance could be performed in parallel first, but multiplication with b_3 has to await computation of v_k. However, the number of multiplier and node precedence levels is not always the same. The motivation for considering multiplier precedence lies in the dominance of multiply execution time frequently encountered. The number of multiplier precedence levels of a structure then determines the minimum sampling period achievable assuming that as many multiplies as possible are carried out in parallel using multiple arithmetic units.

This issue may be of importance in special purpose processor design, but precedence also has important implications in the usual single processing unit case. One implication is that minimum achievable computational delay in the case of direct feedthrough is dependent on precedence, another is that structures with precedence might be preferable with respect to finite wordlength effects. In this case it is necessary to have a description of the structure representing the original coefficients and the original node signals. Such a description has been introduced to the control field by Moroney (1983) and Moroney *et al.* (1980, 1981, 1983). It had previously been used with digital filters by Chan (1978), and recently by Mullis and Roberts (1984) in a VLSI filter chip design context, labelled factored state variable description (FSVD). Using this description, the structure of Fig. 10 would be represented by

$$\begin{bmatrix} x_{1,k+1} \\ x_{2,k} \\ v_k \end{bmatrix} = \begin{bmatrix} -a_0 & 0 & 1 \\ 0 & 1 & 0 \\ b_0 & 0 & b_1 \end{bmatrix} \begin{bmatrix} x_{1,k} \\ x_{2,k} \\ u_k \end{bmatrix} \begin{bmatrix} x_{1,k+1} \\ x_{2,k+1} \\ y_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -a_1 & 1 \\ 0 & b_2 & b_3 \end{bmatrix} \begin{bmatrix} x_{1,k+1} \\ x_{2,k} \\ v_k \end{bmatrix} \tag{38}$$

or

$$r_k = \psi_1 \begin{bmatrix} x_k \\ u_k \end{bmatrix}$$

$$\begin{bmatrix} x_{k+1} \\ y_k \end{bmatrix} = \psi_2 r_k. \tag{39}$$

Serialism is now expressed by the first computed intermediate result r_k.

Each ψ_i matrix necessary in a FSVD corresponds to one node precedence level. The intermediate signals can be represented and so can the coefficients. Note that revoking the factorization, introducing $\Theta = \Psi_2 \Psi_1$ immediately yields a standard state-space description (by partitioning Θ into A, B, C, D appropriately) but neither the intermediate signal nor all original coefficients are then represented.

Thus FSVD could be useful for modelling general structures within an implementation oriented CACE environment. Cascade structure is only one example of such a more general (with respect to standard state-space) structure, a delay-replacement state-space structure based on (33) being another one.

In the work of Moroney *et al.* a slight modification has been made. Owing to their restriction to LQG compensators without direct feedthrough (see Subsection 2.2) they introduce the output (SISO case) as a state and call the result "modified state-space representation". All their work, which has frequently been quoted in this paper, is based on this representation.

Another issue linked with precedence is pipelining. In the example in Fig. 10, imagine that there is double hardware, so that multiplies and adds for the left-hand block 1 and the right-hand block 2 can be executed simultaneously, i.e. in parallel. Then simply letting block 2 hardware wait for completion of block 1 computation so that one hardware unit is always idle would of course be unattractive. But if a delay (i.e. storage) is inserted between the blocks for storing v_k, the multiplier as well as node precedence levels are reduced to one, and both hardware units could always be busy, running at double sampling frequency, see Fig. 11. This is pipelining. It allows an increased throughput rate but introduces delay. In a control feedback loop this delay must then be accounted for in design (Moroney, 1983; Moroney *et al.*, 1981) but despite this delay the control system performance can possibly be improved compared to the lower sampling frequency non-pipelined case.

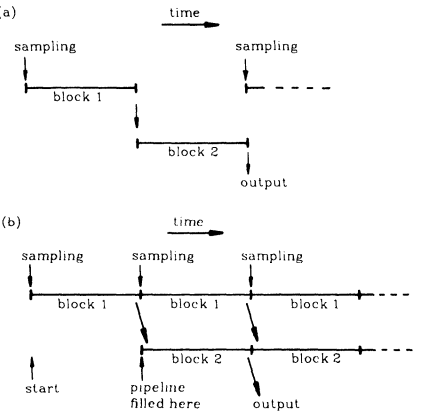


FIG. 11. Pipelining with structure from Fig. 10: (a) unpipelined; (b) pipelined.

6. *Scaling*

At least when using fixed-point arithmetic it is usually necessary to perform scaling on the controller to be implemented. The primary objective is to fit data which are computed during the course of a difference equation calculation into the limited number range, so that overflows are avoided without provoking excessive signal quantization effects. A second objective with scaling is to alter coefficients in such a way that they fit into the coefficient number range. This is not always achieved when

scaling is only oriented towards data overflow avoidance and scale-factors then have to be altered appropriately.

The following discussion is on a controller formulated as a state-space system, but the concepts apply equally well in other formulations, such as (13), for example. The scaling task may be partitioned into three subtasks, which might be called

input and output scaling;
state vector scaling;
scalar product scaling.

They are dicussed below in this order, which also reflects the chronological sequence within the implementation process. Note however that scaling cannot always be handled separately after structure selection. Any kind of structure optimization or evaluation with respect to finite precision arithmetic should have a scaling procedure as an underlying process, because scaling affects the numerical values of the coefficients.

6.1. *Input and output scaling.* During controller design the plant's outputs and control inputs are often conveniently handled as physical variables without normalization, i.e. outputs of a system may be in bar and ms^{-1}. Once the range of values occurring in closed-loop control system operation are known, the transducer gains can be determined. The output of a transducer, say $-10V \ldots +10V$, must be represented in the processor according to the data format used in the controller implementation. Using fractional arithmetic, the bit pattern output of the AD-converter representing $-10V \ldots +10V$ may be aligned to give $-1 \ldots +1$ in the processor, i.e. the most significant bit (msb) of the ADC output is also the msb of the data word then used for the input to the difference equations. In the case of digital input, for instance from a position encoder, the alignment could also be done in this way, and for the outputs of the controller it is just the same.

Let the physical variable range of a plant's input, which has been used in designing the controller, be given for example as y_i in the range $-20A \ldots +20A$ for an electromechanical actuator. This variable is represented in the range $-1 \ldots +1$ in the processor (fractional arithmetic assumed). Possible intermediate variable transformations, for instance into $-10V \ldots +10V$ via a DAC and then into the y_i range via a power amplifier, do not matter here. The gain between the value in the processor and the physical value used in the controller derivation must however be accounted for by scaling the controller equations before supplying them to the further steps of the implementation procedure. In the example the ith row of C and D must be multiplied by 1/20 to obtain the correct numerical values. As a whole, there must be input and output scaling to change the B, C, and D matrices of (6) for example, to BS_u^{-1}, $S_y^{-1}C$, and $S_y^{-1}DS_u^{-1}$ respectively. The scaling matrices are diagonal and their elements are given by

$$s_{u,i} = \frac{R^{pr}}{R^{ph}_{u,i}}, \qquad s_{y,i} = \frac{R^{ph}_{y,i}}{R^{pr}}, \tag{40}$$

where R^{pr} means the number range span in the processor (same number range for inputs and outputs assumed), i.e. 2 for fractional arithmetic, and R^{ph} means the physical variable range span used in the design of the controller which led to the original A, B, C, D matrices, i.e. 40 A for R^{ph}_y. In the case of an unsymmetrical physical variable range, for example 0 ... 40 A, appropriate offset must be added, preferably at the transducer or amplifier side.

In the above discussion it has been assumed that the range of the physical variables, and accordingly the measurement ranges of the transducers are known. This is usually the case. If it is not, the techniques discussed below for determination of maximum deflections of state variables could also be used correspondingly to get that information.

6.2. *State vector scaling.* For a system (6), the state variables are scaled via

$$x_{\text{scaled}} = \operatorname{diag}\left(\frac{1}{s_{x,i}}\right) x = S_x^{-1} x \tag{41}$$

thus the scaled system is given by

$$x_{\text{scaled},k+1} = S_x^{-1} A S_x x_{\text{scaled},k} + S_x^{-1} B u_k \tag{42}$$

$$y_k = C S_x x_{\text{scaled},k} + D u_k \tag{43}$$

leading to new matrices A_s, B_s, and C_s. The scale factors $s_{x,i}$ can always be chosen so that x_{scaled} stays within the number range given by the data format used. But in order to minimize data quantization effects x_{scaled} should not be permanently far off the limits during operation of the closed-loop control system, i.e. scaling should be such that the maximum absolute value of a variable is just below the upper number range limit under worst-case conditions.

What remains to be discussed is determination of the scale factors. Basically, there are two approaches to determining the $s_{x,i}$ analysis and simulation. The conceptually simplest is to simulate the closed-loop control system under various conditions, preferably under conditions which are anticipated to be worst-case with respect to the values of x. The largest absolute values of the components of x can be collected and scale factors can easily be derived from the largest absolute values overall per state variable. All this can be automated by appropriate software. Although the effort in performing a number of simulations might be considerable, the data collection mentioned can often be a by-product of simulations carried out anyway in the case of control design evaluation.

In the digital filtering field, state vector scaling has been dealt with analytically since the early seventies and is represented in most textbooks in this field. Some prominent papers have been published by Jackson (1970a, 1970b), Hwang (1975a,b), Mullis and Roberts (1976). Concepts developed there have been used in control engineering by Moroney (1983), who gives an extensive discussion and bibliography, by Moroney *et al.* (1980, 1983), Sasahara *et al.* (1984), Scharf and Sigurdsson (1984), and Ahmed and Belanger (1984a). In the digital filtering field, the digital systems usually operate in open loop and are of the SISO type. Analytic scaling is based there on certain assumptions about the input signal to be expected. In the MISO or MIMO controller case, such assumptions cannot be made as easily because the plant measurement outputs, which are inputs to the controller, are interdependent according to the plant dynamics and structure. Furthermore, the closed-loop nature of controller operation should be taken into account.

If scaling is allowed to be a bit pessimistic, experience shows that quite often useful scale factors could have been found by driving the digital open-loop controller alone with worst-case input signals. For a SISO servo-control system, for example, full-scale step reference inputs may be supposed to be worst-case. Indeed, the largest deflections of the controller's state variables frequently occur right after the step, and where the plant reacts slowly, the feedback case is not much different from the open-loop case in terms of maximum deflection of the controller state. So, simple calculation of controller state after a step input in open loop yields reasonable scale factors in such a case, called unit-step scaling with respect to filters in Phillips

and Nagle (1984) and also used by Mitchell and Demoyer (1985). Note however that such an approach only works with stable controllers. An integral part in the controller would only be allowed if it affected only one state variable, i.e. if it were decoupled in the controller. It could then be scaled separately, based on what is expected to be specified as its maximum contribution to the actuating variables.

Generally, scaling of controllers would be most safe and least pessimistic when based on closed-loop considerations. As in the open-loop case of digital filters, there must be some assumptions about input signals, but now these are not necessarily input signals to the controller. They may be inputs to the plant, for example disturbances. The closed-loop system must be sufficiently linear, because the analytic scaling approaches rely on linear models.

After a linear discrete model of the closed-loop system has been set up assumptions about input signals are in order. In stochastic settings it is reasonable to assume worst-case stochastic input signals and then to compute the variances $\sigma_{x,i}^2$ of the controller state variables by standard techniques. In the case of zero-mean Gaussian signals, for instance, the probability that the amplitude exceeds $3.3\,\sigma$ is only 0.001, thus a scale factor $s_{x,i}$ in the range $3\sigma_{x,i} \ldots 10\sigma_{x,i}$ should be reasonable for fractional arithmetic overflow limits $-1 \ldots +1$. The actual value selected depends on the supposed quality of the Gaussian model of the real signal.

The variance-oriented scaling approach has been used in connection with control by Moroney *et al.* (1980), Moroney (1983), Scharf and Sigurdsson (1984), Sasahara *et al.* (1984), and Ahmed and Belanger (1984a). If there are constant (not step) disturbances or reference inputs in addition to the stochastic signals, the mean values $\bar{x}_i$ of the controller state variables must also be computed for worst-case situations and scale factors must be selected so that $|\bar{x}_i| + c\sigma_{x,i} \leqslant 1$, $c = 3 \ldots 10$, in the case of fractional arithmetic. Moroney (1983) and Moroney *et al.* (1983) propose some remedy for the non-zero setpoint situation in order to obtain zero-mean controller state variables, but it seems to be rather limited from a practical viewpoint.

Deterministic input signals are probably most often accounted for in practice by simulating the closed-loop system for the operating conditions expected in reality, as mentioned above. Possibly a linear simulation is sufficient for collecting scale factors. In digital filtering other means have been developed to determine scale factors based on deterministic assumptions on input signals. They can be adopted for closed-loop control as discussed by Moroney (1983). The aim is to calculate upper bounds for the controller state variables x_i under some boundedness assumptions on the input signals. The basic idea of bound-based scaling is illustrated by the following.

Given an input signal to the closed-loop system represented by its samples v_k, a controller's state variable is given by

$$x_{i,k} = \sum_{j=0}^{k} h_{i,k-j} v_j, \tag{44}$$

where $\{h_k\}$ is the impulse reponse sequence of this transfer path. The sum in (44) can now be bounded via the Hölder inequality (Epstein, 1970; Hwang, 1975a, 1975b), thus

$$|x_{i,k}| \leq \sum_{j=0}^{k} |h_{i,k-j} v_j| \leq \left[\sum_{j=0}^{k} |h_{i,j}|^p\right]^{1/p} \left[\sum_{j=0}^{k} |v_j|^q\right]^{1/q}, \tag{45}$$

where $\frac{1}{p} + \frac{1}{q} = 1$. Since the factors $(\ldots)^{1/p}$ and $(\ldots)^{1/q}$ are l_p and l_q norms of vectors, scaling based on such bounds is called l_p scaling (Hwang, 1975a, 1975b). The name of the norm used for the impulse response sequence is used by convention.

With digital filtering, l_2 scaling plays an important role in connection with optimal realization structures (see Section 5). In this case the Euclidean norm of the impulse response sequence as well as the input sequence is used, which in fact means that an assumption on an energy bound on $\{v_k\}$ must be made from a deterministic viewpoint. This type of scaling corresponds to the stochastic overflow-probability scaling discussed above in a stochastic setting, with $\{v_k\}$ a white noise sequence, in which case the variance of x_i is given by

$$\sigma_{x,i}^2 = \left[\sum_{j=0}^{\infty} h_{i,j}^2\right] \sigma_v^2. \tag{46}$$

If the only assumption on $\{v_k\}$ were that any sample is bounded by $v_k| \leq M$, then $q = \infty$ *and* $p = 1$ should be taken. Note that this is the limiting case but (45) is still valid (Epstein, 1970). Equation (45) then yields

$$|x_{i,k}| \leqslant M \sum_{j=0}^{k} |h_{i,j}|. \tag{47}$$

It is obvious that this yields an absolutely worst-case, pessimistic bound. Equality in (47) holds if v_j in (44) is always at its limits $+M$ or $-M$ with the sign corresponding to that of $h_{i,k-j}$. Note that the impulse response sequence must form an absolutely convergent series for (47) to be useful, but this is guaranteed for a stable linear closed-loop system (Strejc, 1981). The opposite case to that last discussed is $q = 1$, $p = \infty$, which leads to an assumption on $\Sigma|v_j|$. Only signals whose number sequence is absolutely summable are allowed here.

The norm-based bounding techniques outlined above work in the time domain. Similar techniques are available in the frequency domain based on function space norms, in which case frequency response and input signal spectrum assumptions are involved (Hwang, 1975a, 1975b; Moroney, 1983; Ahmed and Belanger, 1984a).

A strong point of bound-based scaling techniques is that absolutely worst-case (although perhaps conservative) scaling is possible, whereas with simulation the safety of scaling depends on how well the worst-case situations have been anticipated. It was interesting to the author of this survey to check the degree of conservatism in the simple control system example given by Ahmed and Belanger (1984a). They used l_1 norm-based scaling assuming the reference signal to be absolutely bounded by M. It turned out that the l_1 worst-case scaling was not very conservative at all. Compared with a reference step input of value M over-scaling was only about 50%. Since the given data wordlength of a processor may be sufficient to allow for worst-case scaling, it is attractive to let an automatic scaling algorithm perform this. The control engineer then needs only to supply bounds on input signals. Experience with such l_1 scaling applied to the controller alone (open loop) indicates that even this simple automatic scaling method yields good results for stable controllers or controller subsystems.

Note that in the discussion given above as well as in the literature only single input signals have been considered, whereas several signals might act on plant and controller simultaneously in reality. With l_1 scaling this is accomodated for by computing

$$|x_{i,k}| \leqslant \sum_{\nu=1}^{r} \left[M_\nu \sum_{j=0}^{k} |h_{i,j,\nu}|\right], \tag{48}$$

where M_ν are the bounds on the individual input signals, and $\{h_{i,j,\nu}\}$ is the impulse response sequence for an impulse at the νth input.

6.3. *Scalar product scaling.* From the discussion in Section 4, it is not sufficient to scale only the state vector in the case of other than two's complement arithmetic. Partial sum overflow during scalar product evaluation also has to be avoided. With two's complement arithmetic the same is necessary if it cannot be ensured that x_{scaled} is overflow free, and the saturation value should be taken if overflow occurs. The scalar product scaling as discussed in Section 4 can be used for this purpose, leading

to computation of an intermediate downscaled state

$$\tilde{x} = S_{sx}^{-1} A_s x_{\text{scaled},k} + S_{sx}^{-1} B_s u_k, \tag{49}$$

$$S_{sx} = \text{diag}(s_{sx,i}), s_{sx,i} \geqslant 1, \tag{50}$$

which must be rescaled

$$x_{\text{scaled},k+1} = S_{sx} \tilde{x} \tag{51}$$

to yield the new scaled state. This rescaling operation has to be performed by the target processor, whereas downscaling in (49) only modifies the coefficients of the matrices supplied before target processor programming.

A similar situation arises with the output computation (43). With practical control systems it is very likely that outputs of the controller will saturate in certain states of operation. This is a very common case, for instance, with drives and positioning mechanisms. In order to be able to determine correct output saturation it is necessary to perform scalar product scaling here, i.e. first to compute a downscaled overflow-free version of the output vector and then to rescale it using a saturation overflow mechanism.

This downscaling procedure also conveniently scales down the coefficients (matrix elements) of the output equation, which are very often quite large. Values in the hundreds are not uncommon here when fractional arithmetic is used, in which case coefficients should not exceed the range $-1 \ldots +1$ (at least not much; small integer parts may still be realized using multiple adds and subs). The reason for large coefficients here is that in contrast to digital filters, controller transfer functions frequently have gains far above unity. Since the state vector will be scaled so that it fits into the number range, high gains consequently show up in the output equation in the C_s matrix. The direct feedthrough matrix might also have large coefficients, particularly with controllers of PD type where a step input immediately produces a large output. The scalar product scaling technique has been implemented to be carried out automatically in an automatic code generator for a certain signal processor by Loges (1984), Hanselmann and Loges (1984), Loges (1985).

7. *Programming*

In any case where computation speed is not crucial and a common and well-supported general microprocessor is used as target, programming of controllers as introduced in Section 2 should not cause problems. Common general high-level languages (HLL) can then be used, along with convenient floating-point arithmetic. It is, however, necessary to account for real-time operation.

7.1. *Multi-tasking and languages.* Where the controller is the only task for a dedicated processor, timing is sometimes achieved through simply polling a status signal ("ADC ready" for example) of a peripheral which is under timer control, leaving the processor idle while waiting. If there is something more useful for it to do instead of waiting, a foreground/background solution would be better. In that case, the background job is interrupted by a real-time clock whenever the controller has to be served. This type of real-time operation is quite primitive, but may be appropriate for simple systems and is widely used (for an example see Clarke, 1982). It works with HLLs even if they were not originally designed for real-time operation, provided the machine code generated by the compiler is re-entrant. This means that routines which are used in both foreground and background (such as library routines) can be interrupted, re-used, and resumed without errors due to altered local variables.

In a multi-rate system for example, composed of several subsystems, the situation is a bit more complicated, since modules executed at a slower rate have to be interrupted to let high-rate modules be serviced. Foreground/background operation then becomes clumsy. If additionally asynchronous events occur, or if synchronization problems have to be solved, a multi-tasking executive becomes more and more necessary.

There are ways of staying with HLLs, though, because some languages have at least basic real-time operations support built into them, such as Modula 2, some versions of Pascal, and Forth environments, or real-time facilities are achievable through widely available real-time operating system kernels, available to interface with C or Pascal programs for example (Evanczuk, 1983; Ready, 1984; Heider, 1982). Although real-time executives (operating system kernels) are quite an effective means of achieving multi-tasking, they usually require considerable processor execution time for task management. Switching from one task to another may easily take around 100 μs and more, even with a modern 16-bit processor. So, if appropriate, more primitive means might be the choice.

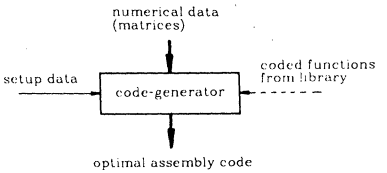


FIG. 12. Automatic code generation.

A problem with HLLs is that they most often only support integer and floating-point arithmetic. If the latter is too slow, fractional arithmetic would be an alternative, but one might be forced to program the equation evaluation parts in assembly language. Emulation of fractional arithmetic through integer computations is possible but with a loss of speed. It is interesting to note that there are Forth language environments which include not only multi-tasking (Pountain, 1985) but also fractional arithmetic. This backs the claim of Forth advocates that this environment is well suited to real-time control, at least for small systems.

The lowest level of programming is of course the use of assembly language. In most cases it is chosen for reasons of speed. With a modern microprocessor the assembly code for implementing a controller can be quite concise owing to powerful instruction sets. For examples of coded digital filters see Phillips and Nagle (1984), where subroutines and loops have been used. Maximum speed is obtained if straight code without loops and subroutines is used because then there is no associated overhead. Straight code, however, contradicts what is good programming style. A satisfying solution to this could come from automatic program generators. Such a generator would generate tailored code once the type, dimensions and numerical values of a controller from Section 2 were known (Fig. 12), and should be fairly easy to write for a general microprocessor.

7.2. *Code generation.* The generator concept has repeatedly been applied to signal processor programming for digital filtering and related tasks (Schafer *et al.*, 1984; Mintzer *et al.*, 1983; Skyttä *et al.*, 1983; Herrmann and Smit, 1983), and also for controller implementation (Hanselmann, 1982; Hanselmann and Loges, 1983, 1984; Loges, 1984, 1985). An interesting project aimed at automatic program generation for a microcontroller (the 8096 from Section 3) has been described by Srodawa *et al.* (1985). Since the starting point is a language description of the computations to be performed, this tool is more a compiler than a code generator. Compilers translating high level descriptions into signal processor code are also emerging commercially, both for special signal processing languages and for suitably modified general HLLs, such as Pascal or C (Marrin, 1985).

An early control-related generator for the Intel 2920 signal processor developed by Hanselmann (1982) was aimed at MIMO controllers in the form of (6). Good experiences with this tool later led to application of the code generator concept to the TMS 32010. A brief description of this generator follows as an example of what can be expected from implementation tools at the programming end today. Details on internals can be found in Loges (1985), and details on how to use it in Hanselmann and Schwarte (1985).

The generator is aimed at implementation of MIMO controllers and accepts the four matrices of (6) as its input. Output is a mnemonic assembly language program, which can be assembled and downloaded to the target. Because everything is automatic, less attention needs to be paid to the readability and length of

the program (as long as there is sufficient program RAM, which is usually the case), and straight code without unnecessary loops and without subroutines is generated to increase speed. The generator also copes with data RAM limitations. If it detects lack of RAM it automatically trades program space against data space by utilizing an immediate multiply instruction of the target processor so that coefficient space is saved in the data RAM. This instruction only accommodates 13-bit numbers, but even in cases where there are too many more-than-13-bit coefficients the generator finds a way out (Loges, 1983, 1985).

Another important option is extended precision arithmetic. The generator provides this on two levels: extended coefficient precision, and extended coefficient and signal variable precision. With the special extended precision computation technique realized by the generator the controller of Section 3.3 and Table 3 for example would run at 7 kHz with full precision (coefficients and signal variables) instead of 31 kHz with single precision. The generator also automatically provides overflow management code along with rescaling of scalar products (see Section 4). Finally, the generator has a facility to include function code automatically. This is particularly useful for extending linear controllers (for which the generator does coding) by non-linear functions

$$\text{destination} := f(\text{destination, states, inputs, aux. variables}),$$

where destintion can be a predefined variable such as a state or output or a user-defined variable used in another function call. A major type of function code performs table lookup with or without interpolation, leading to very fast non-linear function computation. At present, the generator concept is even being applied to tailoring such function code. For instance there is a program which generates square-root function code according to the user's specifications, such as argument range, table length allowed, or precision desired.

Automatic code generation seems to be a viable means of achieving application-specific code with about the same efficiency as an expert programmer coding by hand would attain. This is particularly valuable for target processors with non-standard architectures and instruction sets, such as special signal or custom design processors.

Code generation as just discussed is aimed at production of optimal assembly code, but generation of HLL code should also be mentioned. It helps in translating application oriented descriptions of a controller, for example in the form of a block diagram with transfer functions, into general programming languages. A recent example is the RT BUILD facility of the MATRIX$_x$ CACE-Package (Shah *et al.*, 1985), which generates ADA language source code for controller implementation.

8. *Simulation of digital control systems*

In any realistic control design and implementation project, digital simulation is an invaluable tool. This applies even more to digital control. As mentioned earlier, simulation is useful in the determination of scale factors. It can also reveal effects due to quantization, overflows, spectrum aliasing, and non-simultaneous sampling and output. Such effects are only partly emanable to limited analysis. Very few publications addressing the problems of digital simulation of digital control systems have appeared up to now, although there are indeed several problems, as discussed briefly below. They fall mainly into two groups: efficient simulation/integration methods, and modelling.

If the plant is linear (rare case), a simulation based on transition matrix techniques could be augmented by the modelling of delays (computational, sampling, and output) and quantizers, including overflow simulation if necessary. The usual case, however, will be with additional non-linearities in the continuous part of the control system, so that general integration methods for differential equations must be used. This results in certain peculiarities:

(a) Integration is on the continuous system state only, but the state derivatives depend on the discrete system's outputs, which are held constant between update time instants.

(b) For the sake of accuracy, the integration step boundaries should be made coincident with the controller's sampling and output time instants. This may dictate a small step size with variable step-size integration.

(c) The discrete system introduces discontinuities due to staircase functions. Discontinuities force multi-step integration into restart and since this occurs many times, such integration methods may become inefficient. Fortunately, the time instants at which discontinuities occur are known (as long as there are no sources other than the staircase function output), so if (b) is satisfied there is no need to perform the discontinuity-finding operations (Hay, 1984, 1985) familiar from problems where discontinuity occurrence is state dependent.

(d) Integration of slow subsystems with larger step size (so-called multi-rate simulation (Gear, 1984) may seem to provide a solution to the small step size problem. It requires interpolation in order to provide the samples for the controller, and decimation at fast-to-slow system interfaces which should prevent aliasing effects. Fidelity for instance with respect to limit cycles due to quantization must be questioned. In fact this field seems to be largely unexplored.

The points given are partly addressed by some known simulation packages (for a survey of simulation software see Cellier, 1983), such as MATRIX$_x$ (Shah *et al.*, 1985); SIMNON (Åström and Wittenmark, 1984). There are also commercial packages to be mentioned such as ACSL by Mitchell and Gauthier Inc. and CSSL-IV by Simulation Services. For (a) for instance, there is a so-called DISCRETE section in ACSL which combines with the DERIVATIVE section for the continuous system, and (b) is satisfied because sampling and output instants are placed in an event list supervized by the integration control mechanism, which steers integration step boundaries to coincide with any event. Point (b) can also be satisfied by choosing appropriate integration routines.

The points discussed so far have to do with the event nature of sampling and output and apply to discrete control. They are also partly considered by Stirling (1983) and Zimmerman (1983). Digital control additionally requires simulating AD- and DA-converters, quantizers in general, and possibly overflow behaviour, along with an interactive overflow detection mechanism. Quantizers and limiters are usually available in the package libraries, but not high-level constructs, although it seems possible to build them up from primitives.

A special purpose package where the user no longer deals with quantizers and limiters, but "talks" to the program in higher-level terms such as ADC-wordlength, two's complement arithmetics, 32-bit accumulation, and the like (Hanselmann *et al.*, 1983) proved to be very useful. Simulation on the basis of sufficiently detailed and realistic models abstracted from the actual processor and its software should always be available. In contrast to the processor simulators sometimes supplied by processor vendors, mapping all registers, flags etc., and instructions of a specific device, the use of abstract models yields processor independency. It also allows experiments (with arithmetic for example) which help determine what processor should be used, regardless of availability of the processor or its simulator. This will become particularly important for custom control processor design.

9. *Conclusions*

Controller implementation is a topic involving many disciplines at the same time, from processor technology and electronics through system theory aspects up to software engineering. Even in the rather restricted case of mostly linear control there may be many problems when the idealizations of common theory of algorithms and design methods no longer hold.

Some of the issues arising were already considered in the old direct digital control days in the sixties. Stimulated by microprocessor technology, these issues are once more arousing interest. Some of the problems encountered still require further work, and more experience should be gained to know which of the methods prove to be practical.

Much could be gained by integration of all implementation related tools into CACE software (and also hardware to some extent) environments. All the steps necessary in the implementation process should be integrated into the CACE environment and should be supported as much as possible by software tools (Hanselmann and Loges, 1984; Hanselmann, 1986). Possibilities

range from the minimum of having consistent controller data structures throughout the process up to the coding stage, to the maximum of fully automatic structure selection, scaling, and final code generation for the target processor, accommodating complex controllers, composed of several subsystems, possibly of the multi-rate type.

The advantages of extending CACE to control implementation are now well recognized by control engineers. This is reflected in recent discussions of CACSD/CACE scopes given by Spang (1985), Sutherland and Sonin (1985) and Powers (1985). Designing such software is certainly not a trivial task because of the many disciplines involved, the fast pace of processor technology, and increasing control system complexity.

Acknowledgements—This work is much related to the work of the control systems group at the author's institution, which is headed by Prof. J. Lückel. The author is indebted to his colleagues for fruitful discussions, particularly to W. Loges and A. Schwarte for help with hardware and software. Thanks also to Prof. K.-J. Åström for support and advice in preparation of the paper, and to the anonymous reviewers for their constructive remarks.

References

Agarwal, R. C. and C. S. Burrus (1975). New recursive digital filter structures having very low sensitivity and roundoff noise. *IEEE Trans. Ccts. Syst.*, **CAS-22**, 921.

Ahmed, M. E. and P. R. Belanger (1984a). Scaling and roundoff in fixed-point implementation of control algorithms. *IEEE Trans. Ind. Electron.*, **IE-31**, 228.

Ahmed, M. E. and P. R. Belanger (1984b). Limit cycles in fixed-point implementation of control algorithms. *IEEE Trans. Ind. Electron.*, **IE-31**, 235.

Ahmed, N. and T. Natarjan (1983). *Discrete-time Signals and Systems.* Reston, Virginia.

Antoniou, A., C. Charalambous and Z. Motamedi (1983). Two methods for the reduction of quantization effects in recursive digital filters. *IEEE Trans. Ccts. Syst.*, **CAS-30**, 160.

Åström, K. J. (1983). Theory and applications of adaptive control—a survey. *Automatica*, **19**, 471.

Åström, K. J., P. Hagander and J. Sternby (1984). Zeros of sampled systems. *Automatica*, **20**, 31.

Åström, K. J. and B. Wittenmark (1984). *Computer Controlled Systems.* Prentice-Hall, Englewood Cliffs, New Jersey.

Avenhaus, E. (1972). On the design of digital filters with coefficients of limited word length. *IEEE Trans. Audio Electroacoust.*, **AU-20**, 206.

Barnes, C., B. N. Tran and S. H. Leung (1985). On the statistics of fixed-point roundoff error. *IEEE Trans. Acoust. Speech Sig. Process.*, **ASSP-33**, 595.

Barnes, C. W. (1979). Roundoff noise and overflow in normal digital filters. *IEEE Trans. Ccts Syst.*, **CAS-26**, 154.

Barnes, C. W. (1984). On the design of optimal state-space realizations of second-order digital filters. *IEEE Trans. Ccts Syst.*, **CAS-31**, 602.

Barnes, C. W. and A. T. Fam (1977). Minimum norm recursive digital filters that are free of overflow limit cycles. *IEEE Trans. Ccts Syst.*, **CAS-24**, 569.

Beliczynski, B. and W. Kozinski (1984). A reduced-delay sampled-data hold. *IEEE Trans. Aut. Control*, **AC-29**, 179.

Bertram, J. E. (1958). The effect of quantization in sampled-feedback systems. *Trans. Am. Inst. Elec. Eng.*, **77-2**, 177.

Blasco, R. W. (1983). Floating-point digital signal processing using a fixed-point processor, presented at Southcon; also in *Signal Processing Products and Technology*, Texas Instruments.

Boite, R. (1983). On the quantization of low-level signals: the fixed-point case. *Proc. Eur. Conf. Cct Theory and Design*, Stuttgart.

Bomar, B. W. (1985). New second-order state-space structures for realizing low roundoff noise digital filters. *IEEE Trans. Acoust. Speech Sig. Process.*, **ASSP-33**, 106.

Bondarko, V. A. (1984). Discretization of continuous linear dynamic systems. Analysis of the methods. *Syst. Control Lett.*, **5**, 97.

Booth, A. D. (1951). A signed binary multiplication technique. *Q. J. Mech. Appl. Math.*, **4**, 236. Also in Swartzlander, E. E. (Ed.) (1980), *Computer Arithmetic.* Dowden, Hutchinson & Ross, Stroudsburg, Pennsylvania.

Bose, N. K. (1983). Properties of the *Qn*-matrix in Bilinear transformation. *Proc. IEEE*, **71**, 1110.

Breitzman, R. C. (1985). Development of a custom microprocessor for automotive control. *IEEE Control Syst. Mag.*, 23 May.

Broussard, J. R., D. R. Downing and W. H. Bryant (1985). Design and flight testing of a digital optimal control general aviation autopilot. *Automatica*, **21**, 23.

Büttner, M. (1977). Elimination of limit cycles in digital filters with very low increase in the quantization noise. *IEEE Trans. Ccts Syst.*, **CAS-24**, 300.

Callahan, A. C. (1976). Random rounding: some principles and applications. *Proc. IEEE Int. Conf. Acoust. Speech Sig. Process.*, Philadelphia.

Cappellini, V., A. G. Constantinides and P. Emiliani (1978). *Digital Filters and their Applications.* Academic Press, London.

Cappello, P. R. (Ed.) (1984). *VLSI Signal Processing.* IEEE Press, New York.

Cellier, F. E. (1983). Simulation software: today and tomorrow. In Burger, J. and Y. Jarny (Eds), *Simulation in Engineering Sciences.* Elsevier Science, Amsterdam.

Chan, D. S. K. (1978). *Theory and implementation of multidimensional discrete systems for signal processing.* Ph.D. Dissertation, Mass. Inst. Technology.

Chong, Y. M. (1984). Data flow chip optimizes image processing. *Computer Design*, 15 Oct., 97.

Claasen, T. A. C. M., W. F. G. Mecklenbräuker and J. B. H. Peek (1975). Quantization noise analysis for fixed point digital filters using magnitude truncation for quantization. *IEEE Trans. Ccts Syst.*, **CAS-22**, 887.

Clarke, D. W. (1982). A simple control language for microprocessors and its applications. *Proc. IFAC Congr. Theory Applic. Dig. Control*, New Delhi.

Cole, B. C. (1985). Signal processing: a big switch to digital. *Electronics*, 26 Aug., 42.

Crochiere, R. E. (1975). A new statistical approach to the coefficient wordlength problem for digital filters. *IEEE Trans. Ccts Syst.*, **CAS-22**, 190.

Crochiere, R. E. and A. V. Oppenheim (1975). Analysis of digital networks. *Proc. IEEE*, **63**, 581.

Crowell, C. D. (1985). Floating-point arithmetic with the TMS 32020. *Texas Instruments Application Report.*

Curry, E. E. (1967). The analysis of round-off and truncation errors in a hybrid control system. *IEEE Trans. Aut. Control*, **AC-12**, 601.

Cushman, R. H. (1982). ICs and semiconductors. *EDN*, 16 July, 44.

Davies, E. (1985). Sample and hold—the key to fast A to D conversion. *Electronic Engng*, Mar., 67.

Doyle, J. C. and G. Stein (1981). Multivariable feedback design: concepts for classical/modern synthesis. *IEEE Trans. Aut. Control*, **AC-26**, 4.

Eckhardt, B. (1975). On the roundoff error of a multiplier. *Arch. Elektrische Uebertragungstechnik*, **29**, 162.

Edgar, A. D. and S. C. Lee (1979). FOCUS microcomputer number system. *Comm. ACM*, **22**, 166.

Eldon, J. and G. E. Winter (1983). Floating-point chips carve out FFT systems. *Electron. Des.*, Aug., 4.

Epstein, B. (1970). *Linear Functional Analysis.* Saunders, Philadelphia.

Essig, D., C. Erskine, E. Caudel and S. Magar (1986). A second-generation digital signal processor. *IEEE Trans. Ccts Syst.*, **CAS-33**, 196.

Etzel, M. H. (1983). Logarithmic addition for digital signal processing applications. *Proc. IEEE Int. Symp. Ccts Syst.*, New York.

Evanczuk, S. (1983). Real-time OS. *Electronics*, 24 Mar., 105.

Fadden, E. J. (1984). The System 10 Plus: a major advance in scientific computing. *Proc. Conf. Peripheral Array Processors*, Boston.

Fettweis, A. (1972). On the connection between multiplier word length limitation and roundoff noise in digital filters. *IEEE Trans. Cct Theory*, **CT-19**, 486.

Fettweis, A. (1973). Roundoff noise and attenuation sensitivity in digital filters with fixed-point arithmetic. *IEEE Trans. Cct*

Theory, **CT-20**, 174.
Fettweis, A. (1974). On properties of floating-point roundoff noise. *IEEE Trans. Acoust. Speech Sig. Process.*, **ASSP-22**, 149.
Fettweis, A. (1984). Digital circuits and systems. *IEEE Trans. Cct Syst.*, **CAS-31**, 31.
Flaherty, T. J. (1985). Building blocks stack up to high performance. *Comput. Des.*, Feb, 161.
Flores, I. (1963). *The Logic of Computer Arithmetic.* Prentice-Hall, Englewood Cliffs, New Jersey.
Forsythe, W. (1983). Algorithms for digital control. *Trans. Inst. Meas. Control*, **5**, 123.
Forsythe, W. (1985). A new method for the computation of digital filter coefficients. *Simulation*, **44**, 23; **44**, 75.
Franklin, G. F. and J. D. Powell (1980). *Digital Control of Dynamic Systems.* Addison-Wesley, Reading, Massachusetts.
Frey, M. L. and F. J. Taylor (1985). A table reduction technique for logarithmically architected digital filters. *IEEE Trans. Acoust. Speech Sig. Process.*, **ASSP-33**, 718.
Fromme, G. and M. Haverland (1983). Selbsteinstellende Digitalregler im Zeitbereich. *Regelungstechnik*, **31**, 338.
Gambe, H., T. Ikezawa, N. Kobayashi, S. Sumi, T. Tsuda and S. Fujii (1983). A general purpose digital signal processor. *Proc. Eur. Conf. Cct Theory Des.*, Stuttgart. VDE-Verlag, F.R.G.
Gazsi, L. and Güllüoglu (1983). Discrete optimization in CSD code. *Proc. IEEE MELECON*, Athens.
Gear, C. W. (1984). The numerical solution of problems which may have high frequency components. In Haug, E. J. (Ed.), *Computer Aided Analysis and Optimization of Mechanical System Dynamics*, Nato ASI Series, Vol. **F9**, Springer, Berlin.
Glesner, M., H. Joepen, J. Schuck and N. Wehn (1986). Silicon compilation from HDL and similar sources. In Hartenstein (Ed.), *Advances in CAD for VLSI*, Vol. 7. North-Holland, Amsterdam.
Gold, B. and C. M. Rader (1969). *Digital Processing of Signals.* McGraw-Hill, New York.
Goodwin, G. C. (1985). Some observations on robust estimation and control. *Proc. 7th IFAC Symp. Ident. Syst. Param. Est.* York.
Gupta, A. and H. D. Toong (1983). Microprocessors—the first twelve years. *Proc. IEEE*, **71**, 1236.
Gupta, A. and H. D. Toong (1984). Microcomputers in industrial control applications. *IEEE Trans. Ind. Electron.*, **IE-31**, 2, 109.
Gupta, A. and H. D. Toong (1983). An architectural comparison of 32-bit microprocessors. *IEEE Micro*, Feb., 9.
Haberland, B. L. and S. S. Rao (1973). Discrete-time models: bilinear transform and ramp approximation equivalence. *IEEE Trans. Audio Electroacoust.*, **AU-21**, 382.
Hagiwara, Y., Y. Kita, T. Miyamoto, Y. Toba, H. Hara and T. Akazawa (1983). A single chip digital signal processor and its application to real-time speech analysis. *IEEE Trans. Acoust. Speech Sig. Process.*, **ASSP-31**, 339.
Hall, E. L., D. D. Lynch and S. J. Dwyer (1970). Generation of products and quotients using approximate binary logarithms for digital filtering applications. *IEEE Trans. Comput.*, **C-19**, 97.
Halyo, N. and G. A. McAlpine (1971). A discrete model for product quantization errors in digital filters. *IEEE Trans. Audio Electroacoust.*, **AU-19**, 255.
Hanselmann, H. (1982). Tischrechner programmiert Signalprozessor als digitalen Mehrgrössenregler. *Elektronik*, **31**, 21, 134.
Hanselmann, H. (1984). Diskretisierung kontinuierlicher Regler. *Regelungstechnik*, **32**, 326.
Hanselmann, H. (1986). Einsatz Digitaler Ein-Chip-Signalprozessoren in der Mess- und Regelungstechnik. *Bull. Schweizer Elektrotechnischer Verein* (to appear).
Hanselmann, H., R. Kasper and M. Lewe (1983). Simulation of fast digital control systems. *Proc. 1st Eur. Simulation Cong.*, Aachen, Informatik-Fachbericht 71. Springer, Berlin.
Hanselmann, H. and A. Schwarte (1985). Guide to the TMS 320 controller code generator, version 1.1. University of Paderborn, Dept. Aut. Control in Mech. Eng.
Hanselmann, H. and W. Loges (1983). Realisierung schneller digitaler Regler hoher Ordnung mit Signalprozessoren. *Regelungstechnik*, **31**, 330.
Hanselmann, H. and W. Loges (1984). Implementation of very fast state-space controllers using digital signal processors. *Proc. 9th IFAC Wld Congr.* Pergamon Press, New York.
Hartimo, I., K. Kronlöf, O. Simula and J. Skyttä (1986). DFSP: A data flow signal processor. *IEEE Trans. Comput.*, **C-35**, 23.
Hay, J. L. (1984). ESL—advanced simulation language implementation. *Proc. 84 UKSC Conf.*, Bath. Butterworths, London.
Hay, J. L. (1985). Applications of ESL. *Proc. 11th IMACS Wld Congr.*, Oslo.
Heider, G. (1982). Lct operating systems aid in component design. *Comput. Des.*, Sept.
Henrichfreise, H. (1985). Fast elastic robots: control of an elastic robot axis accounting for nonlinear drive properties. *Proc. 11th IMACS Wld Congr.*, Oslo.
Herrmann, O. E. and J. Smit (1983). A user-friendly environment to implement algorithms on single-chip digital signal processors. *Proc. EU-RASIP.* Elsevier Science, Amsterdam.
Howe, R. M. (1982). Digital simulations of transfer functions. *Proc. Summer Simulat. Conf.*, La Jolla, California.
Hwang, K. (1979). *Computer Arithmetic.* Wiley, New York.
Hwang, S. Y. (1975a). Dynamic range constraint in state-space digital filtering. *IEEE Trans. Acoust. Speech Sig. Process.*, **ASSP-23**, 591.
Hwang, S. Y. (1975b). On monotonicity of $\mathbf{L}_p$ and l_p Norms. *IEEE Trans. Acoust. Speech Sig. Process.*, **ASSP-23**, 593.
Hwang, S. Y. (1977). Minimum uncorrelated unit noise in state-space digital filtering. *IEEE Trans. Acoust. Speech Sig. Process.*, **ASSP-25**, 273.
Jacklin, S. A., J. A. Leyland and W. Warmbrodt (1985). High-speed, automatic controller design considerations for integrating array processor, multi-microprocessor, and host computer system architectures. *Am Control Conf.*, Boston, 1223.
Jackson, L. B. (1970a). On the interaction of roundoff noise and dynamic range in digital filters. *Bell Syst. Tech. J.*, **49**, 159.
Jackson, L. B. (1970b). Roundoff-noise analysis for fixed-point digital filters realized in cascade or parallel form. *IEEE Trans. Audio Electroacoust.*, **AU-18**, 107.
Jackson, L. B. (1976). Roundoff noise bounds derived from coefficient sensitivities for digital filters. *IEEE Trans. Cct Syst.*, **CAS-23**, 481.
Jackson, L. B. (1979). Limit Cycles in State-Space Structures for Digital Filters. *IEEE Trans. Cct Syst.*, **CAS-26**, 67.
Jackson, L. B., A. G. Lindgren and Y. Kim (1979). Optimal Synthesis of Second-Order State-Space Structures for Digital Filters. *IEEE Trans. Cct Syst.*, **CAS-26**, 149.
Jacquot, R. G. (1981). *Modern Digital Control Systems.* Marcel Dekker, New York.
Jaeger, R. C. (1982). Analog data acquisition technology. *IEEE Micro.* Aug., 46.
Jain, R., J. Vandewalle and H. J. de Man (1985). Efficient and accurate multiparameter analysis of linear digital filters using a multivariable feedback representation. *IEEE Trans. Cct Syst.*, **CAS-32**, 225.
Jaswa, V. C., C. E. Thomas and J. T. Pedicone (1985). CPAC—concurrent processor architecture for control. *IEEE Trans. Comput.*, **C-34**, 163.
Johnson, G. W. (1965). Upper bound on dynamic quantization error in digital control systems via the direct method of Liapunov. *IEEE Trans. Aut. Control*, **AC-10**, 439.
Johnson, G. W. (1966). Author's reply. *IEEE Trans. Aut. Control*, **AC-11**, 333.
Jover, J. M. and T. Kailath (1986). A parallel architecture for Kalman filter measurement update and parameter estimation. *Automatica*, **22**, 43.
Kaiser, J. F. (1966). *Digital Filters, System Analysis by Digital Computer.* Wiley, New York.
Källström, C. (1973). *Computing exp (A) and integral exp (As)ds.* Report 7309, Lund Inst. Technol., Div. Aut. Control.
Kanade, T. and D. Schmitz (1985). Development of CMU direct-drive arm II. *Proc. 1985 Am. Control Conf.* Boston, p. 703.
Kaneko, T. and B. Liu (1973). On local roundoff errors in floating-point arithmetic. *Jl ACM*, **20**, 391.
Katz, P. (1981). *Digital Control using Microprocessors.* Prentice-Hall, Englewood Cliffs, New Jersey.
Katzenelson, J. (1962). On errors introduced by combined sampling and quantization. *IRE Trans. Aut. Control*, **AC-72**, 58.

Kawamata, M. and T. Higuchi (1985). A unified approach to the optimal synthesis of fixed-point state-space digital filters. *IEEE Trans. Acoust. Speech Sig. Process.*, **ASSP-33**, 911.

Kerckhoffs, E. J. H., B. Dobbelaere and G. C. Vansteenkiste (1985). Some nonconventional digital computers in simulation. *Proc. 11th IMACS Wld Congr.*, Oslo.

Kingsbury, N. G. and P. J. W. Rayner (1971). Digital filtering using logarithmic arithmetic. *Electron. Lett.*, **7**, 56. Also in Swartzlander (1980).

Kleinman, D. L. and P. K. Rao (1977). Continuous-discrete gain transformation methods for linear feedback control. *Automatica*, **13**, 425.

Knowles, J. B. and E. M. Olcayto (1968). Coefficient accuracy and digital filter response. *IEEE Trans. Cct Theory*, **CT-15**, 31.

Knowles, J. B. and R. Edwards (1965a). Effect of a finite-word-length computer in a sampled-data feedback system. *Proc. IEE*, **112**, 1197.

Knowles, J. B. and R. Edwards (1965b). Finite word-length effects in multirate direct digital control systems. *Proc. IEE*, **112**, 2376.

Knowles, J. B. and R. Edwards (1966). Computational error effects in a direct digital control system. *Automatica*, **4**, 7.

Kung, S. Y. (1984). On supercomputing with systolic/wavefront array processors. *Proc. IEEE*, **72**, 867.

Kuo, B. C. (1980). *Digital Control Systems.* Holt, Rinehart and Winston, Tokyo.

Kuo, B. C., G. Singh and R. Yackel (1973). Digital approximation of continuous-data control systems by point-by-point state comparison. *Comput. Elect. Engng*, **1**, 155.

Kuo, B. C. and D. W. Peterson (1973). Optimal discretization of continuous-data control system. *Automatica*, **9**, 125.

Kwakernaak, H. and R. Sivan (1972). *Linear Optimal Control systems.* Wiley, New York.

Lack, G. N. T. (1966). Comments on "Upper bound on dynamic quantization error in digital control systems via the direct method of Liapunov". *IEEE Trans. Aut. Control*, **AC-11**, 331.

Lang, J. H. (1984). On the design of a special-purpose digital control processor. *IEEE Trans. Aut. Control*, **AC-29**, 195.

Lee, S. C. and A. D. Edgar (1977). The focus number system. *IEEE Trans. Comput.*, **C-26**, 1167.

Leonhard, W. (1986). Microcomputer control of high dynamic performance ac-drives—a survey. *Automatica*, **22**, 1.

Liu, B. and T. Kaneko (1969). Error analysis of digital filters realized with floating-point arithmetic. *Proc. IEEE*, **57**, 1735.

Loges, W. (1983). Regelsysteme höherer Ordnung mit dem Signalprozessor TMS 320. *Elektronik*, **32**, 25, 53.

Loges, W. (1984). Codegenerator erstellt Reglerprogramm für den TMS 320. *Elektronik*, **33**, 22, 154.

Loges, W. (1985). *Realisierung schneller digitaler Regler hoher Ordnung mit Signalprozessoren.* Doctoral dissertation, University of Paderborn, Dept. Aut. Control in Mech. Eng.; also VDI Verlag, Düsseldorf.

Long, J. L. and T. N. Trick (1973). An absolute bound on limit cycles due to roundoff errors in digital filters. *IEEE Trans. Audio Electroacoust.*, **AU-21**, 27.

MacSorley, O. L. (1961). High-speed arithmetic in binary computers. *IRE Proc.*, **49**, 67. Also in Swartzlander, E. E. (Ed.) (1980), *Computer Arithmetic.* Dowden, Hutchinson & Ross, Stroudsberg, Pennsylvania.

Magar, S., E. Caudel, D. Essig and C. Erskine (1985). Digital signal processor borrows from μP to step up performance. *Electron Des.*, 21 Feb., 175.

Magar, S., S. J. Robertson and W. Gass (1985). Interface arrangement suits digital processor to multiprocessing. *Electron. Des.*, 7 March, 189.

Marrin, K. (1986). Six DSP processors tackle high-end signal-processing applications. *Comput. Des.*, 1 March, 21.

Marrin, K. E. (1985). VLSI and software move DSP techniques into mainstream. *Comput. Des.*, 15 Sept., 69.

McDonough, K., E. Caudel, S. Magar and A. Leigh (1982). Microcomputer with 32-bit arithmetic does high-precision number crunching. *Electronics*, Feb., 105.

Meisinger, R. and B. Lange (1976). Berücksichtigung der Rechnertotzeit beim Entwurf eines diskreten Regelungs- und Beobachtungssystems. *Regelungstechnik*, **24**, 232.

Middleton, R. H. and G. C. Goodwin (1985). Improved finite word length characteristics in digital control using delta operators. *Dept. Electr. Comp. Eng. Report*, Univ. of Newcastle, Australia.

Miller, D. F. (1985). Multivariable linear digital control via state-space output matching. *Opt. Control Applic. Meth.*, **6**, 13.

Mills, W. L., C. T. Mullis and R. A. Roberts (1978). Digital filter realizations without overflow oscillations. *Proc. IEEE Int. Conf. Acoust. Speech Sig. Process.*, Tulsa, Oklahoma.

Mills, W. L., C. T. Mullis and R. A. Roberts (1981). Low roundoff noise and normal realizations of fixed point IIR digital filters. *IEEE Trans. Acoust., Speech Sig. Process.*, **ASSP-29**, 893.

Mintzer, F., K. Davies, A. Peled and F. N. Ris (1983). The real-time signal processor. *IEEE Trans. Acoust. Speech Sig. Process.*, **ASSP-31**, 83.

Mita, T. (1985). Optimal digital feedback control systems counting computation time of control laws. *IEEE Trans. Aut. Control*, **AC-30**, 542.

Mitchell, E. E. and R. Demoyer (1985). A versatile digital controller algorithm incorporating a state observer and state feedback. *IEEE Trans. Ind. Electron.*, **IE-32**, 78.

Mitra, S. K., K. Hirano and H. Sakaguchi (1974). A simple method of computing the input quantization and multiplication roundoff errors in a digital filter. *IEEE Trans. Acoust. Speech Sig. Process.*, **ASSP-22**, 326.

Moroney, P. (1983). *Issues in the Implementation of Digital Feedback Compensators.* MIT Press, Cambridge, Massachusetts.

Moroney, P., A. S. Willsky and P. K. Houpt (1980). The digital implementation of control compensators: the coefficient word-length issue. *IEEE Trans. Aut. Control*, **AC-25**, 621.

Moroney, P., A. S. Willsky and P. K. Houpt (1981). Architectural issues in the implementation of digital compensators. *Proc. 8th IFAC Wld Congr.*, Kyoto.

Moroney, P., A. S. Willsky and P. K. Houpt (1983). Roundoff noise and scaling in the digital implementation of control compensators. *IEEE Trans. Acoust. Speech Sig. Process.*, **ASSP-31**, 1464.

Mullis, C. T. and R. A. Roberts (1976). Synthesis of minimum roundoff noise fixed point digital filters. *IEEE Trans. Ccts Syst.*, **CAS-23**, 551.

Mullis, C. T. and R. A. Roberts (1982). An interpretation of error spectrum shaping in digital filters. *IEEE Trans. Acoust. Speech Sig. Process.*, **ASSP-30**, 1013.

Mullis, C. T. and R. A. Roberts (1984). Digital processing structures for VLSI implementation. In Cappello, P. R. (Ed.), *VLSI Signal Processing.* IEEE Press, New York.

Nagle, H. T. and V. P. Nelson (1981). Digital filter implementation on 16-bit microcomputers. *IEEE Micro*, 23 Feb.

Neuman, C. P. and C. S. Baradello (1979). Digital transfer functions for microcomputer control. *IEEE Trans. Syst. Man Cybern.*, **SMC-9**, 856.

Nishimura, S., K. Hirano and R. N. Pal (1981). A new class of very low sensitivity and low roundoff noise recursive digital filter structures. *IEEE Trans. Ccts. Syst.*, **CAS-28**, 1152.

Nishitani, T., R. Maruta, Y. Kawakami and H. Goto (1981). A single-chip digital signal processor for telecommunication applications. *IEEE Jl Solid State Ccts*, **SC-16**, 372.

Orlandi, G and G. Martinelli (1984). Low-sensitivity recursive digital filters obtained via the delay replacement. *IEEE Trans. Ccts. Syst.*, **CAS-31**, 654.

Oppenheim, A. V. and A. S. Willsky (1983). *Signals and Systems.* Prentice-Hall, Englewood Cliffs, New Jersey.

Oppenheim, A. V. and R. W. Schafer (1975). *Digital Signal Processing.* Prentice-Hall, Englewood Cliffs, New Jersey.

Patney, R. K. and S. C. Dutta Roy (1980). A different look at roundoff noise in digital filters. *IEEE Trans. Ccts Syst.*, **CAS-27**, 59.

Pei, S. C. (1985). Comments on "Properties of the *Qn*-matrix in bilinear transformation". *Proc. IEEE*, **73**, 841.

Pei, S. C. and K. C. Ho (1984). Comments on "Adaptive digital control implemented using residue number systems". *IEEE Trans. Aut. Control*, **AC-29**, 863.

Peled, A. and B. Liu (1976). *Digital Signal Processing.* Wiley, New York.

Peled, U. and J. D. Powell (1978). The effect of prefilter design on sample rate selection in digital flight control systems. *Proc.*

AIAA Guid. Control Conf., Palo Alto, California.
Phillips, C. L. (1980). Using simulation to calculate floating-point quantization errors. *Simulation*, June, 207.
Phillips, C. L. and H. T. Nagle (1984). *Digital Control Systems Analysis and Design*. Prentice-Hall, Englewood-Cliffs, New Jersey.
Pickvance, R. (1985). A single chip digital signal processor. *Electron. Engng*, Feb., 53; March, 55; Apr., 87.
Pope, S., J. Rabaey and R. W. Brodersen (1984). Automated design of signal processors using macrocells. In Cappello, P. R. (Ed.), *VLSI Signal Processing*. IEEE Press, New York.
Pountain, D. (1985). Multitasking FORTH. *Byte*, March, 363.
Powers, W. F. (1985). Computer tools for modern control systems design. *IEEE Control Syst. Mag.*, Feb., 14.
Quarmby, D. J. (1984). *Signal Processor Chips*. Granada, London.
Quong, D. and R. Perlman (1984). Single-chip accelerators speed floating-point and binary computations. *Electron. Des.*, 15 Nov., 246.
Rabaey, J., S. Pope and R. W. Brodersen (1987). An integrated automated layout generation system for DSP circuits. *J. Comput. Aided Des.* (to appear).
Rattan, K. S. (1981). Digital redesign of existing multiloop continuous control systems. *Proc. Jt Aut. Control Conf.*, Charlottesville, Virginia.
Rattan, K. S. (1982). Digitalizing existing continuous-data control systems via "continous frequency matching". *Proc. IFAC Symp. Theory Applic. Dig. Control*, New Delhi.
Rattan, K. S. (1984). Digitalization of existing continuous control systems. *IEEE Trans. Aut. Control*, **AC-29**, 282.
Rattan, K. S. and H. H. Yeh (1978). Discretizing continuous-data control systems. *Comput.-Aided Des.*, **10**, 299.
Ready, J. F. (1984). Operating systems conform to application needs. *Mini-Micro Systems*, Dec., 137.
Rink, R. E. and H. Y. Chong (1979a). Performance of state regulator systems with floating-point computation. *IEEE Trans. Aut. Control*, **AC-24**, 411.
Rink, R. E. and H. Y. Chong (1979b). Covariance equation for a floating-point regulator system. *IEEE Trans. Aut. Control*, **AC-24**, 980.
Rojek, P. and W. Wetzel (1984). Mehrgrössenregelung mit Signalprozessoren. *Elektronik*, **33**, 16; 109.
Rubinfield, L. P. (1975). A proof of the modified Booth's algorithm for multiplication. *IEEE Trans. Comput.*, **C-24**, 1014.
Sandberg, I. W. (1967). Floating-point-roundoff accumulation in digital-filter realizations. *Bell Syst. Tech. J.*, **46**, 1175.
Sandberg, I. W. and J. F. Kaiser (1972). A bound on limit cycles in fixed-point implementations of digital filters. *IEEE Trans. Audio Electroacoust.*, **AU-20**, 110.
Sasahara, H., M. Kawamata and T. Higuchi (1984). Design of microprocessor-based LQG control systems with minimum quantization error. *Proc. IECON '84*, Tokyo.
Schafer, R., R. M. Mersereau and T. P. Barnwell (1984). Software package brings filter design to PCs. *Comput. Des.*, Nov., 119.
Scharf, L. L. and S. Sigurdsson (1984). Fixed point implementation of fast Kalman predictors. *IEEE Trans. Aut. Control*, **AC-29**, 850.
Schittke, H. J. and R. Dettinger (1975). Simulation von linearen zeitinvarianten Systemen bei stückweise linearem Verlauf des Steuervektors. *Regelungstechnik*, **23**, 422; **24**, 27.
Schmidt, L. A. (1978). Designing programmable digital filters for LSI implementation. *Hewlett-Packard J.*, **29**, 13, 15.
Schumacher, W. and W. Leonhard (1983). Transistor-Fed AC-servo drive with microprocessor control. *Proc. Int. Power Electron. Conf.*, Tokyo.
Shah, S. C., M. A. Floyd and L. L. Lehman (1985). MATRIX$_x$: control design and model building CAE capability. In Jamshidi, M. and C. J. Herget (Eds), *Advances in Computer-Aided Control Systems Engineering*. North-Holland, Amsterdam.
Shaw, R. F. (1950). Arithmetic operations in a binary computer. *Rev. Sci. Instrum.*, **21**, 687. Also in Swartzlander, E. E. (Ed.) (1980), *Computer Arithmetic*. Dowden, Hutchinson & Ross, Stroudsberg, Pennsylvania.
Shieh, L. S., Y. F. Chang and R. E. Yates (1982). Model simplification and digital design of multivariable sampled-data control systems via a dominant-data matching method. *Proc. IFAC Symp. Theory Applic. Dig. Control*, New Delhi.
Shoreys, F. (1982). New approach to high-speed high-resolution analogue-to-digital conversion. *IEE Electron. Power*, Feb., 175.
Simmers, C. and D. Arnett (1985). Specialized I/O and high-speed CPU yields efficient microcontroller for automotive applications. *IEEE Trans. Ind. Electron.*, **IE-32**, 278.
Singh, G., B. C. Kuo and R. A. Yackel (1974). Digital approximation by point-by-point state matching with high-order holds. *Int. J. Control*, **20**, 81.
Sjoding, T. W. (1973). Noise variance for rounded two's complement product quantization. *IEEE Trans. Audio Electroacoust.*, **AU-21**, 378.
Skyttä, J., O. Hyvärinen, I. Hartimo and O. Simula (1983). Experimental signal processing and development system. *Proc. Eur. Conf. Cct Theory Des.*, Stuttgart.
Slaughter, J. B. (1964). Quantization errors in digital control systems. *IEEE Trans. Aut. Control*, **AC-9**, 70.
Slivinski, Ch. and J. Borninski (1985). Control system compensation and implementation with the TMS32010. *Texas Instruments Application Report.*
Smith, J. M. (1977). *Mathematical Modelling and Digital Simulation for Engineers and Scientists*. Wiley, New York.
Spang, H. A. (1985). Experience and future needs in computer-aided control design. *IEEE Control Syst. Mag.*, Feb., 18.
Sripad, A. B. and D. L. Snyder (1977). A necessary and sufficient condition for quantization errors to be uniform and white. *IEEE Trans. Acoust., Speech Sig. Process.*, **ASSP-25**, 442.
Srodawa, R. J., R. E. Gach and A. Glicker (1985). Preliminary experience with the automatic generation of production-quality code for the Ford/Intel 8061 microprocessor. *IEEE Trans. Ind. Electron.*, **IE-32**, 318.
Steinlechner, S., E. Auer and E. Lueder (1983). A fast digital signal processor without multipliers. *Proc. Conf. Cct Theory ECCTD*, Stuttgart.
Stirling, R. (1983). Simulation of a digital aircraft flight control system. *Simulation*, May, 171.
Strejc, V. (1981). *State Space Theory of Discrete Linear Control*. Wiley, New York.
Sutherland, H. A. and K. L. Sonin (1985). Control engineers workbench—a methodology for microcomputer implementation of controls. *IEEE Control Syst. Mag.*, Feb., 22.
Swartzlander, E. E. (Ed.) (1980). *Computer Arithmetic*. Dowden, Hutchinson & Ross, Stroudsburg, Pennsylvania.
Swartzlander, E. E. and A. G. Alexopoulos (1975). The sign/logarithm number system. *IEEE Trans. Comput.*, **C-24**, 1238. Also in Swartzlander, E. E. (Ed) (1980), *Computer Arithmetic*. Dowden, Hutchinson & Ross, Stroudsberg, Pennsylvania.
Tabak, D. and G. J. Lipovski (1980). MOVE architecture in digital controllers. *IEEE Trans. Comput.*, **C-29**, 180.
Taetow, W. (1984). CMOS Bausteine für mikroprogrammierbare Signalprozesoren. *Elektronik*, **33**, 22, 136, 23, 138.
Tan, C. and B. C. McInnis (1982). Adaptive digital control implemented using residue number systems. *IEEE Trans. Aut. Control*, **AC-27**, 449, 499.
Taylor, R. (1984). Signal processing with occam and the transputer. *IEE Proc.*, **131**, 610.
Toong, H. D. and A. Gupta (1982). Evaluation kernels for microprocessor performance analyses. *Perform. Evaluat.*, **2**, 1.
Vaidyanathan, P. P. (1985). On error-spectrum shaping in state-space digital filters. *IEEE Trans. Ccts Syst.*, **CAS-32**, 88.
Van Wingerden, A. J. M. and W. L. de Koning (1984). The influence of finite word length on digital optimal control. *IEEE Trans. Aut. Control*, **AC-29**, 385.
Wallich, P. (1985). Toward simpler, faster computers. *IEEE Spectrum*, Aug., 38.
Walrath, C. D. (1984). Adaptive bearing friction compensation based on recent knowledge of dynamic friction. *Autmatica*, **20**, 717.
Waser, Sh. and M. J. Flynn (1982). *Introduction to Arithmetic for Digital Systems Designers*. CBS College Publishing, New York.
Weinstein, C. and A. V. Oppenheim (1969). A comparison of roundoff noise in floating point and fixed point digital filter realizations. *Proc. IEEE*, **57**, 1181.
Widrow, B. (1956). A study of rough amplitude quantization by means of Nyquist sampling theory. *IRE Trans. Cct Theory,*

PGCT-3, 266.
Widrow, B. (1961). Statistical analysis of amplitude-quantized sampled-data systems. *Trans. AIEE*, **79**, 555.
Widrow, B. and E. Walach (1983). Adaptive signal processing for adaptive control. *Proc. IFAC Workshop Adapt. Syst. Control Sig. Process.*, San Francisco.
Williamson, D. (1985). Finite wordlength design of digital Kalman filters for state estimation. *IEEE Trans. Aut. Control*, **AC-30**, 930.
Willsky, A. S. (1979). *Digital Signal Processing and Control and Estimation Theory*. MIT Press, Cambridge, Massachusetts.
Windsor, W. A. (1985). IEEE floating point chips implement DSP architectures. *Comput. Des.*, Jan., 165.
Wittenmark, B. (1985). Sampling of a system with a time delay. *IEEE Trans. Aut. Control*, **AC-30**, 507.
Yackel, R. A., B. C. Kuo and G. Singh (1974). Digital redesign of continuous systems by matching of states at multiple sampling periods. *Automatica*, **10**, 105.
Yekutiel, O. (1980). A reduced-delay sampled-data hold. *IEEE Trans. Auto. Control*, **AC-25**, 847.
Zimmerman, B. G. (1983). MODEL S, a sampled-data simulation language. *Simulation*, May, 183.

The Programming Language DSPL

a problem oriented approach for
digital signal processing using DSP

Albert Schwarte and Herbert Hanselmann

dSPACE digital signal processing and control engineering GmbH
Paderborn, West Germany

Abstract

Digital signal processors (DSP) are increasingly used in many application fields like motion control systems and power conversion systems due to their impressive computational performance. However, appropriate tools for programming such devices are still lacking. Therefore DSPs are mainly programmed using assembly language. The high level language DSPL introduced here has been developed with the typical application fields in mind. Characteristic elements of DSPs have also been regarded. This results in compilers capable of generating extremely efficient code. Furthermore DSPL's automatic scaling features simplify programming of applications for DSP with fixed-point arithmetic.

Introduction

For a few years now digital signal processors have been available as very powerful devices for computational intensive applications possibly demanding real-time performance. DSPs have been developed primarily for signal processing applications like filtering, speech analysis, data communication and the like. Comparing the mathematical algorithms used in these fields with the algorithms used in modern multi-variable control theory shows however, that both application fields have to deal with many common problems. Thus DSPs are increasingly used for the implementation of complex control systems and other industrial applications like motion control systems, power conversion systems and hardware-in-the-loop simulation systems.

DSPs are a very special class of microprocessors. They typically contain hardware optimized to carry out multiplications and accumulations. Most DSPs are able to perform a multiplication within a single machine cycle and perform the accumulations of products in parallel. This leads to extremely high throughput for the computation of scalar products, a central element of signal processing algorithms. Another feature that distinguishes DSPs from conventional microprocessors is the Harvard-architecture used by many such devices. They usually have several separate memory blocks connected to the CPU core with multiple data and address busses. These data paths can be used in parallel so that several operands can be transferred at the same time.

Utilizing such specific DSP elements is nearly impossible with conventional high level programming languages like C or Pascal, because such languages have no appropriate constructs which allow a compiler writer to make use of these elements. Another problem not addressed by these languages is the lack of an appropriate data type for DSPs using fixed-point arithmetic. Fixed-point arithmetic is however still used by most DSPs, and especially the low-cost ones embedded in products manufactured in large quantities.

Special features of DSPL

Nevertheless most of the few high level language compilers available represent a more or less comprehensive subset of the C programming

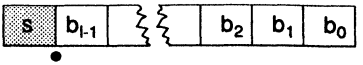
Reprinted, with permission, from *PCIM*, June 25 – 28, 1990.

language. The Digital Signal Processing Language (DSPL) introduced here follows a more problem oriented approach. It has been developed with the intention to be particularly useful for the special application fields of digital signal processing using DSPs for the implementation. Especially for fixed-point DSPs DSPL provides extensive support by defining an appropriate data type and automatic scaling features.

DSPL data formats

Standard DSPs like the first and second generation TMS 320 series use a 16 bit fixed-point data format. Using this format for the conventional integer arithmetic leads to a quantization of 8 bit for data and coefficients in order to avoid overflows when computing a product. Accumulation of products as required for a scalar product requires additional scaling, so that the worst case sum of the partial products does not overflow the integer value range. Using only 8 bits for the representation of data and coefficients however results in a very small number range with low resolution. This is not acceptable for most industrial applications. As the same problem arises for conventional microprocessors system designers have developed algorithms to perform floating-point arithmetic with fixed-point processors. With the aid of floating-point arithmetic an arbitrary number range with arbitrary resolution can be realized according to the number format selected, but at the cost of largely increased execution time. This is also possible for DSPs, of course, but using such a floating-point software package decreases the DSP's performance so far that conventional microprocessors combined with hardware floating-point coprocessors seem more attractive, at least for applications where the price of the processors is not a primary issue.

DSPL follows a third way which can provide a good compromise for most applications. It couples the speed of integer arithmetic with a resolution of 16 bit for the above mentioned processors. To achieve this DSPL provides the fractional data format. Data are interpreted as two's complement numbers having the binary point directly right to the sign bit (MSB) which leads to a value range of -1.0 .. 0.99996.. .



Obviously the multiplication of fractional numbers can never overflow the fractional value range and can be implemented easily as most DSPs provide an accumulation register at least twice as long as the data format used, e.g. 32 bit for the TMS 320 series. The fractional format allows to use all 16 bit for the representation of data which results in a quantization good enough for most applications. Only when accumulating fractional numbers the result can overflow the value range. On the one hand this can be avoided by properly scaling data during preparation of the implementation, and on the other hand by using DSPL's automatic scaling features for the computation of scalar products. As the fractional data format is just another interpretation of the binary data fractional arithmetic except division can be implemented on the machine instruction level which results in the same execution speed as integer arithmetic. Fractional numbers are the main vehicle for carrying out computations in DSPL. They are supported by the compilers not only for the computation of scalar products but also for any other basic arithmetic expression including division.

Besides the fractional format a conventional integer data type and boolean data are also supported. In addition to the basic operations DSPL allows bitwise handling of integer variables with logical operators. This is especially useful for manipulating hardware devices on the bit level, particularly because variables can be allocated at arbitrary physical addresses. Boolean variables can be used in arbitrary expressions as well. They are mainly useful for controlling program flow in conjunction with if-statements.

Scalar product computation

Many digital signal processing algorithms consist mainly of the computation of scalar products. FIR filters and difference equations of controllers or IIR filters provide good examples.

$$r = c_1 d_1 + c_2 d_2 + \ldots + c_n d_n$$

Implementing scalar products on processors with fixed-point arithmetic is a cumbersome and error-prone task due to the scaling requirements. DSPL supports the implementation of scalar products by providing the necessary constructs on the language level including automatic scaling for products of a coefficient vector and a variable vector. Scalar product scaling guarantees that

- overflows can be detected and handled appropriately by saturation conditions simulated in software
- coefficients outside the fractional value range can be realized
- coefficient scaling can be performed automatically by the compiler.

Scaling of all c_i is performed completely at compile time. Only the necessary rescaling operations for the final result (r) need to be done at runtime. Rescaling is implemented by optimized code constructs depending on the actual data. Within scalar products even coefficients outside the fractional number range can be realized with special code constructs. If scalar product scaling is performed automatically by the DSPL compiler a worst case scaling is performed. Maximum scaling values can optionally be specified by the user in case they are already known from simulation or measurements, for example. Scalar product scaling can also be completely disabled. The code necessary for rescaling can automatically include instructions to test for overflows of the scalar product result. Saturation conditions can then be simulated by software upon request. A special form of the scalar product statement allows the implementation of a FIR filter with a single DSPL statement. In this case the update of the variable vector is performed in parallel to the computation of the filter taps.

High level language compilers usually rely on library routines for the computations of scalar products, which simply execute a loop for all elements of the vectors involved to compute the sum of the partial products. However, this kind of computation is very inefficient, particularly for control algorithms where often sparse coefficient matrices have to be multiplied by variable vectors. This leads to the problem of loading the processor with unnecessary code for multiplying zeroes. Using appropriate transformations the number of non-zero coefficients can be minimized. DSPL does never use library routines but generates the appropriate code in-line depending on the actual data. The code is extremely efficient because every information the compiler needs for code generation is already known at compile time. Not a single instruction is wasted to perform address computations or adjust loop counters at runtime. Immediate instructions can often be used to realize small coefficients which leads to very economical use of data memory, a very scarce resource on some DSPs.

Block moves of data

Many DSPs contain special hardware provisions or at least efficient machine instructions to perform moving a block of data in memory. Block moves are required by many signal processing algorithms to implement the z^{-1} operation or to move the data samples through a filter. Such special elements can only be utilized by a compiler if an appropriate language construct is defined. DSPL provides the update-statement for this purpose. It allows to copy a data vector to a second one. Because the size of the vectors is already known during compile time code can be generated code without containing time consuming instructions for address computations.

Realization of sampling systems

Digital signal processing systems often require the algorithm to be carried out with a defined sampling period. DSPL provides an appropriate statement which allows the specification of the required sampling period. The compiler generates appropriate code to realize the sampling clock based on macros adaptable to the target hardware. Usually a timer capable of generating interrupts will be used for this purpose. In case a hardware system contains several timers with

interrupt capabilities even multi-rate systems can easily be implemented.

DSPL language constructs

The following tables provide a summary of the declarations and statements available in DSPL. Short comments will describe the meaning of each element.

Declaration	Purpose
TYPE	declaration of fractional data representation details
fractional, integer, boolean	scalar data types, fractional data can also be declared as vectors, constants and variables possible
SCPTYPE	type declaration used for defining details of scalar product computations like automatic scaling and saturation handling
SCALABLE	attribute of a fractional constant, allows the constant to be scaled by the compiler
ALTERABLE	attribute of a fractional constant, allows a scalable constant to be included in scalar product computation, such a constant may be altered during runtime as required by adaptive systems
AT	address clause, allows to specify the physical address where the declared object shall be allocated
INPUT / OUTPUT	instructs the compiler to associate the declared variable as with a physical input or output channel
EXTERNAL	allows the declaration of formal procedure headers, external procedures must be implemented in assembly language
INTERRUPT	instructs the compiler to associate this name with an interrupt source
RENAME	declares an alias name for a component of a fractional vector

Table 1 : Declarations provided by DSPL

Statement	Purpose
BEGIN	start of executable program body
ON ident DO .. END INTERRUPT	surrounds the interrupt service routine for an identifier declared as an interrupt source, any number of interrupt-statements are possible

EVERY time DO .. END EVERY	surrounds the block of statements to be executed with regular time intervals, the time specified represents the sampling period of sampled data systems
ACCUMULATE SCALPRO (ident) .. END ACCUMULATE	a complete scalar product with an arbitrary number of partial products is accumulated, the identifier references a scalar product type declaration
ACCUMULATE PRESCALPRO (ident) .. END ACCUMULATE	same as before except that the accumulation register is pre-loaded with a full accumulator-length value
ACCUMULATE SCALPRO (ident) AND UPDATE ident .. END ACCUMULATE	special form of scalar product accumulation, allows efficient computation of FIR filter
INPUT	a scalar or a vector of input variables is read from an I/O channel
OUTPUT	a scalar or vector of output variables is written to an I/O channel
UPDATE	copies a variable vector to a second one
assignment	the assignment statement allows the computation of arbitrarily complex arithmetic expressions.
ABS * / + - = /= < <= >= >	operators defined for fractional operands
ABS NOT * / MOD + - = /= < <= >= > AND OR XOR	operators defined for integer operands
NOT AND OR XOR	operators defined for boolean operands

IF condition THEN
ELSIF condition THEN
ELSE
END IF
	the if-statement allows to control program flow, any number of ELSIF parts are allowed, the ELSIF and ELSE parts are optional

LOOP .. EXIT .. END LOOP	the loop-statement in conjunction with the exit-statement allows the implementation of any kind of program loops
FOR ident IN range LOOP .. END LOOP	the for-statement allows the implementation of loops with a determined number of repetitions, up-counting and down-counting loops are possible

procedure call	allows the call of external procedures defined in the declarative section, actual parameters must be specified according to the formal procedure header declaration
in-line assembler	assembly language statements may be inserted anywhere, access to DSPL variables by name is supported

Table 2 : Statements provided by DSPL

Hardware independence

A DSPL program is nearly independent from the target hardware system. Each DSPL compiler can support arbitrary hardware environments surrounding a particular target DSP. This great flexibility is possible because every DSPL program is augmented by an environment description. This description instructs the compiler which address ranges it may use for program and data allocation, for example. It also contains the necessary connections between logical input and output variables of the DSPL program and the physical I/O channels. DSPL compilers are open-ended with respect to all the language constructs depending on hardware characteristics. They use macros for the implementation of input and output and for the realization of the sampling clock for example. These macros can easily be adapted to any target hardware system by the user, which needs to be done only once.

The following table describes the information contained in the environment description valid for the DSPL compiler for TMS 320C25 DSPs.

Declaration	Purpose
PROCESSOR IS "TMS 320C25"	declares the target processor, used by the compiler for consistency check
PROGRAM SPACE OFF CHIP IS ...	declaration of memory section available for program code allocation
DATA SPACE ON CHIP IS ... DATA SPACE OFF CHIP IS ...	declaration of memory sections available for data allocation
STACK SPACE IS ...	declaration of memory section available for stack allocation
CYCLE TIME IS ...	declaration of basic machine cycle, used for the computation of execution time statistics
PROGRAM MEMORY WAIT STATE IS ...	number of wait states required by the target hardware when accessing external program memory, used for the computation of execution time statistics
DATA MEMORY WAIT STATE IS ...	number of wait states required by the target hardware when accessing external data memory, used for the computation of execution time statistics

INTERRUPT ident IS VECTOR ...	declares the connection between the DSPL name of an interrupt source and an actual hardware interrupt
INPUT SPECIFICATION IS ident IS CHANNEL number USING macro .. SEQUENTIAL ident IS CHANNEL number USING macro .. END INPUT	declares the connection between the DSPL name of an input variable and a physical input channel, for each single input channel an appropriate macro can be used, optionally sequential inputs can be used in cases the target hardware prescribes a particular sequence for reading input channels
OUTPUT SPECIFICATION IS ident IS CHANNEL number USING macro .. SEQUENTIAL ident IS CHANNEL number USING macro .. END OUTPUT	declares the connection between the DSPL name of an output variable and a physical output channel, for each single output channel an appropriate macro can be used, optionally sequential outputs can be used in cases the target hardware prescribes a particular sequence for writing output channels

Table 3 : Elements of the environment description

Compiler output

A DSPL compiler generates completely documented assembly language source files which a user might optionally try to optimize. After assembling the program it can be downloaded to the target hardware and is ready for execution. Complete statistical information is also generated. This includes a detailed cross-reference listing showing allocation information for code and data sections. More interesting however is that the compiler also computes execution time statistics as far as possible. The cross-reference listing will contain information about the execution time requirements of the block-statements and compute the processor load based on the requested sampling rates. The assembly language source listing will contain information about the machine cycles used by the code generated for each single DSPL statement. These statistics will even regard such issues as the influence of wait-states required by the target hardware for the access to different memory sections. In case of programming errors the compilers generate a source listing with interspersed error messages giving detailed information about the errors detected.

Depending on the program compiled the DSPL compilers compile from several hundred to several thousand lines of code per minute on typical PCs.

Development system

Currently DSPL compilers for the TMS320C25 DSP and the TMS 310C1X DSP family are available. Although they can be used stand-alone as powerful development tools they can also be used in conjunction with a complete development system primarily designed for the realization of control systems. This development system consists of additional software and hardware components using PC-AT class machines as host. The IMPEX software supports all the necessary steps for the preparation of linear multi-variable control systems prior to the implementation. Starting from differential or difference equations IMPEX supports discretization, scaling, structure transformation, simulation of closed loop systems including effects of DSP arithmetic and A/D and D/A converters and the generation of the appropriate DSPL program. On the DSPL level any non-linear extensions can be added to the pro-

gram. This can be supported by the NMAC tool which can generate optimized table-lookup based external DSPL procedures for the implementation of arbitrary one-dimensional non-linear functions. After assembling the assembly language source file resulting from the DSPL compilation the object code can be down-loaded to the target hardware where it can be examined with a powerful real-time TRACE module. This module works on the system level rather than on the machine instruction level and is capable of displaying the time response of arbitrary variables. Sophisticated hardware systems built around the TMS 320 family DSPs, including the new TMS 320C30 floating-point DSP which is programmed in C rather than DSPL, augmented by powerful peripheral boards for analog and digital I/O and incremental encoder interfaces support the automatic implementation of standard applications often within minutes by providing completely software controlled board setups, for example.

Examples and applications

A large number of applications have already been realized using DSPL as the programming language. Some examples are described below in order to give an estimate about the computational performance of DSPs and of the quality of the code generated by the DSPL compilers. Impressive sampling rates can be achieved even for very complex applications.

The first example regards a 3rd order PD controller with notch filter as described by the following equations.

$$x_k = \begin{pmatrix} 0.333333 & 0.0 & 0.0 \\ 0.0 & 0.383240 & 0.252007 \\ 0.0 & -0.518211 & 0.383240 \end{pmatrix} x_{k-1} + \begin{pmatrix} -0.666667 \\ 0.473315 \\ 0.587474 \end{pmatrix} u_{k-1}$$

$$y_k = (-16.723549 \quad -13.152899 \quad 0.0)\ x_k \quad -6.098620\ u_k$$

Assuming that all state variables are properly scaled for the fractional number range so that no overflow test and saturation handling is required for the states, and that overflow test and saturation handling are included for the output by using scalar product scaling, a TMS 320C25 DSP can execute the code generated by the DSPL25 compiler within 7.3 μs. The same program compiled with the DSPL1X compiler can be executed within 10.4 μs by a TMS 320E14 DSP. This does not include time required for i/o and timer interrupt processing. The corresponding DSPL program and excerpts from the compiler generated assembly language source are presented below. The statistical information computed by the compiler is also presented.

The second example represents a 9th order state controller with Kalman filter having 2 inputs and one output. The controller was designed for a disk drive (computer peripheral). As this controller includes an integrator the corresponding state variable is computed with saturation using scalar product scaling. Otherwise the same assumptions apply as given above. A TMS 320C25 DSP can execute the necessary code within 19 μs. The execution time for a TMS 320E14 DSP is 27.5 μs.

Other applications implemented with DSPL include the following (sampling rates are given for a TMS 320C25).

Compliant articulated robot with electrical drives: Linear vibration damping / tracking controller with 10 sensors, 3 motors, 9 reference inputs, running at 20 kHz.

High-acceleration gantry type robot with hydraulic drives: Vibration damping / tracking controller of order 10 (including Kalman filter and non-linear compensation for hydraulic effects) for each single axis, with 1 sensor (position encoder), 1 motor and 3 reference inputs, running at 10 kHz. Several axes can be served by a single DSP.

Kalman-filter-based track following control (see second example above).

Notch-filter-based controller of 11th order for the same application runs at > 30kHz.

Vehicle control: Various active suspension controllers of up to 40th order running with sampling rates in the kHz range.

Hardware-in-the-loop simulation: Hydraulic cylinder for active vehicle suspension under test and actuating cylinder simulating the stress and motion, both given in hardware. The DSP hardware system does the rest, i.e. controls the suspension and actuating cylinder, simulates wheel and car body dynamics, and performs the noise filtering for road surface simulation, all at 14 kHz.

Anti-skid-braking (ABS) hardware-in-the-loop simulation: Four-wheeled non-linear vehicle model of 18th order (11 mechanical degrees of freedom) running at 6 kHz on a TMS 320C25. Used to test and optimize ABS in the lab.

Simplified proportional-differential control and plant identification: Just to show a mixture of DSPL constructs in an application program. The sampling rate is > 20 kHz for a TMS 320C25. The listings below show the DSPL program, the associated environment description, the statistical information and excerpts from the code generated by the DSPL25 compiler.

```
system specification controller_gain_ident is
type fractional is
  fix'(bits => 16, fraction => 15, representation => twoscomplement);
scptype state1 is
  fix'(acculength => 32, round => on, scale => on, saturation => on);
scptype del is
  fix'(acculength => 32, round => on, scale => off, saturation => off);
scptype out1 is
  fix'(acculength => 32, round => on, scale => common, saturation => on);
a1 : scalable constant vector (1) of fractional := (0.333);
b1 : scalable constant vector (2) of fractional := (0.330, -0.330);
c1 : scalable constant vector (1) of fractional := (-14.141);
d1 : scalable constant vector (2) of fractional := (7.699, -7.699);
xk  : vector (1) of fractional;
xk1 : vector (1) of fractional;
u   : vector (2) of fractional;
input is u;
y   : vector (1) of fractional;
output is y;
temp1 : rawaccumulator;
r_coeff : scalable constant vector (1) of fractional := (17.2405);
rk_del_coeff : scalable constant vector (3) of fractional := (0, 0, 1);
cnt : integer;
lk : fractional;
rk_del : vector(3) of fractional;
rk : fractional;
yfk : fractional;
ufk : vector(1) of fractional;
yfk1 : fractional;
gain_old : fractional := 0.2;
gain : fractional;
ginc : fractional;
a1_flt : scalable constant vector (2) of fractional := (0.950, 0.074);
a2_flt : scalable constant vector (2) of fractional := (-0.017, 0.950);
b1_flt : scalable constant vector (1) of fractional := (0.067);
b2_flt : scalable constant vector (1) of fractional := (-0.046);
c1_flt : scalable constant vector (2) of fractional := (-0.671, -1.049);
d1_flt : scalable constant vector (1) of fractional := (9.379E-04);
```

```
xk_flt1  : vector (2) of fractional;
xk1_flt1 : vector (2) of fractional;
u_flt1   : vector (1) of fractional;
y_flt1   : vector (1) of fractional;
temp1_flt1 : rawaccumulator;
xk_flt2  : vector (2) of fractional;
xk1_flt2 : vector (2) of fractional;
u_flt2   : vector (1) of fractional;
y_flt2   : vector (1) of fractional;
temp1_flt2 : rawaccumulator;
begin
  every 1.0E-04 do
    -- controller
    update (xk1, xk);
    -- sample inputs
    input (u);
    accumulate prescalpro (out1)
      y(1) := temp1 + d1 * u;
    end accumulate;
    -- output to plant
    output (y);
    accumulate scalpro (state1)
      xk1(1) := a1 * xk + b1 * u;
    end accumulate;
    accumulate scalpro (out1)
      temp1 := c1 * xk1;
    end accumulate;
    -- identification
    u_flt1(1) := y(1);
    u_flt2(1) := u(2);
    -- low-rate identification
    cnt := cnt + 1;
    if cnt > 10 then
      cnt := 0;
      ufk(1) := y_flt1(1);
      yfk1 := yfk;
      yfk := y_flt2(1);
      lk := yfk - yfk1;
      accumulate scalpro (state1)
        rk := r_coeff*ufk;
      end accumulate;
      rk_del(1) := rk;
      -- FIR delay-line
      accumulate scalpro (del) and update rk_del
        rk := rk_del_coeff*rk_del;
      end accumulate;
      gain_old := gain;
      ginc := (lk-rk*gain_old)*rk;
      gain := gain_old + ginc + ginc;
    end if;
    -- high-rate lowpass filtering for gain identification
    -- input filter
    update (xk1_flt1, xk_flt1);
```

```
    accumulate prescalpro (out1)
      y_flt1(1) := temp1_flt1 + d1_flt * u_flt1;
    end accumulate;
    accumulate scalpro (state1)
      xk1_flt1(1) := a1_flt * xk_flt1 + b1_flt * u_flt1;
    end accumulate;
    accumulate scalpro (state1)
      xk1_flt1(2) := a2_flt * xk_flt1 + b2_flt * u_flt1;
    end accumulate;
    accumulate scalpro (out1)
      temp1_flt1 := c1_flt * xk1_flt1;
    end accumulate;
    -- output filter
    update (xk1_flt2, xk_flt2);
    accumulate prescalpro (out1)
      y_flt2(1) := temp1_flt2 + d1_flt * u_flt2;
    end accumulate;
    accumulate scalpro (state1)
      xk1_flt2(1) := a1_flt * xk_flt2 + b1_flt * u_flt2;
    end accumulate;
    accumulate scalpro (state1)
      xk1_flt2(2) := a2_flt * xk_flt2 + b2_flt * u_flt2;
    end accumulate;
    accumulate scalpro (out1)
      temp1_flt2 := c1_flt * xk1_flt2;
    end accumulate;
  end every;
end controller_gain_ident;
```

Listing 1: DSPL example program

```
environment "DS1001" is
processor is "TMS 320C25";
program space off chip is from 20h to 3fffh;
data space on chip is from 200h to 3ffh;
data space off chip is from 400h to 3fffh;
stack space is from 60h to 7fh;
cycle time is 100;
program memory wait state is 0;
data memory wait state is 0;
input specification is
  u(1) is channel 0ee01h using ds2001 with start;
  u(2) is channel 0ee03h using ds2001 with start;
end input;
output specification is
  y(1) is channel 0ef0bh using ds2101;
end output;
end environment;
```

Listing 2: Environment description

```
DSPL - cross compiler, Vs 2.01, MS-DOS, target CPU : TMS 320C25
Copyright (C) 1988, 1989 by dSPACE GmbH

source file      : pcim.dsp
environment file : pcim.env
assembler file   : pcim.asm
xref file        : pcim.xrf
error file       : pcim.err

Compilation completed. No errors detected.

execution time requirements

task |  cycles  | rate (kHz) | time (us) | rqst (us) |  use (%)
-------------------------------------------------------------------
   1 |      431 |     23.202 |    43.100 |   100.000 |    43.10

total processor load    43.10 %

  498 words of code (off-chip).
   45 words of data (on-chip).
   32 words stack   (on-chip).
  134 lines compiled.
 2323 lines / minute.
```

Listing 3: Statistical information generated by DSPL25 compiler

```
; line 113
       zac
       lt     _v6              ; xk_flt1(1)
       mpyk   -564             ; a2_flt(1)
       lta    _v7              ; xk_flt1(2)
       mpy    _c8              ; a2_flt(2)
       lta    _v18             ; u_flt1(1)
       mpyk   -1533            ; b2_flt(1)
       apac
       adlk   1, 14 - 0        ; perform rounding
; overflow test and rescaling 0 bit
       sach   *, 1             ; save result
       sfl                     ; sign bit into carry flag
       bc     _120             ; branch if result < 0
       bgez   _121             ; branch if no positive overflow
       lalk   07fffh, 0        ; use positive saturation
       b      _122             ; update result
_120
       blz    _121             ; branch if no negative overflow
       lalk   08000h, 0        ; use negative saturation
       b      _122             ; update result
_121
       lac    *, 0             ; reload result
_122
       sacl   _v5, 0           ; xk1_flt1(2)
;
; ---- 24 cycles
```

```
;
; line 116
        zac
        lt      _v4                 ; xk1_flt1(1)
        mpyk    -688                ; c1_flt(1)
        lta     _v5                 ; xk1_flt1(2)
        mpyk    -1074               ; c1_flt(2)
        apac
        sacl    _v31, 0             ; temp1_flt1
        sach    _v31 + 1, 0         ; raw format
;
; ---- 8 cycles
;
; line 119
        blkd    00207h, _v10        ; xk1_flt2(1) --> xk_flt2(1)
        blkd    00208h, _v11        ; xk1_flt2(2) --> xk_flt2(2)
;
; ---- 6 cycles
;
```

Listing 4: Excerpts from assembly language source code generated by DSPL25 compiler

Conclusions

General purpose programming languages seem not very suitable for signal processing applications because of the lack of appropriate language constructs. Taking into account the special problems of digital signal processing and the special features of DSPs when designing a programming language, allows the implementation of compilers capable of generating extremely compact and efficient code. It also allows to provide the user with powerful support in the area of scaling, which is particularly important when working with fixed-point processors.

Literature

H. Hanselmann, "Digital Signal Processors in Motion Control", Proceedings International Workshop on Microcomputer Control of Electric Drives, Triest, Italy, July 3 - 4, 1989.

H. Henrichfreise, "The Control of an elastic Manipulation Device Using DSP", Proceedings American Control Conference, Atlanta, Georgia, Vol. 2, pp. 1029 - 1035, June 15 - 17, 1988.

H. Hanselmann and A. Engelke, "LQG-Control of a Highly Resonant Disk Drive Head Positioning Actuator", IEEE Transactions on Industrial Electronics, pp. 100 - 104, February 1988.

H. Hanselmann and W. Moritz, "High Bandwidth Control of the Head Positioning Mechanism in a Winchester Disk Drive", IEEE Control Systems Magazine, pp. 15 - 19, October 1987.

H. Hanselmann and A. Schwarte, "Generation of Fast Target Processor Code From High Level Controller Descriptions", Proceedings 10th IFAC World Congress, Munich, 1987.

Application of Kalman Filtering in Motion Control Using TMS320C25

Dr. S. Meshkat
The Control Group

One common problem in many industrial drive/control applications is sensing-sensing variables such as position, velocity or current for the purpose of control. The task of sensing signals that truly represent system variable is difficult either because of cost, imperfect sensors or environmentally induced random noise. The result is a control loop with less than optimum performance. To perform a proper control one has to "estimate" all or some of the missing system variables from a measurement that may be corrupted by noise (like a noisy encoder or current sensor) from a system that is excited by a random external force such as torque disturbance. The output of an optimum observer can be used in a feedback control system for the purpose of tracking or regulation.

But let's first define an estimation process. Estimation is referred to the process of extracting information, unavailable for measurement for any reason, from the available data. This data may contain measurement error and may also be influenced by external random disturbance. You may imagine, for instance, in a radar antenna positioning application where wind acts as a random torque disturbance, upon the motor shaft - a shaft whose position measurement is corrupted by random noise. In this application the observer or estimator must estimate the pure values for position and velocity. A Kalman filter is an optimum observer for these problems when state excitation noise (i.e., torques disturbance) and observation noise (i.e., the encoder noise) are uncorrelated, in other words encoder noise is totally unrelated to the torque disturbance.

To present the idea of designing a Kalman filter let's start with the model of a dc motor (See Appendix A, "Model of a dc Motor.")

$$\frac{\Theta(s)}{u(s)} = \frac{K_m}{s(\tau_m s + 1)} \tag{1}$$

Since the filter is implemented in a digital control environment we transfer this equations to the z domain.

$$G(z) = K_0 \frac{(z + b)}{(z-1)(z-e^{-aT})}, \quad K_0 = K_m \frac{aT - 1 + e^{-aT}}{a}$$

$$b = \frac{1 - e^{-aT} - aTe^{-aT}}{aT - 1 + e^{-aT}}, \quad a = \frac{1}{\tau_m} \tag{2}$$

In terms of state space representation:

$$\begin{pmatrix} \Theta(n+1) \\ \omega(n+1) \end{pmatrix} = A \begin{pmatrix} \Theta(n) \\ \omega(n) \end{pmatrix} + B\,u(n) \tag{3}$$

$$A = \begin{pmatrix} 1 & (a - e^{-aT})/a \\ 0 & e^{-aT} \end{pmatrix} \qquad B = \begin{pmatrix} K_m(T - 1/a + e^{-aT}) \\ K_m(1 - e^{-aT}) \end{pmatrix}$$

$$u(n) = -F \begin{pmatrix} \Theta(n) \\ \omega(n) \end{pmatrix}$$

This system may be excited by a random torque disturbance, W(n), furthermore position measurement may include random noise V(n) (see figure 1.)

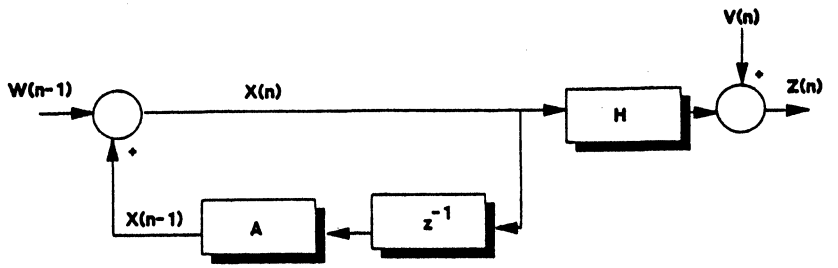


Figure 1: State space representation of a motion control system with torque disturbance W and measurement noise V

Optimum Observer

The problem can be stated as follows: design an observer that uses the measurement, z(n), as well as the statistical information about the measurement noise, V(n), and disturbance, W(n), to optimally estimate the actual position and velocity.

The reconstruction of data must be based on a structure that penalizes the deviation of estimator's output from the actual system output to correct the estimation process.

$$\hat{\underline{x}}(n) = A\,\hat{\underline{x}}(n-1) + K\,[z(n) - H\hat{\underline{x}}(n-1)] \qquad (4)$$

where $\hat{\underline{x}}(n)$ is the estimated vector of position and velocity and H is the output vector (e.g. for position H = [1 0])

This is presented in figure 2.

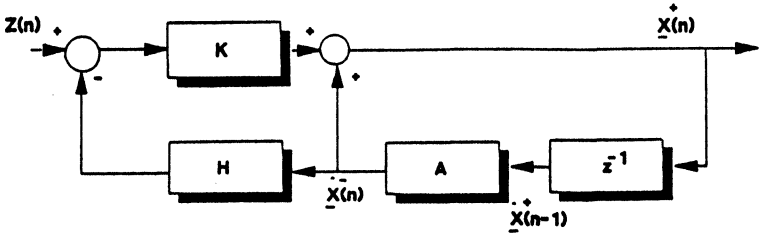


Figure 2: State space representation of optimum observer

Therefore the design problem can be simplified to finding filter K. Designing K requires the statistical information about the random disturbance and the measurement noise. This must be intuitively clear; simply because one could not imagine that without this information any "proper" reconstruction would be possible. This statistical information can be obtained from the knowledge of the torque disturbance intensity and the frequency range over which it is active. For our disturbance intensity and the frequency range over which it is active. For our measurement noise, we need to know the rms value of the noise and its frequency range. To be more precise, this information helps us compute the "state variance matrix of reconstruction error" from which K can be extracted.

Let's assume our motor is disturbed by external torque with an intensity of 12.5 N^2m^2s over the frequency spectrum of 0 - 30 Hz and the position measurement is corrupted by noise with the rms value of 0.2 degrees which has a flat spectral density over a 350 Hz range.

Figures 3 (a) and (b) show the actual position and velocity of our motor shaft when the motor is driven by the torque disturbance only. That is, if we had perfect position and velocity sensors we could take measurements like those illustrated in figures 3 a and b.

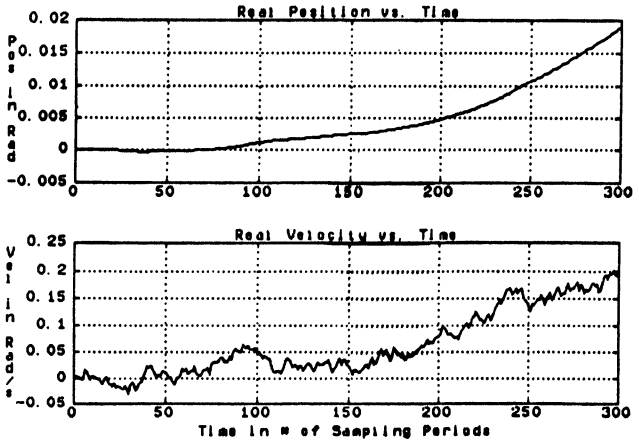


Figure 3: (a) the actual motor shaft position (b) actual shaft velocity

However, the real measurement is totally distorted by a random noise making it appear as shown in figure 4. Figure 4 shows a position measurement that looks absolutely hopeless. You must remember the noise, corrupting our position measurement, is not a high frequency noise that may be filtered by a conventional low pass filter. This noise spans a wide frequency range! The Kalman filter, however, can estimate the actual position (see figure 5) form the measurement signal (see figure 4). You may observe the perfect performance of this filter for velocity estimation as well (see figure 6). Our assumption is that the mean values of random disturbance and measurement noise are both zero.

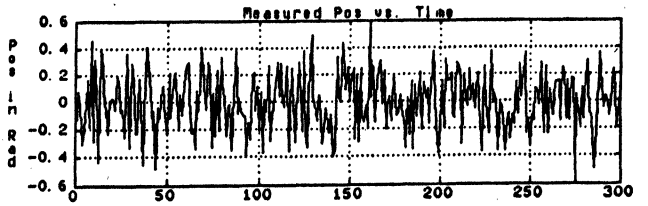


Figure 4: The measured position corrupted by noise

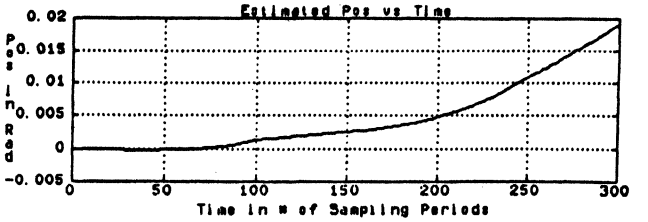


Figure 5: Estimated position using the optimum observer

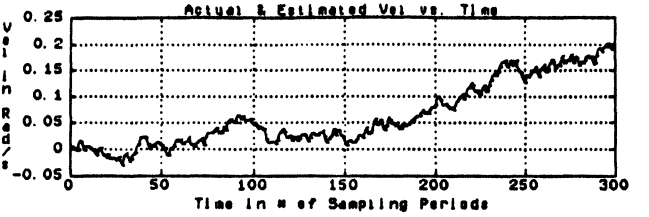


Figure 6: Estimated velocity using the optimum observer

Optimum Control

The estimation process provides the feedback data useful for the purpose of control. Using optimum control theory we may use the output of an optimum observer in a state feedback control configuration. The state feedback controller multiplies a designed control gain matrix, F, by the output of our estimator, x(n), in order to compute the control signal, u(n). Like any control design, F must be designed such that it satisfies a certain performance criteria. The performance criteria are dictated by the application. For example, in punch press application where achieving a fast response time is of crucial importance we need a time optimal control design. In machine tool applications where the instantaneous position/velocity error must be minimized, a linear quadratic controller may be an optimum choice. Although the performance criteria will influence the design procedure for matrix F, the implementation process in a state feedback control algorithm remains the same.

Combining Observer and Controller

Let's now look at the combination of our optimum observer and state feedback controller using linear quadratic criteria. Again, we start with the actual position and velocity values depicted in figures 7(a) and 7(b). Figure 8 shows the noisy measurement signal. The idea is to design an estimator combined with a regulator that use the

measured position, estimate the variables and use them in a state feedback control for the purpose of position and velocity regulation. Figures 9(a) and 9(b) show the contrast between what was available to the controller and the regulated results. The impressive contrast shows the power of optimum control in motion control applications.

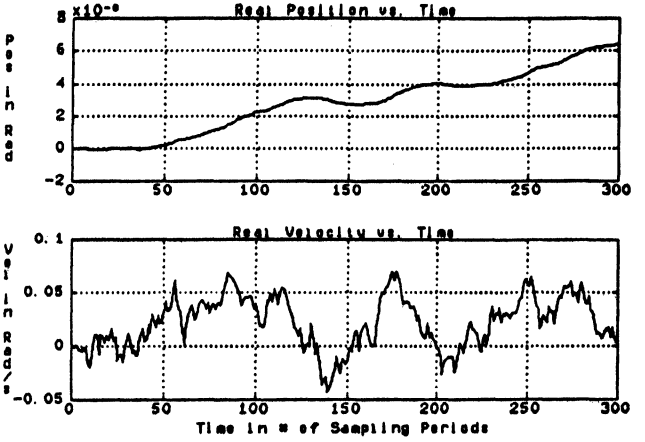


Figure 7: (a) actual motor shaft position (b) actual shaft velocity

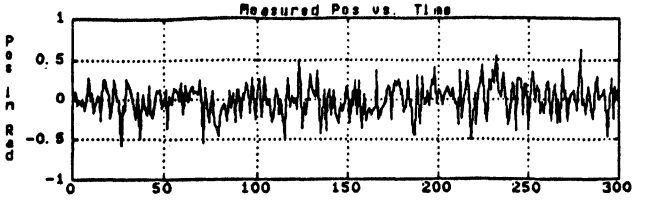


Figure 8: Measured position signal corrupted by noise

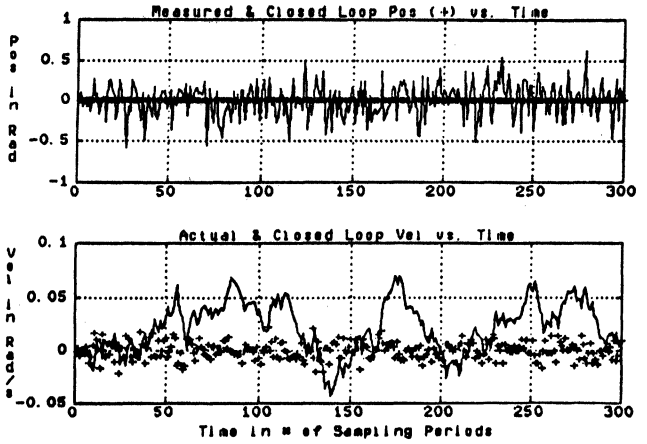


Figure 9: (a) measured position vs regulated position (b) measured vs regulated velocity

In a DSP environment an optimum observer combined with a linear quadratic regulator can be implemented and run at a sampling rate less than 30 microseconds. This can be done through cascading the various blocks of our control algorithm. Control algorithms implemented by DSPs allow systems with imperfect sensors to achieve and impressive level of performance - performance that is not achievable with classical control techniques.

Design and Implementation of Kalman Filter

To design a Kalman filter you may follow the steps discussed in sections "Theoretical Background..." then you may proceed with the selection of a simulation program. Three of the more popular control simulation programs that run on an IBM PC are: Matlab, Control C and Matrix X.

We used Matlab for the design and simulation of the Kalman filter. The Matlab program starts with data entry for your system matrices. You are also required to enter the statistical information about the plant disturbance and measurement noise. The program will simulate and plot the actual position and velocity on your EGA screen. It discretizes your system using the sampling period it was initially provided with. From this information, the discrete, stationary Kalman gain is computed and used in an optimum state observer. The estimated position is plotted and contrasted with the measurement signal. (Please see appendix B)

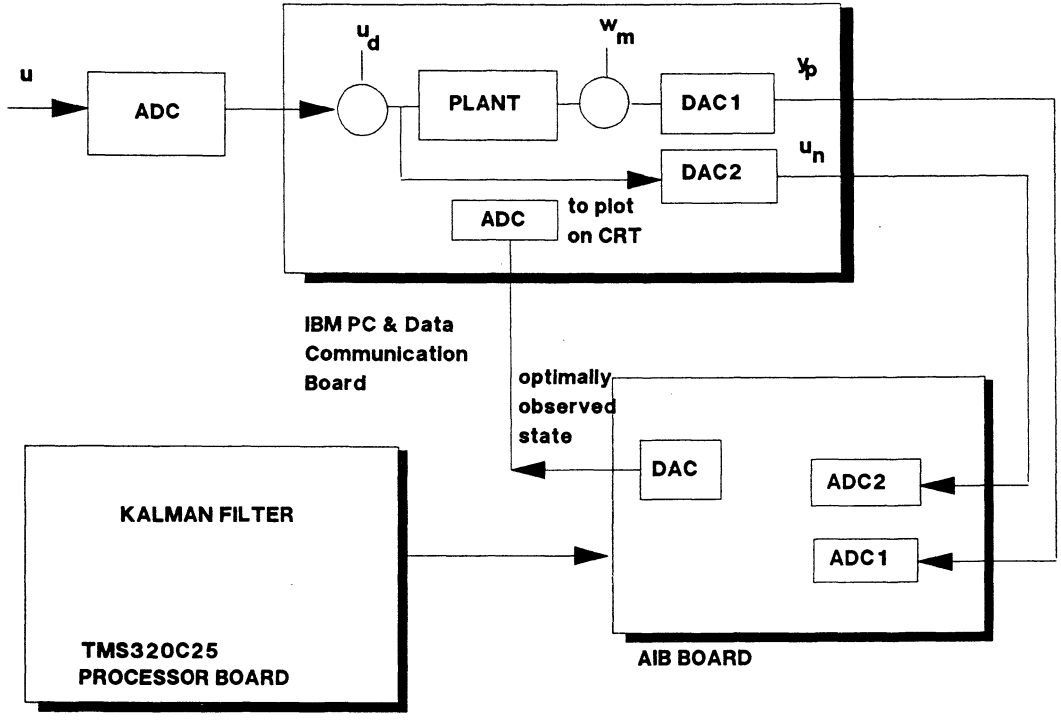
Hardware Setup

Once the design proves successful, you may readily convert your Kalman filter to a form that is implementable on TMS320C25 processor. For our implementation experiment we use the setup as appears in Figure 1. We used the IBM PC with a 80386 processor and 80387 co-processor to emulate the motor, in real time. The C program on the PC was also responsible for the generation of the uncorrelated normally distributed random disturbance and measurement noise. We used the IBM Data Communication Card to D/A and A/D our data. Through the communication card and Texas Instruments AIB board we could connect our emulated system to a TMS320C25 processor board. Needless to say, the TMS320C25 processor board was responsible for the implementation of Kalman filter and the initiation
of the two A/Ds and one D/A with a sampling period of 1 ms on the AIB board.

The filter we implemented embodies the generic form of equation 4 in the section entitled, "Application of Kalman Filtering in Motion Control Using DSPs." The state equations which were finally implemented are:

$$x_1(n+1) = x_1(n) + a_1x_2(n) + a_2y_p$$

$$x_2(n+1) = b_0u(n) + b_1x_2(n) + b_2y_p$$


u
ADC
u_d
w_m
PLANT
DAC1
y_p
DAC2
u_n
ADC
to plot
on CRT
IBM PC & Data
Communication
Board
optimally
observed
state
DAC
ADC2
ADC1
AIB BOARD
KALMAN FILTER
TMS320C25
PROCESSOR BOARD

where:

$a_1 = 0.0020$
$a_2 = 0.1737$

$b_0 = 16.000$
$b_1 = 0.9905$
$b_2 = 0.0008$

x_1 is the estimated position
x_2 is the estimated velocity
u_n is the input signal, u, plus the disturbance u_d
y_p is the measured signal corrupted by noise w_m

(Please see appendix B)

Theoretical Background for Designing Kalman Filter

Let's present a continuous time system by the following state equations.

$$\begin{aligned} x(t) &= a(t)x(t) + b(t)u(t) + w_1(t) \\ y(t) &= c(t)x(t) + w_2(t) \end{aligned} \tag{1}$$

where $w_1(t)$ and $w_2(t)$ are the state excitation noise and the measurement noise respectively.

The joint process of the two noise signals (i.e. col$[w_1\ w_2]$) can be expressed, as white noise, by the intensity matrix, V(t):

$$\begin{aligned} &E\{\ col[w_1(t_1)\ w_2(t_1)][w_1{}^T(t_2)\ w_2{}^T(t_2)]\} = \\ &V(t_1)\quad (t_1 - t_2) \end{aligned} \tag{2}$$

When the two noise signals are uncorrelated $v_{12} = v_{21} = 0$
and the intensity matrix becomes:

$$V(t) = \begin{pmatrix} v_1(t) & 0 \\ 0 & v_2(t) \end{pmatrix} \tag{3}$$

We can form the full order observer as:

$$\hat{x}(t) = a(t)\hat{x}(t) + b(t)u(t) + K(t)[y(t) - c(t)\hat{x}(t)] \tag{4}$$

The reconstruction error can be defined as:

$$e(t) = x(t) - \hat{x}(t) \tag{5}$$

Further we define the mean square reconstruction error as:

$$E\{e^T(t)W(t)e(t)\} \tag{6}$$

where W(t) is a positive-definite symmetric matrix.

The mean square reconstruction error value is a criterion to measure the observer's reconstruction capability.

So, the design problem can be stated as: Design K(t) such that the mean square reconstruction error is minimized.

It can be proven that the solution to the optimum observer problem can be obtained from:

$$K(t) = Q(t)c^T(t)v_2^{-1}(t) \tag{7}$$

Where Q(t) is the solution of the matrix Riccati equation:

$$Q(t) = a(t)Q(t) + Q(t)a^T(t) + v_1(t) - Q(t)c^T(t)v_2^{-1}(t)c(t)Q(t) \tag{8}$$

Therefore, the design process starts with obtaining all information regarding the process and the initial conditions for the estimated states. In addition, you need to obtain the values for the disturbance covariance matrix, v_1, and the measurement covariance matrix, v_2. This information helps you solve the matrix Riccati equations (8).

In the time-invariant case where all the matrices of equation 8 are constant, the steady-state solution to the observer's Riccati equation (8) can be obtained from:

$$0 = aQ + Qa^T + v_1 - Qc^T v_2^{-1} c\, Q \tag{9}$$

Accordingly, "the steady-state optimum observer gain matrix" can be calculated as:

$$K = Qc^T v_2^{-1} \tag{10}$$

Notice that in the time invariant case, there is always a trade off between the observer's speed and the immunity to the observation noise. In terms of design practice, one may experiment with the two factors of observer speed and noise immunity. To do this: keep v_1 constant, choose a positive-definite symmetric matrix for v_2 with a positive scalar multiplier m. Clearly, increasing m will increase the state reconstruction speed. The value for m may be increased to a point that while the observer attains a fast speed, noise immunity is not compromised.

Example:

Let's assume that our plant is a motor, disturbed by a zero mean white noise external torque, T_d, and our shaft position measurement is corrupted by a zero mean white noise, M_n, uncorrelated to the disturbance noise. This plant can be modeled as:

$$\frac{\Theta(s)}{u(s)} = \frac{Km}{s(T_m s+1)}$$

where $K_m = [1/K_T]$ (KT is motor torque constant) and $T_m = RJ/K_T^2$ (R is the armature resistanc and J is the total inertial load.)

In terms of state equations:

$$x(t) = \begin{pmatrix} 0 & 1 \\ 0 & 1/\tau_m \end{pmatrix} x(t) + \begin{pmatrix} 0 \\ K_T/RJ \end{pmatrix} u(t) + \begin{pmatrix} 0 \\ 1/J \end{pmatrix} T_d(t)$$

The state disturbance noise intensity, v_d may be obtained from the variance of the torque disturbance and the frequency range over which it is active.

$$v_d = \frac{\text{torque disturbance variance}}{2(\text{active frequency range})}$$

The same for the measurement noise intensity, v_m

$$v_m = \frac{\text{Measurement noise variance}}{2(\text{active frequency range})}$$

from this information v_1 and v_2 can be obtained as follows:

$$v_1 = \begin{pmatrix} 0 & 0 \\ 0 & v_d/J^2 \end{pmatrix}$$

and

$$v_2 = v_m$$

The above information enables us to solve equation 9 for Q and plug in the result in equation 10. The solution obtained for the optimum observer gain, K can be used in our reconstruction equation (i.e., equation 4).

APPENDIX A

Model For a DC Motor

The model equations are obtained using the physical relation between the variables in each functional block. We use Laplace operator, s, to simplify the solution method; but remember that "s" is an appropriate operator only for linear systems. The mathematical model of a dc motor will allow us to simulate the system dynamic response on a computer before an actual design.

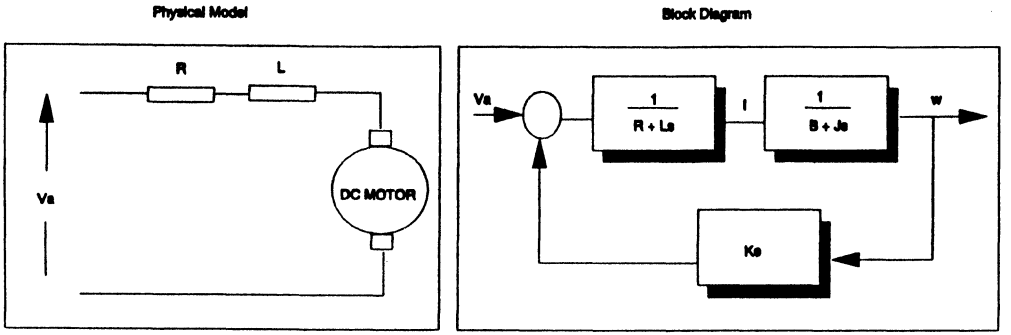


Figure 1 shows an electromechanical block diagram of a DC motor. This model describes the relationship between the voltage applied across the armature winding ,u, and velocity, w.

Where

R	=	armature's resistance
L	=	armature's inductance
J	=	motor inertia
B	=	viscous damping coefficient
K_t	=	Torque constant
K_e	=	back emf voltage constant

Using figure 1, the relationship between (s) and u(s) can be written as:

$$\frac{\omega(s)}{u(s)} = \frac{1}{LJ} \quad \frac{K_t}{s^2 + (R/L + B/J)s + RB + K_tK_e/LJ}$$

Equation 1 describes a second order model for a DC motor. In MKS system, $K_e = K_t$.

$$\frac{\omega(s)}{u(s)} = \frac{K_t}{LJ} \quad \frac{1}{s^2 + (R/L + B/J)s + RB + K_t^2/LJ}$$

In a practical motor the roots of the denominator, "poles" are in general real and negative. These roots are:

$$\lambda_1, \lambda_2 = (-1/2)(R/L + B/J) +/- \sqrt{1/2} \quad (R/L + B/J)^2 - 4(RB + K_t)/LJ$$

For a step-wise input voltage to a motor, u(s) =1/s, output velocity is:

$$\omega(s) = \frac{1}{s} \cdot \frac{K_t}{LJ} \cdot \frac{1}{(s+\lambda_1)(s+\lambda_2)} \qquad (1)$$

The model presented by the above equation can be simplified to a first order model using the following assumptions. The first assumption is that the electrical time constant, τ_e, in most conventional DC motors is much shorter than the mechanical time constant, τ_m. This will let us ignore the term s.L in equation 1.

$$\frac{\omega(s)}{u(s)} = \frac{K_t}{RJs + RB + K_t^2}$$

The second assumption is $K_t^2 >> RB$

$$\frac{\omega(s)}{u(s)} = \frac{1}{K_t} \cdot \frac{1}{1 + \frac{RJ\ s}{K_t^2}}$$

Where $\tau_m = RJ/Kt^2$ is the mechanical time constant. So, for a stepwise input voltage applied to the armature winding the shaft speed w(s) is given by:

$$\omega(s) = \frac{1}{s} \cdot \frac{1}{K_t} \cdot \frac{1}{(1+\tau_m s)}$$

Extending this relation to the angular position, will result in:

$$\frac{\Theta(s)}{u(s)} = \frac{K_m}{s(\tau_m s + 1)}$$

Appendix B

```
%              Discrete Time, Stationary Kalman Filter
%
%       In this segment of program you will enter the
%       continuous time system matrices a,b and c.
%       We assume d = 0.
%
%
subplot (211)
input('input the continuous time system matrix a: ')
a = ans;
input('input the continuous time input vector b: ')
b=ans;/
input('input the continuous time output vector c: ')
c=ans;
%
%
%       At this point you will enter the sampling period, T.
%       This value enables your program to discretize the
%       entered system equations.
%
%
input('input the sampling period T: ')
T = ans;
%
%
%       At this point you will be asked to enter the
%       statistical information about the disturbance
%       noise and the measurement noise.  For more
%       information please refer to the document entitled
%       "theoretical background."
%
input('input the system disturbance vector g: ')
g = ans;
input('input the disturbance covariance matrix q; ')
q = ans;
input('input the variance value for the disturbance vard: ')
vard = ans;
input('input the variance value for the measurement varm: ')
varms = ans;
r=varm/1000;
%
%
%       In this part you will enter any known input signal from
%       which the program will generate the total input.
%
%
input('Enter the input u')
u2 = ans;
[A,B] = c2d(a,b,T)
pause
u=rand('normal');
u1=rand('normal');
u=vard*rand(300,1);
u=u+u2;
```

```
u1 = varm*rand(300,1);
%
%
%       At this point program can simulate the
%       actual position and velocity signals as well as the
%       optimum discrete observer gains.
%
%
yp = dlsim(A,B,c,0,u);
yv = dlsim(A,B,[0 1],0,u);
q1 = vard/1000;
q = q1*q;
[L,M,P] = dlqe(A,T*g,c,q*T,r/T)
pause
t = 1:1:300;
plot(t,yp)
title('Real Position vs. Time')
grid
ylabel('Pos in Rad')
plot(t,yv)
title('Real Velocity vs. Time')
grid
xlabel('Time in # of Sampling Periods')
ylabel('Vel in Rad/s')
pause
yp = yp+u1;
plot(t,yp)
title('Measured Pos vs. Time')
grid
ylabel('Pos in Rad')
x = [0;0];
%
%
%       Using the Kalman gain, the program will structure a
%       recursive equation for the optimum estimation process.
%
%
kgain = A-L*c;

for i = 1:1:300;
                x = kgain*x+B*u(i,1)+L*yp(i,1);
                pos(i,1) = x(1,1);
                vel(i,1) = x(2,1);
end
%
%
%       At this point the program will plot the estimated
%       position and velocity, and contrast them against
%       measured ones.
%
plot(t,pos)
title('Estimated Pos vs Time')
grid
ylabel('Pos in Rad')
xlabel('Time in # of Sampling Periods')
pause
plot(t,yp,'.',t,pos)
```

```
title('Measured & Estimated Pos vs. Time')
grid
ylabel('Pos in Rad')
plot(t,yv,'.',t,vel)
title('Actual & Estimated Vel vs. Time')
grid
ylabel('Vel in Rad/s')
xlabel('Time in # of Sampling Periods')
meta mm
subplot
end
```

Appendix C

```
0001  ************************************************
0002  *                                              *
0003  *       Kalman Filtering using TMS320C25       *
0004  *                                              *
0005  ************************************************
0006
0007
0009  0000              text
0010  0000              TEMP    .equ        0h        ;For temportary storage
0011
0012  0001              YP      .equ        1h        ;BLOCK B0 FOR STATE
                                                      VARIABLES
0013  0002              VN      .equ        2h        ;DATA MEMORY
0014  0003              XNL     .equ        3h        ;
0015  0004              XNH     .equ        4h        ;
0016  0005              UN      .equ        5h        ;
0017                    *
0018                    *
0019  0006              A2      .equ        6h        ;THEY STORE THE
                                                      COEFFICIENTS
0020  0007              A1      .equ        7h        ;
0021  0008              B2      .equ        8h        ;
0022  0009              B1      .equ        9h        ;
0023  000A              B0      .equ        0Ah       ;
0024                    *
0025
0026  0000              .asect  "AORG00"    00h
0027  0000  FF80        B                   STRT
0001  0020
0028
0029
0030
0031  0000                      .data
0032  0010              .asect  "AORG01"    10h
0033
0034  0010  1387        RATE    .word       4999      ;sampling period 1 msec [= (5MHz
                                                      /(RATE+1)]
0035  0011  00FA        MODE    .word       0FAh      ;For AIB initization
0036
0037
0038
0039
0040  0000              .text
0041  0020              .asect  "AORG02"    20h
0042
0043              * ----------------------------
0044              *
0045              *    Start main program
0046              *
0047              * ----------------------------
0048
0049
0050  0020              STRT:   .equ        $
0051  0020  CE06                RSXM                  ;TURN OFF THE SIGN
```

```
                                                        EXTENTION MODE
0052
0053    0021    C800            LDPK    0       ;
0054    0022    5589            LARP    AR1     ;
0055
0056
0057    0023    C100            LARK    AR1,0   ;ZERO THE DATA
                                                MEMORY (0h TO 5h)
0058    0024    CA00            ZAC             ;
0059    0025    CB07            RPTK    7       ;
0060    0026    60A0            SACL    *+      ;
0061
0062
0063
0064            * ---------------------------------
0065            * Initialize the coefficients
0066            * ---------------------------------
0067
0068
0069
0070    0027    D001            LALK    712,0   ;STORING COEFFS IN DATA
                                                MEM (A2 = .1737 Q12)
        0028    02C8
0071    0029    6006            SACL    A2,0    ;AND XFER THEM TO
                                                PROG MEM
0072
0073    002A    D001            LALK    8,0     ;(A1 = 0.002 STORED IN
                                                        Q12)
        002B    0008
0074    002C    6007            SACL    A1,0    ;
0075
0076    002D    D001            LALK    3,0     ;(B2 = 0.0007324 STORED
                                                IN Q12)
        002E    0003
0077    002F    6008            SACL    B2,0    ;
0078
0079    0030    D001            LALK    4057,0  ;(B1 = 0.9905 STORED IN Q12)
        0031    0FD9
0080    0032    6009            SACL    B1,0    ;
0081
0082    0033    D001            LALK    4096,0  ;(B0 = 16  STORED IN Q8)
        0034    1000
0083    0035    600A            SACL    B0,0    ;
0084
0085
0086
0087            * ---------------------------------
0088            * Initialize the AIB board
0089            * ---------------------------------
0090
0091    0036            LOOP:   .equ    $
0092    0036    CA10            LACK    RATE    ;AIB BOARD SET FOR 1 MS
                                                SAMPLING RATE
0093                    *                       ;AND FOR 2 ANALOG TO
                                                DIGITAL
                                                CONVERTERS
0094    0037    5800            TBLR    TEMP    ;
```

```
0095  0038  E100          OUT   TEMP,1  ;WRITE THE SAMPLING PERIOD
                                        TO AIB
0096                *                   ;BOARD PORT 1
0097
0098  0039  CA11          LACK  MODE    ;INITIALIZE THE AIB BOARD
0099  003A  5800          TBLR  TEMP    ;
0100  003B  E000          OUT   TEMP,0  ;WRITE THE SAMPLING PERIOD
                                        TO AIB
0101                *                   ;BOARD PORT 0
0102  003C  CE08          SPM   0       ;reset the P register output shift mode
0103
0104
0105
0106
0107  003D  FA80    WAIT: BIOZ  TAKE    ;WAIT FOR THE A/D
                                        INTERRUPT COMES
      003E  0041
0108  003F  FF80          B     WAIT
      0040  003D
0109
0110
0111
0112
0113        * ====================================
0114        *
0115        * Start doing each sampling calculations
0116        *
0117        * ====================================
0118
0119
0120  0041          TAKE: .equ  $
0121
0122  0041  8201          IN    YP,2    ;TAKE SAMPLE OF FIRST ADC --
                                        STORE IN YP
0123  0042  8305          IN    UN,3    ;TAKE SAMPLE OF SECOND
                                        ADC -- STORE IN UN
0124
0125  0043  CA00          ZAC           ;CLEAR ACCUMULATOR
0126
0127  0044  3C01          LT    YP      ;
0128  0045  3806          MPY   A2      ;P REG. = A2*YP SHIFTED 4
                                        PLACES LEFT
0129  0046  CE0A          SPM   2       ;
0130
0131
0132  0047  3D02          LTA   VN      ;MULT REG = A2*YP SHIFTED 4
                                        PLACES LEFT
0133  0048  3807          MPY   A1      ;
0134  0049  CE0B          SPM   3       ;RIGHT SHIFT 6 PLACES
0135  004A  CE15          APAC          ;ACC = A1*VN + A2*YP
0136
0137  004B  4903          ADDS  XNL     ;ADD XNH,XHL TO ACC
0138  004C  4804          ADDH  XNH     ;
0139
0140  004D  6003          SACL  XNL,0   ;SAVE THE NEW STATE VALUE
0141  004E  6804          SACH  XNH,0   ;
0142
```

```
0143
0144
0145
0146    004F    CA00        ZAC                 ;CLEAR ACCUMULATOR
0147
0148    0050    3809        MPY     B1          ;P=B1*VN
0149    0051    CE0A        SPM     2           ;
0150
0151    0052    3D05        LTA     UN          ;ACC=B1*VN
0152    0053    380A        MPY     B0          ;P=B0*UN
0153
0154    0054    CE0B        SPM     3           ;
0155    0055    3D01        LTA     YP          ;ACC = B1*VN+B0*UN
0156    0056    3808        MPY     B2          ;P=YP*B2
0157    0057    CE15        APAC                ;ACC = B2*YP + B1*VN +
                                                B0*UN(RESULTt
0158                    *                       ;IS SHIFTED LEFT 4 PLACE TO
                                                MAKE
0159                    *                       ;B0 Q12 AND THUS B0*UN Q18u
0160
0161    0058    6802        SACH    VN          ;STORE THE NEW VN
0162
0163    0059    E205        OUT     UN,2        ;CHECK TO SEE IF OPERATION
                                                ENDED
0164

0165    005A    FF80        B       WAIT        ;BACK FOR MORE SAMPLES
        005B    003D
0166
0167                        END
```

Implementation of a PID Controller on a DSP†

Karl Johan Åström
Department of Automatic Control
Lund Institute of Technology
Lund, Sweden

Hermann Steingrímsson
Graduate School of Business
University of Wisconsin
Madison, Wisconsin, USA

1. Introduction

The PID controller is by far the most commonly used control algorithm. [Deshpande 1981] Although it is of limited complexity it can be used to solve a large number of industrial control problems. The textbook version of the PID controller can be described by the equation

$$u(t) = K_c\left(e(t) + \frac{1}{T_i}\int^t e(s)ds + T_d\frac{de(t)}{dt}\right) \quad (1)$$

where u is the control variable and e is the control error, defined as $e = y_{sp} - y$, where y_{sp} is the set point and y is the process output. The parameters of the controller are: gain K_c, integral time T_i, and derivative time T_d.

The purpose of the integral action is to increase the low-frequency gain and thus reduce steady-state errors. Derivative action adds phase lead, which improves stability and increases system bandwidth.

Implementation of a PID controller using a DSP will be discussed in this paper. A lot of experience has accumulated over many years of use of the algorithm. This has led to significant modification of the algorithm (1). These modifications will be discussed in Section 2, where the discretization issues are also dealt with. The result is a nonlinear digital algorithm that is suitable for implementation on a general purpose digital computer.

The algorithm can be implemented in a straightforward way in a DSP with floating point hardware. Implementation using an ordinary DSP does, however, require special considerations, because all calculations have to be made in integer arithmetic. These issues are discussed in Section 3. Some special problems related to quantization in AD- and DA-converters are discussed in Section 4. An overview of the DSP code for a PID controller is described in Section 5. The complete code is given in the Appendix. In Section 6 it is described how the code can be tested. The tests given include both linear and nonlinear behavior.

† Part of this work was done when the first author was visiting professor and the second author a graduate student at the University of Texas at Austin.

2. Modification and Discretization

The algorithm (1) has several drawbacks. Significant modifications of linear and nonlinear behavior are necessary in order to obtain a practically useful algorithm. See [Åström and Hägglund 1988]. To obtain equations that can be implemented using computer control it is also necessary to replace continuous time operations like derivation and integration by discrete time operations. See [Åström and Wittenmark 1990]. These modifications will be described in this section.

Proportional Term

The proportional term $K_c e(t)$ is implemented simply by replacing the continuous time variables with their sampled equivalences. One additional modification set point weighting [Åström and Hägglund 1988] has been found useful. This means that the proportional term only acts on a fraction b of the command signal. The proportional term then becomes

$$P(t_k) = K_c\big(by_{sp}(t_k) - y(t_k)\big) \quad (2)$$

where $\{t_k\}$ denotes the sampling instants. The parameter b admits independent adjustment of set point and load disturbance responses. It may also be viewed as "zero-placement".

Integral Term

When a controller operates over a wide range of operating conditions, the control variable may reach actuator limits. The feedback loop is then broken and the system effectively runs open loop. When this happens in a controller with integral action, the error will continue to be integrated and the integral term may become very large. The integrator "winds up". The error must then change sign for a long period of time to "unwind" the integrator and bring the system back to normal. Windup can also cause problems when the controller is implemented on a microprocessor having finite word length. Since the processor can only store numbers limited in magnitude, windup may cause overflow oscillations in the control variable, unless saturation arithmetic is used.

There are several ways to avoid windup. One possibility is to introduce an extra feedback loop by measuring the output from the actuator and forming an error signal as the difference between the controller output v, and the actuator output u. If the output of the actuator is not available, the signal may be computed by using a mathematical model of the actuator. The error signal is fed to the input of the integrator through the gain $1/T_t$, where the constant T_t is called the tracking time constant. The extra feedback will ensure that the integral obtains a value so that the controller output tracks the saturated output. Tracking is accomplished with the time constant T_t. Using this method of avoiding windup the integral term becomes

$$I(t) = \frac{K_c}{T_i}\int e(s)ds + \frac{1}{T_t}\int \big(u(s) - v(s)\big)ds \quad (3)$$

To obtain an algorithm that can be implemented on a computer, the integral term $I(t)$ is differentiated

$$\frac{dI(t)}{dt} = \frac{K_c}{T_i}e(t) + \frac{1}{T_t}e_s(t)$$

where $e_s(t) = u(t) - v(t)$. Approximating the derivative by a forward difference gives

$$\frac{I(t_{k+1}) - I(t_k)}{h} = \frac{K_c}{T_i}e(t_k) + \frac{1}{T_t}e_s(t_k)$$

where h is the sampling period. Finally, by rearranging terms, we get the following equation to compute the integral term

$$I(t_{k+1}) = I(t_k) + \frac{K_c h}{T_i}e(t_k) + \frac{h}{T_t}e_s(t_k) \quad (4)$$

Derivative Term

A pure derivative should not be implemented, because the controller gain becomes very large at high frequency. This leads to amplification of high-frequency noise. The derivative term is therefore approximated by

$$sT_d \approx \frac{sT_d}{1 + sT_d/N} \quad (5)$$

Notice that the approximation is good for signals whose frequency contents are significantly below N/T_d. Also notice that the approximating transfer function has a maximum gain of N. Parameter N is therefore called maximum derivative gain. In analog controllers N is given a fixed value, typically in the range of 5–20.

It is also advantageous not to let the derivative act on the set point signal. The set point is constant for most of the time and its derivative is therefore zero. A step change in the set point may, however, cause an undesirable jump in the control variable if the derivative acts on the set point. With these modifications the derivative term can be written as

$$D + \frac{T_d}{N}\frac{dD}{dt} = -K_c T_d \frac{dy}{dt} \quad (6)$$

There are several methods to approximate the derivative. Common methods are the forward difference approximation, the backward difference approximation, Tustin's approximation and ramp equivalence. See [Åström and Wittenmark 1990]. These approximations all have the same form

$$D(t_k) = aD(t_{k-1}) - b\big(y(t_k) - y(t_{k-1})\big) \quad (7)$$

and are stable only if $|a| < 1$. The forward difference approximation is stable if $T_d > Nh/2$. It thus becomes unstable for small values of T_d. Tustin's approximation has the disadvantages that a goes to 1 as T_d goes to zero. This gives a ringing response for small T_d. The ramp equivalence approximation gives exact outputs at the sampling instants if the signal is continuous and piece wise linear between the sampling instants, but it requires computations of an exponential. The backward difference approximation gives good results for all values of T_d, The parameter a goes to zero as T_d goes to zero. Here the backward difference approximation is chosen.

The following is obtained when Equation (6) is approximated by a backward difference:

$$D(t_k) + \frac{T_d}{N} \cdot \frac{D(t_k) - D(t_{k-1})}{h} = -K_c T_d \frac{y(t_k) - y(t_{k-1})}{h}$$

Rearranging terms, gives (7) with

$$a = \frac{T_d}{T_d + Nh} \quad \text{and} \quad b = \frac{K_c T_d N}{T_d + Nh}$$

which is the formula that will be used to compute the derivative term.

The PID Algorithm

Summarizing we find that a practical version of the PID algorithm can be described by the following equations:

$$\begin{aligned} P(t_k) &= K_c(by_{sp} - y(t_k)) \\ D(t_k) &= a_d D(t_{k-1}) + b_d(y(t_{k-1}) - y(t_k)) \\ v(t_k) &= P(t_k) + I(t_k) + D(t_k) \\ u(t_k) &= f(v(t_k)) \\ I(t_{k+1}) &= I(t_k) + b_i(y_{sp} - y(t_k)) \\ &\quad + b_t(u(t_k) - v(t_k)) \end{aligned} \tag{8}$$

This algorithm has anti-windup reset, limitation of derivative gain (N) and set point weighting (b).

The function f describes the nonlinear characteristic of the actuator. For a linear actuator with saturation at u_{min} and u_{max} we have

$$f(v(t_k)) = \begin{cases} u_{max} & \text{if } v(t_k) > u_{max} \\ u_{min} & \text{if } v(t_k) < u_{min} \\ v(t_k) & \text{otherwise} \end{cases} \tag{9}$$

For actuators with other limitations the function f should be modified. The parameters a_d, b_d, b_i and b_t are related to the primary parameters K_c, T_i, T_d, T_t and N at the PID controller as follows:

$$\begin{aligned} a_d &= \frac{T_d}{T_d + Nh} \\ b_d &= \frac{K_c N T_d}{T_d + Nh} \\ b_i &= K_c h / T_i \\ b_t &= h / T_t \end{aligned} \tag{10}$$

Since Equations (10) have to be updated only when the controller parameters are changed, the code should be organized so that parameters a_d, b_d, b_i and b_t are computed initially and when the PID parameters are changed. This will reduce the computational load during the execution of the PID algorithm. The structure of the PID algorithm given by Equation (8) is shown in Figure 1. Notice that the algorithm is in parallel form.

The PI algorithm

In many cases the derivative action is not necessary. The algorithm then reduces to

$$\begin{aligned} P(t_k) &= K_c(by_{sp} - y(t_k)) \\ v(t_k) &= P(t_k) + I(t_k) \\ u(t_k) &= f(v(t_k)) \\ I(t_{k+1}) &= I(t_k) + b_i(y_{sp} - y(t_k)) \\ &\quad + b_t(u(t_k) - v(t_k)) \end{aligned} \tag{11}$$

which is a PI controller with anti-windup reset and set point weighting (b).

The function f is the same as in Equation (9) and the parameters b_i and b_t are related to the parameters K_c, T_i and T_t as follows:

$$\begin{aligned} b_i &= K_c h / T_i \\ b_t &= h / T_t \end{aligned} \tag{12}$$

which is the same as Equation (10). The reason for considering this special case is that PI controllers are in fact more common than controllers with derivative action.

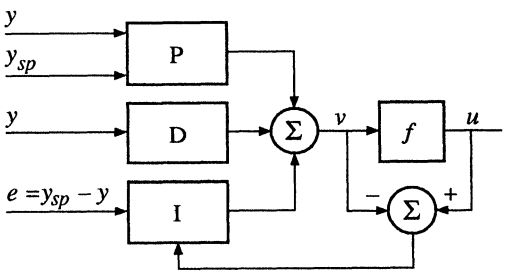


Figure 1. Structure of the PID controller with anti-windup.

Table 1. Number of arithmetic operations for PI and PID control.

	PI		PID	
	M	A	M	A
P	2	1	2	1
D	0	0	2	2
v	0	1	0	2
f	X	X	X	X
I	2	4	2	4
Tot	4	6	6	9

Operations Count

It is a common practice to estimate computation times by a simple operation count. This can be strongly misleading when using fixed point calculation, because much of the computation time may be spent on overflow handling and scaling. Table 1 shows the minimum number of multiplications and additions required for the PID and PI algorithms. The PID algorithm requires 15 arithmetic operations, while the PI algorithm requires 10 operations.

3. Implementation Issues

Implementation of a PID-controller using a DSP with fixed point will now be discussed. General practice on implementing algorithms for DSP are given in [Texas Instruments 1986], [Texas Instruments 1989a], [Texas Instruments 1989b], [Texas Instruments 1990a] and [Texas Instruments 1990b].

To perform fix-point calculations it is necessary to know orders of magnitude of all variables. Simulations were performed to get this information. In the simulations the process model

$$G(s) = \frac{1}{(s+1)^4}$$

was used. Figure 2 shows the step response of the system with parameters $K_c = 0.6$, $T_d = 0.5$, $T_i = 2.2$, $T_t = 0.5$, $N = 8$, and a sampling period of 0.1 s. At the time $t = 0.3$ s a load disturbance of 0.3 V is introduced.

Two C-programs were written to test the effects of scaling and roundoff. One program implements the PID controller in double precision arithmetics with no attempt to simulate the effect of finite word length. The other program simulates

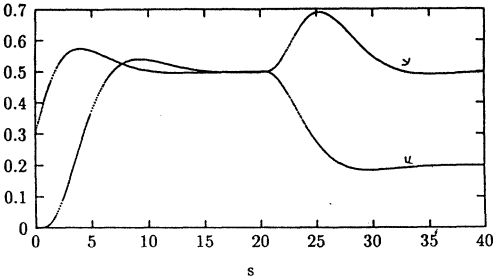


Figure 2. Step response of the system.

the Texas Instruments DSP by using a 32-bit accumulator and a 16-bit word length. The effect of using different resolution of the A/D- and D/A-converters can also be simulated.

Selection of Sampling Period

There are several rules of thumb for choosing the sampling period for digital controllers. For a PI-controller the sampling period is related to the integration time. A rule of thumb [Åström and Wittenmark 1990] is

$$\frac{h}{T_i} \approx 0.1 - 0.3$$

A PID controller requires a much shorter sampling period. The sampling period should be short enough so that the pole $s = -N/T_d$, introduced to limit the high frequency gain of the derivative, can be approximated appropriately. This leads to the following rule of thumb:

$$\frac{hN}{T_d} \approx 0.2 - 0.6$$

See [Åström and Wittenmark 1990].

Integral Offset

Roundoff may give an offset when the integral term is implemented on a computer with a short word length. This can be understood as follows. Consider the equation for the integral term in Equation (8). The correction term $b_c e(t_k) = K_c h/T_i \cdot e(t_k)$ is usually small in comparison to $I(t_k)$ and may therefore be rounded off. With fractional arithmetic, the largest magnitude of the correction term is $K_c h/T_i$. To avoid roundoff, it is therefore necessary to have a word length of at least

$$\text{number of bits} = -\frac{\log(K_c h/T_i)}{\log(2)}$$

More bits are of course required to obtain meaningful values. For example, with $h = 0.02$ s, $T_i = 10$ s and $K_c = 0.1$ the number of bits required to obtain less than 5% error in the integral requires a word length of at least

$$\text{number of bits} = -\frac{\log(0.0002 \cdot 0.05)}{\log(2)} \approx 17$$

Longer sampling periods for computing the integral may be used to avoid the offset. This can be done simply by adding the error over each sampling period and updating the integral term in regular intervals. Another way to avoid offset due to roundoff is to store the integral with higher precision. In most DSPs (like the TMS320xx) values can be stored in double precision, with little overhead.

Scaling

The PID controller given by Equations (8) is already in parallel form, with the modules of zero and first order. Figure 1 illustrates the realization of the controller. Because of the parallel form, the P, I and D terms can be scaled and computed separately and then unified to form v.

Coefficient Scaling

Because of the wide number range of the parameters, some restrictions must be imposed on the magnitude of coefficients. It follows from Equation (10) that b_d is the largest parameter. A limit should therefore be set on the gain K_c, and the high-frequency derivative gain N. If K_c and N are limited to 16, we have $b_d < K_c N = 256$ and $K_c \leq 16$. These parameters must therefore be divided by 256 and 16 respectively before they are stored. To restore the magnitude of the signal, the derivative term must be shifted left by 8 bits and the proportional term shifted left by 4 bits.

The other parameters, a_d, b_i and b_t are within the number range, but because b_i and b_t may become very small, it is advantageous to also set a lower limit on h/T_i and h/T_t.

Signal Scaling and Saturation Arithmetic

It must be insured that overflow does not occur when computing the states of the controller. With the structure of the PID controller shown in Figure 1 the states are $D(t_k)$ and $I(t_{k+1})$. Care must also be taken so that overflow does not occur when the P, I and D terms are added to obtain v.

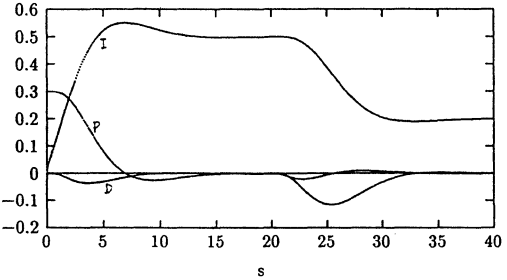


Figure 3. The terms of the PID controller.

The proportional term will always be within the number range, since the multiplication of a fraction with a fraction gives a fraction. Overflow can occur if K_c is larger than 1 when the magnitude of the signal is restored. It is therefore necessary to use saturation arithmetic when computing the proportional term.

One additional advantage of using the anti-windup reset when computing the integral term is that the integral is within the number range. Saturation arithmetic is therefore not necessary. Integration can result in overflow if anti-windup is not used or if T_t is chosen poorly. Saturation arithmetic should therefore be used before the integral is stored.

Since the derivative depends only on the process output, it is difficult to use analytic scaling methods effectively. It is easy to predict the worst possible input, but for most processes that would be too pessimistic. A good engineering approach is therefore to simulate the closed loop system and store the output of the derivative for a few representative examples. The derivative should normally not account for more than 20% of the control signal. Since b_d can take large values, saturation arithmetic should be used before storing the derivative. A number of simulations were made in order to obtain typical orders of magnitude of the proportional, integral and derivative term. It turns out, that under normal operation conditions, the variables are within the number range. Since we are allowing a gain larger than one, it is very likely that an overflow will occur under some operation condition, for example during start-up. Saturation arithmetic is therefore used on both states and on the control signal v. Figure 3 shows Simnon plots of the P, I and D terms for step response and load disturbances, for the process and the controller previously used.

Gain, Input and Output Scaling

To implement a high gain ($K_c > 1$) one can either include the gain in the digital algorithm or move the gain "outside" of the DSP by using a linear amplifier. The advantage of the latter approach is that the control algorithm can be scaled to eliminate the danger of overflow and therefore avoiding the large overhead associated with saturation arithmetic. This gives a shorter code and a faster controller. But there is also a disadvantage. Under normal steady-state operation the error is small and any changes in the control signal will be a relatively small part of the whole dynamic range. A change in the control signal of one quantization step will be amplified, resulting in a large jump. It may also give rise to limit cycles. When a high gain is incorporated in the DSP code, saturation arithmetic must be used on internal calculations.

4. Quantization Effects

Issues related to the interfacing of the DSP to the plant will now be considered. The key questions are related to quantization of A/D- and D/A-converters

Quantization of the Set-Point Value

When implementing the controller the set point should be quantized in the same way as the controller input. That is, the set-point value should either be read through the same, or a similar, A/D-converter as is used for the input signal (if A/D-converter is being used) or quantized internally by using the same resolution as of the A/D-converter. If this is not done there may be an offset or a limit cycle due to the quantization. Figure 4 shows the result of a simulation, when a 6-bits A/D-converter is used for the input signal but the set-point value of 0.455 V is represented with a 16-bit accuracy. The system goes into a limit cycle with a period of 6.77 seconds and an amplitude of 3.8 mV. The reason for this is that the set-point value of 0.455 V can not be represented by the 6-bits A/D-converter. In steady-state the error between the process output and the set-point value will be either 17.5 mV or –13.8 mV. This error will be summed up by the integrator, resulting in a limit cycle.

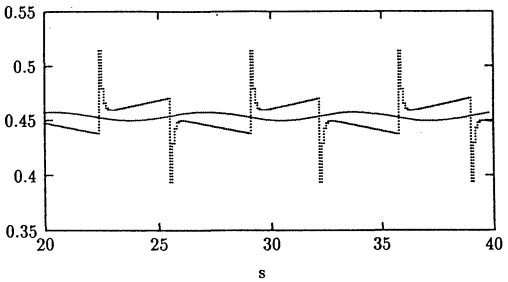


Figure 4. Limit cycles due to high resolution of the set point.

Because the limit cycle is very close to a sinusoid it is reasonable to assume that the period and the amplitude of the limit cycle can be predicted by using describing function analysis. Since the system is in steady-state and the oscillation corresponds to one quantization step of the A/D-converter, we can assume a zero set-point value and model the A/D-converter by a relay nonlinearity centered around zero with the quantization limits +0.00157 and –0.00157. The describing function for this nonlinearity is

$$N(A) = \frac{2q}{\pi a} = \frac{0.0199}{a}$$

where a is the amplitude of the input signal and $q/2$ is half the quantization step. The calculations are simplified if the digital PID-controller is approximated by a continuous-time PI-controller with the transfer function

$$G_c(s) = K + \frac{K}{Ts}$$

where $K = 0.6$ and $T = 2.2$. Possible limit cycle is given by the equation

$$1 + Y_q(A)L(j\omega) = 0$$

Which is equivalent to

$$L(j\omega) = \frac{-1}{N(a)} \tag{13}$$

where L is the loop transfer function of the controller and the process, in cascade, i.e.

$$L(s) = \frac{K + TKs}{Ts(s+1)^4} \tag{14}$$

Since the describing is real-valued, one simply has to find the intersection of $L(j\omega)$ with the negative real axis. When $j\omega$ is substituted for s in Equation (14) we get, after separating the real and the imaginary part

$$L(j\omega) = \frac{K\big(A(\omega) + iB(\omega)\big)}{T(4\omega^4 - 4\omega^2)^2 + (\omega^5 - 6\omega^3 + \omega)^2} \quad (15)$$

where $A = T(\omega^6 - 6\omega^4 + \omega^2) + 4\omega^4 - 4\omega^2$ and $B = T(4\omega^5 - 4\omega^3) - \omega^5 + 6\omega^3 - \omega$. The problem is therefore reduced to finding the frequency where the imaginary part is zero, i.e.

$$7.8\omega^4 - 2.8\omega^2 - 1 = 0 \quad (16)$$

The equation has one positive real root $\omega = 0.7616$, which corresponds to a limit-cycle period of 8.25 s. This is longer than the period $T = 6.77$ s, obtained in the simulation. The amplitude of the limit cycle is then determined by solving Equation (13) for $\omega = 0.7616$, which gives $a = 5.6$ mV. The value $a = 3.8$ mV was obtained in the simulation.

A/D- and D/A-Conversion

If the controller is interfaced to the plant by A/D- and D/A-converters the effect of the resolution of the converters has to be determined. Figure 5 shows the result of one of several simulations where the A/D-converter has a higher resolution than the D/A-converter. A limit cycle was observed in those simulations. Because of the higher resolution of the A/D-converter, the controller produces control signals which are not representable by the D/A-converter. This results in an oscillation over one quantization step of the D/A-converter. This phenomenon can also be predicted by using describing function analysis, where we assume a zero set-point

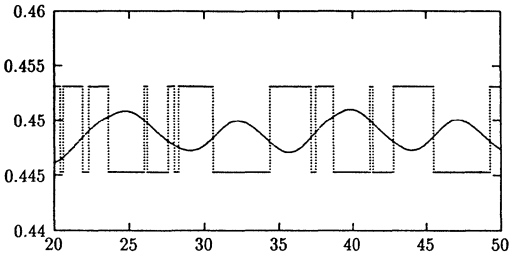


Figure 5. Response with a 10-bit A/D and a 8-bit D/A.

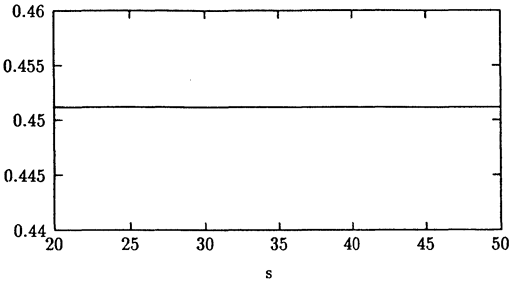


Figure 6. Response with a 8-bit A/D and a 10-bit D/A.

value and the D/A-converter is approximated by a relay. The problem can be avoided by replacing the function f given by Equation (9) by a function that also models the roundoff in the D/A-converter.

Figure 6 shows a good result when an 8-bit A/D-converter and a 10-bit D/A-converter is used when a step input of 0.45 V is applied. These observations indicate that using a D/A-converter with a lower resolution than the A/D-converter may give rise to a limit cycle. It should be emphasized that there are of course many other factors which may be responsible for limit cycles. There are also many other factors that influence the selection of the resolution of the A/D- and D/A-converters, e.g. the required accuracy of the system.

Simulations also showed that a very low resolution (down to 4-bits) of the converters did not have much effect on the step response of the system. The accuracy of the system is, of course, less with low resolution converters. Figure 7 shows the response of the same system when a load disturbance of 0.3 V is introduced at $t = 20$ s.

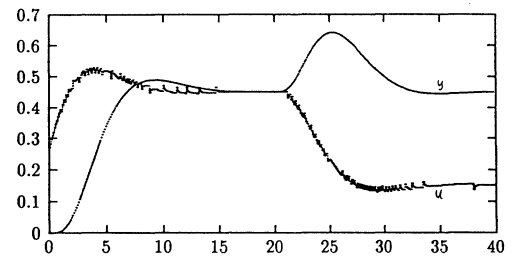


Figure 7. Same as Figure 6 but with a load disturbance.

5. The DSP-Code

To develop and test assembly code of the PID-controller on the Texas Instruments Family of DSPs the Texas Instruments Software Development System (SWDS) was used. This system consists of a PC-board with a TMS320C25 signal processor and PC development environment, which has many features. It is possible to set break-points and single-step through the program. One useful feature is the possibility to specify an input file (or files) to the DSP and to direct the output (or outputs) of the DSP to an output file. This feature makes it easy to test an algorithm, since a predefined input signal can be fed to the controller to test its open loop response.

Programs for PI- and PID-controllers were written for the signal processors TMS32010 and TMS320C25. The complete codes are given in Appendices A, B, C and D. The code for the PID-controller is organized in the following way:

```
INITIALIZE
    load constants from program memory
        to data memory
    clear variables
    load y(n-1) and ysp
    reset external devices (f.ex. analog board)
PID
    wait for input y(n)
    compute derivative (D)
    round off, check for overflow and store D
    compute proportional part (P)
    add D, P and I
    round off, check for overflow and store in v(n)
    compute u(n) from saturation function
    output u(n)
    compute I
    check for overflow and store I
        in double precision
GOTO PID
```

The code for the PI-controller is obtained by deleting the computations of the D-term.

Initialization

After reset the program jumps into the initialization routine. This part disables interrupts, sets overflow mode and loads coefficients from program memory (where they are stored permanently) into data memory. Then the states of the controller are cleared, the set point value (ysp) is read from PA3 and the process output ($y(n-1)$) from PA0. By filling up the y-vector before entering the PID loop a jump due to the derivative is avoided. The program then goes into an infinite loop, to compute the control signal.

PID Calculations

The magnitude of the coefficient b_d of the derivative term is less than 256. To represent it in the DSP it must be scaled by dividing by 256. This can be done by shifts. Before the derivative is stored it is therefore Shifted left by 9 bits (8 bits plus one left shift to account for the extra sign bit which is generated in the multiplication).

The largest proportional gain is 16. The proportional term is therefore divided by 16. It was advantageous also to divide the D and I terms by 16 and restore the signal after the control signal v has been calculated. The same saturation, rounding and shifting can then be applied to both the derivative term and the control signal. Since the derivative must be divided by 16 before it is added to the proportional part, it is advantageous to store a_d divided by 16. A little trick was used to calculate the correct derivative. After $a_d D(t_{k-1})$ has been calculated and stored in the accumulator the term $b_d(y(t_{k-1}) - y(t_k))$ is calculated, and the result is stored in the P register. The value of the P register is then added 16 times to the accumulator to form the correct derivative divided by 16. By doing this in overflow mode, overflow results in saturation of the accumulator. This would not be the case if the value in the accumulator were simply shifted left. With the TMS320C25 adding is easily done using the repeat instruction. After these calculations the derivative is in the accumulator. The proportional term is then added to the accumulator to obtain (P+D)/16. In this way the proportional term does not have to be stored separately.

To obtain the output v, the integral computed previously is divided by 16 by shifting the value right 4 bits. It is then added to P+D in the accumulator. The output then goes through the saturation arithmetic. It is rounded and shifted before it is stored as a 16-bit number. The saturation function f is called to form the final output u.

Since the control signal u depends on the integral from the previous sample, it can be converted to analog form before the integral is updated. This shortens the computational delay between the A/D

Table 2. Cycle count and maximum sampling frequency for PI- and PID-controllers.

DEVICE	PI		PID	
	cycles	KHz	cycles	KHz
TMS32010	94	53	145	34
TMS320C14	94	66	145	43
TMS320C25	89	112	141	70

and D/A-conversions. To avoid integral offset, the integral is computed and stored in double precision. Saturation arithmetic is performed before it is stored, although it is actually not necessary if proper anti-windup is used.

With the chosen method of organizing the calculations the P, D and I terms are added, to form v, with a precision of 27 bits. The terms D and v are then stored with a precision of 16 bits and the integral is calculated and stored with a precision of 31 bits.

Saturation Arithmetic. Before the derivative or the control signal v is stored in memory as a 16-bit value, it must by shifted left by 5 bits, because the signal is divided by 16 in internal calculations and an additional left shift must be performed to account for the extra sign bit generated in the multiplication. The value is rounded and checked for overflow before shifting it. If overflow is detected, the value is replaced by the largest positive or negative number.

Set-Point Value. The set point is read via interrupt. This interrupt is disabled when the control value is computed, but is allowed for a short period, before the next process output is read.

Computation Time

By using the timer on the TMS320C25 it was possible to count the cycles required for one execution of the PID (or PI) loop. To find the number of cycles required for one execution of the TMS32010 (TMS320C14) code, a simple cycle count was done. In all instances it is assumed that the internal memory of the DSPs are used.

Table 2 shows the number of cycles for each controller and the maximum sampling frequency which can be used. From this table we see that the calculation of the derivative consumes a large portion of the total cycles, approximately 50%. The reason for this is that the shifting and saturation arithmetic on the derivative is complicated, because the coefficients of the controller are scaled differently. If the coefficients would all have the same upper limit the same scaling constant could be used and the shifting and saturation arithmetic would be simpler and faster. Table 3 shows how the cycles are divided between different functions of the algorithm. Notice that the division is somewhat arbitrary, because it is not obvious when one operation begins and the other ends. The saturation arithmetic-, rounding- and shifting-function used on the derivative and the output v uses 19 cycles, the saturation arithmetic on the integral uses 10 cycles and the anti-windup function uses 12 cycles.

Table 3. Cycles count for different parts of the PID-controller.

OPERATION	TMS32010 cycles	TMS320C25 cycles
Derivative	9	9
- " - srss	43	45
Proportional	7	7
Integral shifting	12	8
srss on v	23	23
anti-windup	12	12
Integral	15	13
Integral s.a.	10	10
I/O and other	14	14
Total	145	141

srss = saturation, round, shift, store
s.a. = saturation arithmetic

Notice that the code must be modified if K_c and N are to be larger than 16. Also notice that the code can be improved if the parameters of the controller can be limited to smaller ranges. For specific applications, where tighter bounds on parameters and controller states are available, the code can be shortened drastically by removing saturation arithmetic and by simplifying scaling.

It is interesting to note that a crude time estimate, based on the operation count in Table 1, underestimates the computation time by an order of magnitude.

6. Testing

To obtain high quality code it is necessary to develop good testing procedures. The DSP code for the PI and PID controllers were tested by simple laboratory experiments to verify that the controller worked as a proper PID controller. To ensure that the code gives the correct numerical results, the following procedure was introduced. Since a PID

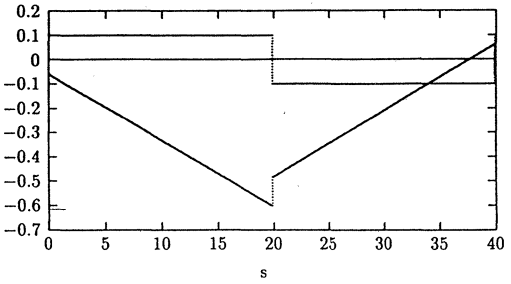


Figure 8. Test of the proportional and integral actions.

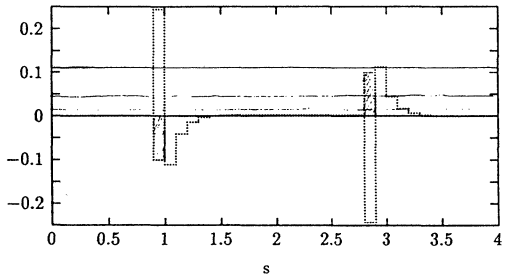


Figure 9. Test of the derivative action.

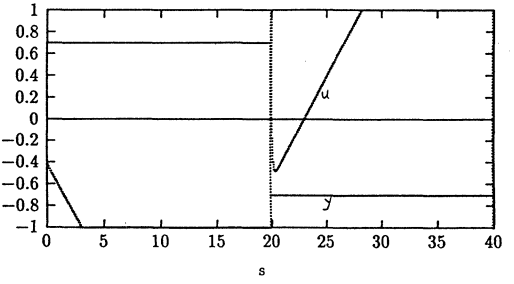


Figure 10. Test of the saturation arithmetic.

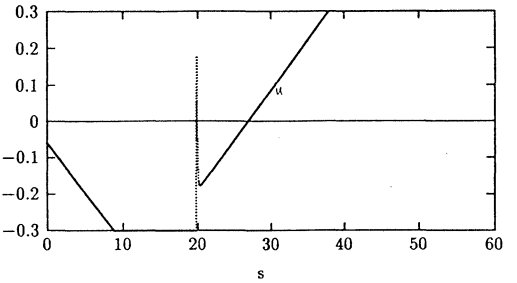


Figure 11. Test of the anti-windup.

controller is a dynamical system, its behavior can be tested by computing its response to given input data with known responses. The test can easily be automated by storing the data in files. This was easily done using the facilities in the Texas Instrument Software Development System. This section describes how the testing was done. The parameters used were $K = 0.6$, $T_d = 0.5$, $T_i = 2.2$, $T_t = 0.5$ and $N = 8$. Parameters a_d, b_d, b_i and b_t were calculated by assuming a sampling period of 0.1 s.

To test proportional and integral action a symmetrical square wave with a period of 40 s and an amplitude of 0.1 V was used as an input sequence. To get a simple case the parameters of the derivative term were set to zero (which is really not necessary, since the derivative dies out very quickly). This sequence can therefore also be used to test a PI controller. Figure 8 shows the input and the resulting output. For a constant input the output of the controller at the time t should be

$$u(t) = \frac{tK_c}{T_i}e + I(0) + K_c e$$

With $I(0) = 0$ the output should be equal to -0.6055 V after 20 seconds. The line $y = -0.6055$ is also drawn in Figure 8 indicating that the proportional and the integral term work properly.

To test the derivative action two impulses, lasting one sampling period, of magnitude -0.1 V and +0.1 V where applied to the input at the time $t = 1$ sec. and $t = 3$ sec. Figure 9 shows the result. The formula for the derivative term is

$$D(t_k) = a_d D(t_{k-1}) + b_d\big(y(t_{k-1}) - y(t_k)\big)$$

If an impulse of magnitude 0.1 V is applied to the derivative we get the sequence: -0.2446, 0.1136, 0.0437, 0.0168, 0.0065,.... The first numbers of this sequence are also plotted on the Figure 9, showing that the derivative action works properly. The small error in the beginning of the second response is due to the integral of the first impulse. This integral is canceled out by the second impulse resulting in a final output equal to zero. To test the saturation arithmetic the amplitude of the input square wave was increased to 0.7 V. Figure 10 shows good results. When the output reaches the limit it is saturated without causing overflow oscillations. Finally, Figure 11 shows the result when the anti-windup reset function is used to limit the output to ± 0.3 V. All versions of the PI- and PID-controller were tested by using these input

sequences. Once a correct set of output files have been obtained one can test modified algorithms simply by comparing the output files, either by plotting the output or by using a file-compare program.

Other testing procedures were also developed using ideas similar to the ones described above.

7. Conclusions

This paper has given algorithms for high quality PI and PID controllers with features like set-point weighting, limitation of derivative gain and anti-windup. It has also been demonstrated how the code can be implemented on a DSP using fix-point calculations. Such an implementation necessarily requires some a priori knowledge of signal and parameter ranges. This means that the code given here only works well in cases that fit the assumptions made.

We have attempted to describe our reasoning in sufficient detail so that the code can be easily adapted to other situations. Some test procedures that we have found useful are also presented. The performance estimates show that PI controller can be executed at 53 kHz on a TMS32010 and at 112 kHz on a TMS320C25.

8. References

Åström, K. J., and T. Hägglund (1988): *Automatic Tuning of PID Controllers*, ISA, Research Triangle Park, NC.

Åström, K. J., and B. Wittenmark (1990): *Computer Controlled Systems – Theory and Design*, Second edition, Prentice-Hall, Englewood Cliffs, NJ.

Deshpande, P. B., and R. H. Ash (1981): *Computer Process Control*, ISA, Research Triangle Park, NC.

Texas Instruments (1986): *Digital Signal Processing Applications with the TMS320 Family – Theory, Algorithms, and Implementations*, Digital Signal Processing, Semiconductor Group.

Texas Instruments (1989a): *TMS320C1x / TMS320C2x – User's Guide*, Digital Signal Processor Products.

Texas Instruments (1989b): *TMS320 Family Development Support – Reference Guide*, Digital Signal Processor Products.

Texas Instruments (1990a): *Digital Signal Processing – Applications with the TMS320 Family*, Application book volume 3, Digital Signal Processor Products.

Texas Instruments (1990b): *TMS320C3x – User's Guide*, Digital Signal Processor Products.

Appendix A: PI-Controller for TMS32010

```
; PI Controller for TMS32010 Version 1.0
; Author: Hermann Steingrimsson
; Date: 3-26-1990
;
;
; RESERVE SPACE IN DATA MEMORY FOR CONSTANTS AND VARIABLES
        .bss    HTE1,1          ;Temporary storages
        .bss    LTE1,1
        .bss    HTE2,1
        .bss    LTE2,1
        .bss    IH,1            ;Integral high
        .bss    IL,1            ;Integral low
        .bss    KC,1            ;Coeff for P
        .bss    KCB,1
        .bss    BI,1            ;Coeff for I
        .bss    BT,1
        .bss    UMAX,1          ;Maximum output
        .bss    UMIN,1          ;Minimum output
        .bss    MODE,1          ;Extra constant
        .bss    CLOCK,1         ;Sampling rate
        .bss    ONE,1           ;One
        .bss    MAXNUM,1        ;Maximum number
        .bss    MINNUM,1        ;Minimum number
DTend   .bss    MINUS,1         ;FFFF
;End of parameters in data memory

        .bss    YN,1            ;y(n)
        .bss    YNM1,1          ;y(n-1)
        .bss    YSP,1           ;y set point
        .bss    UN,1            ;Output
        .bss    VN,1            ;Output before f
        .bss    STA0,1          ;Space to store status register

;Begin program memory

        .sect   "IRUPTS"
        B       START           ;Branch to start of program
        B       ISR             ;Interupt service routine

;Store parameters in program memory

        .data
Ptable  .set    $
        .word   1229,1229,894,6554,9830,-9830,1,1,1,32767,-32768
        .word   -1
Ptend   .set    $-1
SCALE   .set    15
```

```
PI Controller for TMS32010  Version 1.0

;Initialize

        .text
START   DINT                    ;Disable interupts
        NOP
        SOVM                    ;Set overflow mode

;Load coeff from prog. mem to data mem. use TBLR (not BLKP) for 1. generation
;devices

        LARK   AR0,DTend        ;AR0 points to end of data block
        LARK   AR1,Ptend-Ptable ;Counter
        LACK   Ptend            ;Beginning address in program memory
LOAD    LARP   AR0              ;Point to AR0
        TBLR   *-,AR1           ;Move, decr. AR0 and point to AR1
        SUB    ONE              ;Subtract one from accumulator
        BANZ   LOAD             ;AR1 not 0 then decr. AR1 and branch
                                ;=> Coeff loaded into data memory

;Initialize variables

        LDPK   IH               ;Point to correct data page
        ZAC                     ;Clear variables
        SACL   IH
        SACL   IL

        OUT    MODE,PA4         ;Init analog board
        OUT    CLOCK,PA5

WAIT1   BIOZ   GET1             ;Load ysp
        B      WAIT1
GET1    IN     YSP,PA3

WAIT2   BIOZ   GET2             ;Load y(n-1)
        B      WAIT2
GET2    IN     YNM1,PA0

;Begin PI

WAIT    BIOZ   GET              ;Wait for input
        B      WAIT
GET     IN     YN,PA0

        ZAC                     ;Clear accumulator

;P-section

        LT     YSP
        MPY    KCB              ;y(n) * KCB
```

```
        LTA   YN                ;acc = y(n)*KCB - ysp*KC
        MPY   KC
        SPAC

        SACH  HTE1              ;Store P temporarily
        SACL  LTE1

        ZALH  IH                ;Shift integral right 4
        ADDS  IL
        SACH  HTE2
        SACL  LTE2
        LAC   LTE2,12
        SACH  LTE2
        LAC   MINUS,12
        XOR   MINUS
        AND   LTE2
        ADD   HTE2,12           ;I in acc rigth shifted 4

        ADDS  LTE1              ;Add P to acc to form P + I
        ADDH  HTE1

        LARK  AR0,VN            ;Point AR0 to VN
        LARP  AR0
        CALL  ROUOF4            ;Round off and overflow check

        CALL  FUNCT             ;Actuator saturation function
        OUT   UN,PA1            ;Output control signal

;I-section

        ZAC
        LT    YSP
        MPY   BI

        LTA   YN
        MPY   BI
        SPAC

        LT    UN
        MPY   BT

        LTA   VN
        MPY   BT
        SPAC

        ADDS  IL                ;Add old I with double precision
        ADDH  IH
```

```
PI Controller for TMS32010  Version 1.0

        SACH  IH                ;Store integral
        SACL  IL

        BLZ   INEG              ;Overflow check (10 instr. cycles)
        SUB   MAXNUM,SCALE      ;Subtract maximum pos. number
        BLEZ  OUT4              ;If acc <= 0 then no overflow
        LAC   MAXNUM,SCALE      ;else store maximum number
        SACH  IH
        SACL  IL
        B     OUT5

INEG    SUB   MINNUM,SCALE      ;Subtract maximum neg number
        BGEZ  OUT4              ;If acc >= 0 then no overflow
        LAC   MINNUM,SCALE      ;else store minimum number
        SACH  IH
        SACL  IL
        B     OUT5

OUT4    NOP
        NOP
        NOP
        NOP
        NOP
OUT5    EINT                    ;Enable interupt
        NOP
        NOP
        DINT                    ;Disable interupt
        B     WAIT              ;Loop again

;Rounding and overflow function (11 cycles)

ROUOF4  BLZ   RNEG              ;Check if number negative
        ADD   ONE,SCALE-5       ;Round
        SACH  HTE1              ;Store value
        SACL  LTE1
        SUB   MAXNUM,SCALE-4    ;Subtract scaled max pos number
        BLEZ  RNO               ;If acc <= 0 then no overflow
        ZALS  MAXNUM            ;else store max num
        SACL  *
        RET

RNEG    ADD   ONE,SCALE-5       ;Round
        SACH  HTE1              ;Store value
        SACL  LTE1
        SUB   MINNUM,SCALE-4    ;Subtract scaled min neg number
        BGEZ  RNO               ;If acc >= 0 then no overflow
        ZALS  MINNUM            ;else store min neg number
        SACL  *
        RET
```

```
PI Controller for TMS32010  Version 1.0


RNO     ZALH  HTE1              ;Shift number left 4 before store
        ADDS  LTE1
        SACH  HTE1,4
        SACL  LTE1
        ZALH  LTE1
        SACH  LTE1,4
        ZALH  HTE1
        ADDS  LTE1
        SACH  *,16-SCALE
        RET

;Saturation function (14 instr. cycles)

FUNCT   ZALH  VN                ;Load VN
        SUBH  UMIN
        BLZ   LOWER1            ;Branch if v < umin
        ZALH  VN
        SUBH  UMAX
        BLZ   SAME              ;Branch if v < umax
        B     HIGHER            ;v >= umax

LOWER1  ZALH  UMIN
        SACH  UN                ;u = umin
        NOP                     ;Always same time
        NOP
        NOP
        NOP
        NOP
        NOP
        RET

SAME    ZALH  VN
        SACH  UN                ;u = v
        NOP
        NOP
        RET

HIGHER  ZALH  UMAX
        SACH  UN                ;u = umax
        RET

;Interupt service routine. To read set point value

ISR     SST   STA0              ;Save status
        IN    YSP,PA3           ;Load ysp
        LST   STA0              ;Restore status
        RET                     ;Return
        .end
```

Appendix B: PI-Controller for TMS320C25

```
; PI Controller for TMS320C25 Version 1.0
; Author: Hermann Steingrimsson
; Date: 3-26-1990
;
;
; RESERVE SPACE IN DATA MEMORY FOR CONSTANTS AND VARIABLES
        .bss    HTE1,1          ;Temporary storages
        .bss    LTE1,1
        .bss    HTE2,1
        .bss    LTE2,1
        .bss    IH,1            ;Integral high
        .bss    IL,1            ;Integral low
        .bss    KC,1            ;Coeff for P
        .bss    KCB,1
        .bss    BI,1            ;Coeff for I
        .bss    BT,1
        .bss    UMAX,1          ;Maximum output
        .bss    UMIN,1          ;Minimum output
        .bss    MODE,1          ;Extra constant
        .bss    CLOCK,1         ;Sampling rate
        .bss    ONE,1           ;One
        .bss    MAXNUM,1        ;Maximum number
        .bss    MINNUM,1        ;Minimum number
DTend   .bss    MINUS,1         ;FFFF
;End of parameters in data memory

        .bss    YN,1            ;y(n)
        .bss    YNM1,1          ;y(n-1)
        .bss    YSP,1           ;y set point
        .bss    UN,1            ;Output
        .bss    VN,1            ;Output before f
        .bss    STA0,1          ;Space to store status register
        .bss    STA1,1

;Begin program memory

        .sect   "IRUPTS"
        B       START           ;Branch to start of program
        B       ISR             ;Interupt service routine

;Store parameters in program memory

        .data
Ptable  .set    $
        .word   1229,1229,894,6554,9830,-9830,1,1,1,32767,-32768
        .word   -1
Ptend   .set    $-1
SCALE   .set    15
```

```
PI Controller for TMS320C25  Version 1.0


;Initialize

        .text
START   DINT                    ;Disable interupts
        NOP
        SOVM                    ;Set overflow mode
        SSXM                    ;Set sign-extension mode
        SPM   0                 ;No shifting from P register

;Load coeff from prog. mem to data mem.

        LRLK   AR0,DTend        ;AR0 points to end of data block
        LARK   AR1,Ptend-Ptable ;Counter
        LALK   Ptend            ;Beginning address in program memory
LOAD    LARP  AR0               ;Point to AR0
        TBLR  *-,AR1            ;Move, decr. AR0 and point to AR1
        SUBK  1                 ;Subtract one from accumulator
        BANZ  LOAD              ;AR1 not 0 then decr. AR1 and branch
                                ;=> Coeff loaded into data memory

;Initialize variables

        LDPK  IH                ;Point to correct data page
        ZAC                     ;Clear variables
        SACL  IH
        SACL  IL

        OUT   MODE,PA4          ;Init analog board
        OUT   CLOCK,PA5

;WAIT1  BIOZ  GET1              ;Load ysp
;       B     WAIT1
GET1    IN    YSP,PA3

;Begin PID

;WAIT   BIOZ  GET               ;Wait for input
;       B     WAIT
WAIT    IN    YN,PA0           ;Change WAIT to GET when ; are removed

;P-section

        LT    YSP
        MPY   KCB               ;y(n) * KCB

        LTP   YN                ;acc = y(n)*KCB - ysp*KC
        MPY   KC
        SPAC
```

```
        SACH  HTE1              ;Store P
        SACL  LTE1

        ZALH  IH                ;Shift integral right 4
        ADDS  IL                ;because coeff of P where divided by 16
        SFR
        SFR
        SFR
        SFR

        ADDS  LTE1              ;Add P to acc to form P + I
        ADDH  HTE1

        LRLK  AR0,VN            ;Point AR0 to VN
        LARP  AR0
        CALL  ROUOF4            ;Round off and overflow check

        CALL  FUNCT             ;Actuator saturation function
        OUT   UN,PA1            ;Output control signal

;I-section

        LT    YSP
        MPY   BI

        LTP   YN
        MPY   BI

        LTS   UN
        MPY   BT

        LTA   VN
        MPY   BT
        SPAC

        ADDS  IL                ;Add old I with double precision
        ADDH  IH

        SACH  IH                ;Store integral
        SACL  IL

        BLZ   INEG              ;Overflow check (10 instr. cycles)
        SUB   MAXNUM,SCALE      ;Subtract maximum pos. number
        BLEZ  OUT4              ;If acc <= 0 then no overflow
        LAC   MAXNUM,SCALE      ;else store maximum number
        SACH  IH
        SACL  IL
        B     OUT5
```

```
PI Controller for TMS320C25  Version 1.0


INEG    SUB   MINNUM,SCALE      ;Subtract maximum neg number
        BGEZ  OUT4              ;If acc >= 0 then no overflow
        LAC   MINNUM,SCALE      ;else store minimum number
        SACH  IH
        SACL  IL
        B     OUT5

OUT4    NOP
        NOP
        NOP
        NOP
        NOP
OUT5    EINT                    ;Enable interupt
        NOP
        NOP
        DINT                    ;Disable interupt
        B     WAIT              ;Loop again

; Rounding, overflow and shifting function (13 cycles)

ROUOF4  BLZ   RNEG              ;Check if number negative
        ADD   ONE,SCALE-5       ;Round
        SACH  HTE1              ;Store value
        SACL  LTE1
        SUB   MAXNUM,SCALE-4    ;Subtract scaled max pos number
        BLEZ  RNO               ;If acc <= 0 then no overflow
        ZALS  MAXNUM            ;else store max num
        SACL  *
        NOP
        RET

RNEG    ADD   ONE,SCALE-5       ;Round
        SACH  HTE1              ;Store value
        SACL  LTE1
        SUB   MINNUM,SCALE-4    ;Subtract scaled min neg number
        BGEZ  RNO               ;If acc >= 0 then no overflow
        ZALS  MINNUM            ;else store min neg number
        SACL  *
        NOP
        RET

RNO     ZALH  HTE1              ;Shift number left 4+1 before store
        ADDS  LTE1
        SACH  *,5
        RET

;Saturation function (12 instr. cycles)
```

```
PI Controller for TMS320C25  Version 1.0

FUNCT   ZALH  VN              ;Load VN
        SUBH  UMIN
        BLZ   LOWER1          ;Branch if v < umin
        ZALH  VN
        SUBH  UMAX
        BLZ   SAME            ;Branch if v < umax
        ZALH  UMAX            ;v >= umax
        SACH  UN              ;u = umax
        RET

LOWER1  ZALH  UMIN
        SACH  UN              ;u = umin
        NOP                   ;Always same time
        NOP
        NOP
        NOP
        RET

SAME    ZALH  VN
        SACH  UN              ;u = v
        RET


;Interupt service routine. To read set point value

ISR     SST   STA0            ;Save status
        SST1  STA1
        IN    YSP,PA3         ;Load ysp
        LST   STA0            ;Restore status
        LST1  STA1
        RET                   ;Return
        .end
```

Appendix C: PID-Controller for TMS32010

```
; PID Controller for TMS32010 Version 1.0
; Roundoff Corrected
; Hermann Steingrimsson
; Date: 3-26-1990
;
;
; ad and Kc must be divided by 16 before stored
; bd must be divided by 256 before storage
;
; RESERVE SPACE IN DATA MEMORY FOR CONSTANTS AND VARIABLES
        .bss    HTE1,1          ;Temporary storages
        .bss    LTE1,1
        .bss    HTE2,1
        .bss    LTE2,1
        .bss    IH,1            ;Integral high
        .bss    IL,1            ;Integral low
        .bss    DH,1            ;Derivative high
        .bss    KC,1            ;Coeff for P
        .bss    KCB,1
        .bss    BI,1            ;Coeff for I
        .bss    BT,1
        .bss    BD,1            ;Coeff for D
        .bss    AD,1
        .bss    UMAX,1          ;Maximum output
        .bss    UMIN,1          ;Minimum output
        .bss    MODE,1          ;Extra constant
        .bss    CLOCK,1         ;Sampling rate
        .bss    ONE,1           ;One
        .bss    MAXNUM,1        ;Maximum number
        .bss    MINNUM,1        ;Minimum number
DTend   .bss    MINUS,1         ;FFFF
;End of parameters in data memory

        .bss    YN,1            ;y(n)
        .bss    YNM1,1          ;y(n-1)
        .bss    YSP,1           ;y set point
        .bss    UN,1            ;Output
        .bss    VN,1            ;Output before f
        .bss    STAO,1          ;Space to store status register

;Begin program memory

        .sect   "IRUPTS"
        B       START           ;Branch to start of program
        B       ISR             ;Interupt service routine

;Store parameters in program memory
```

```
PID Controller for TMS32010  Version 1.0

        .data
Ptable  .set    $
        .word   1229,1229,894,6554,236,788,9830,-9830,1,1,1,32767,-32768
        .word   -1
Ptend   .set    $-1
SCALE   .set    15

;Initialize

        .text
START   DINT                    ;Disable interupts
        NOP
        SOVM                    ;Set overflow mode

;Load coeff from prog. mem to data mem. use TBLR (not BLKP) for 1. generation
;devices

        LARK   AR0,DTend        ;AR0 points to end of data block
        LARK   AR1,Ptend-Ptable ;Counter
        LACK    Ptend           ;Beginning address in program memory
LOAD    LARP  AR0               ;Point to AR0
        TBLR  *-,AR1            ;Move, decr. AR0 and point to AR1
        SUB   ONE               ;Subtract one from accumulator
        BANZ  LOAD              ;AR1 not 0 then decr. AR1 and branch
                                ;=> Coeff loaded into data memory

;Initialize variables

        LDPK  IH                ;Point to correct data page
        ZAC                     ;Clear variables
        SACL  IH
        SACL  IL
        SACL  DH

        OUT  MODE,PA4           ;Init analog board
        OUT  CLOCK,PA5

WAIT1   BIOZ  GET1              ;Load ysp
        B     WAIT1
GET1    IN    YSP,PA3

WAIT2   BIOZ  GET2              ;Load y(n-1)
        B     WAIT2
GET2    IN    YNM1,PA0

;Begin PID

WAIT    BIOZ  GET               ;Wait for input
        B     WAIT
```

```
PID Controller for TMS32010  Version 1.0

GET     IN    YN,PA0            ;Change WAIT to GET when ; are removed

;D-section

        ZALH  YNM1              ;y(n-1) - y(n)
        SUBH  YN
        SACH  HTE1              ;Store difference
        DMOV  YN                ;Copy YN into YNM1

        LT    DH                ;ad*D (ad was divided by 16)
        MPY   AD
        PAC

        LT    HTE1              ;difference * bd
        MPY   BD

        APAC    ;1              ;Since bd was divided by 256, bd*diff is
        APAC    ;2              ;added 16 times to the accumulator to
        APAC    ;3              ;form D divided by 16. By doing this the
        APAC    ;4              ;overflow mode will take care of overflow
        APAC    ;5
        APAC    ;6
        APAC    ;7
        APAC    ;8
        APAC    ;9
        APAC    ;10
        APAC    ;11
        APAC    ;12
        APAC    ;13
        APAC    ;14
        APAC    ;15
        APAC    ;16


        SACH  HTE2              ;Store derivative
        SACL  LTE2
        LARK  AR0,DH            ;Point to DH
        LARP  AR0
        CALL  ROUOF4            ;Check for overfl. shift and store
        ZALH  HTE2              ;Restore the derivative
        ADDS  LTE2

;P-section

        LT    YSP
        MPY   KCB               ;y(n) * KCB

        LTA   YN                ;acc = y(n)*KCB - ysp*KC
        MPY   KC
```

```
PID Controller for TMS32010  Version 1.0

        SPAC

        SACH  HTE1              ;Store P + D
        SACL  LTE1

        ZALH  IH                ;Shift integral right 4
        ADDS  IL
        SACH  HTE2
        SACL  LTE2
        LAC   LTE2,12
        SACH  LTE2
        LAC   MINUS,12
        XOR   MINUS
        AND   LTE2
        ADD   HTE2,12           ;I in acc right shifted 4

        ADDS  LTE1              ;Add P + I to acc to form P + I + D
        ADDH  HTE1

        LARK  AR0,VN            ;Point AR0 to VN
        LARP  AR0
        CALL  ROUOF4            ;Round off and overflow check

        CALL  FUNCT             ;Actuator saturation function
        OUT   UN,PA1            ;Output control signal

;I-section

        ZAC
        LT    YSP
        MPY   BI

        LTA   YN
        MPY   BI
        SPAC

        LT    UN
        MPY   BT

        LTA   VN
        MPY   BT
        SPAC

        ADDS  IL                ;Add old I with double precision
        ADDH  IH

        SACH  IH                ;Store integral
        SACL  IL
```

```
        BLZ   INEG              ;Overflow check (10 instr. cycles)
        SUB   MAXNUM,SCALE      ;Subtract maximum pos. number
        BLEZ  OUT4              ;If acc <= 0 then no overflow
        LAC   MAXNUM,SCALE      ;else store maximum number
        SACH  IH
        SACL  IL
        B     OUT5

INEG    SUB   MINNUM,SCALE      ;Subtract maximum neg number
        BGEZ  OUT4              ;If acc >= 0 then no overflow
        LAC   MINNUM,SCALE      ;else store minimum number
        SACH  IH
        SACL  IL
        B     OUT5

OUT4    NOP
        NOP
        NOP
        NOP
        NOP
OUT5    EINT                    ;Enable interupt
        NOP
        NOP
        DINT                    ;Disable interupt
        B     WAIT              ;Loop again

; Rounding, overflow and shifting function (19 cycles)

ROUOF4  BLZ   RNEG              ;Check if number negative
        ADD   ONE,SCALE-5       ;Round
        SACH  HTE1              ;Store value
        SACL  LTE1
        SUB   MAXNUM,SCALE-4    ;Subtract scaled max pos number
        BLEZ  RNO               ;If acc <= 0 then no overflow
        ZALS  MAXNUM            ;else store max num
        SACL  *
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        RET

RNEG    ADD   ONE,SCALE-5       ;Round
        SACH  HTE1              ;Store value
        SACL  LTE1
        SUB   MINNUM,SCALE-4    ;Subtract scaled min neg number
```

```
        BGEZ  RNO               ;If acc >= 0 then no overflow
        ZALS  MINNUM            ;else store min neg number
        SACL  *
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        RET

RNO     ZALH  HTE1              ;Shift number left 4 before store
        ADDS  LTE1
        SACH  HTE1,4
        SACL  LTE1
        ZALH  LTE1
        SACH  LTE1,4
        ZALH  HTE1
        ADDS  LTE1
        SACH  *,16-SCALE
        RET

;Saturation function (12 instr. cycles)

FUNCT   ZALH  VN                ;Load VN
        SUBH  UMIN
        BLZ   LOWER1            ;Branch if v < umin
        ZALH  VN
        SUBH  UMAX
        BLZ   SAME              ;Branch if v < umax
        ZALH  UMAX              ;v >= umax
        SACH  UN                ;u = umax
        RET

LOWER1  ZALH  UMIN
        SACH  UN                ;u = umin
        NOP                     ;Always same time
        NOP
        NOP
        NOP
        RET

SAME    ZALH  VN
        SACH  UN                ;u = v
        RET


;Interupt service routine. To read set point value
```

```
ISR     SST   STA0              ;Save status
        IN    YSP,PA3           ;Load ysp
        LST   STA0              ;Restore status
        RET                     ;Return
        .end
```

Appendix D: PID-Controller for TMS320C25

```
; PID Controller for TMS320C25
; Roundoff Corrected
; Author: Hermann Steingrimsson
; Date: 3-26-1990
;
;
; ad and Kc must be divided by 16 before stored
; bd must be divided by 256 before storage
;
; RESERVE SPACE IN DATA MEMORY FOR CONSTANTS AND VARIABLES
        .bss    HTE1,1          ;Temporary storages
        .bss    LTE1,1
        .bss    HTE2,1
        .bss    LTE2,1
        .bss    IH,1            ;Integral high
        .bss    IL,1            ;Integral low
        .bss    DH,1            ;Derivative high
DTbeg   .bss    KC,1            ;Coeff for P
        .bss    KCB,1
        .bss    BI,1            ;Coeff for I
        .bss    BT,1
        .bss    BD,1            ;Coeff for D
        .bss    AD,1
        .bss    UMAX,1          ;Maximum output
        .bss    UMIN,1          ;Minimum output
        .bss    MODE,1          ;Extra constant
        .bss    CLOCK,1         ;Sampling rate
        .bss    ONE,1           ;One
        .bss    MAXNUM,1        ;Maximum number
        .bss    MINNUM,1        ;Minimum number
DTend   .bss    MINUS,1         ;FFFF
;End of parameters in data memory

        .bss    YN,1            ;y(n)
        .bss    YNM1,1          ;y(n-1)
        .bss    YSP,1           ;y set point
        .bss    UN,1            ;Output
        .bss    VN,1            ;Output before f
        .bss    STA0,1          ;Space to store status register
        .bss    STA1,1

;Begin program memory

        .sect   "IRUPTS"
        B       START           ;Branch to start of program
        B       ISR             ;Interupt service routine

;Store parameters in program memory
```

```
        .data
Ptable  .set    $
        .word   1229,1229,894,6554,236,788,9830,-9830,1,1,1,32767,-32768
        .word   -1
Ptend   .set    $-1
SCALE   .set    15

;Initialize

        .text
START   DINT                    ;Disable interupts
        NOP
        SOVM                    ;Set overflow mode
        SSXM                    ;Set sign-extension mode
        SPM   0                 ;No shifting from P register

;Load coeff from prog. mem to data mem. use BLKP

        LRLK  AR0,DTbeg         ;AR0 points to end of data block
        LARP  AR0
        RPTK  Ptend-Ptable      ;Set up counter
        BLKP  Ptable,*+         ;Move data
                                ;=> Coeff loaded into data memory

;Initialize variables

        LDPK  IH                ;Point to correct data page
        ZAC                     ;Clear variables
        SACL  IH
        SACL  IL
        SACL  DH

        OUT  MODE,PA4           ;Init analog board
        OUT  CLOCK,PA5

WAIT1   BIOZ  GET1              ;Load ysp
        B     WAIT1
GET1    IN    YSP,PA3

WAIT2   BIOZ  GET2              ;Load y(n-1)
        B     WAIT2
GET2    IN    YNM1,PA0

;Begin PID

WAIT    BIOZ  GET               ;Wait for input
        B     WAIT
GET     IN    YN,PA0            ;Change WAIT to GET when ; are removed
```

```
PID Controller for TMS320C25  Version 1.0


;D-section

        ZALH  YNM1              ;y(n-1) - y(n)
        SUBH  YN
        SACH  HTE1              ;Store difference
        DMOV  YN                ;Copy YN into YNM1

        LT    DH                ;ad*D (ad was divided by 16)
        MPY   AD

        LTP   HTE1              ;difference * bd, and store previous product
        MPY   BD

        RPTK  15                ;Since bd was divided by 256, bd*diff is
        APAC                    ;added 16 times to the accumulator to
                                ;form D divided by 16. By doing this the
                                ;overflow mode will take care of overflow

        SACH  HTE2              ;Store derivative
        SACL  LTE2
        LRLK  AR0,DH            ;Point to DH
        LARP  AR0
        CALL  ROUOF4            ;Check for overfl. shift and store
        ZALH  HTE2              ;Restore the derivative
        ADDS  LTE2

;P-section

        LT    YSP
        MPY   KCB               ;y(n) * KCB

        LTA   YN                ;acc = y(n)*KCB - ysp*KC
        MPY   KC
        SPAC

        SACH  HTE1              ;Store P + D
        SACL  LTE1

;P + D are now divided by 16 => shift integral right 4 bits before adding
;to P + D

        ZALH  IH                ;Shift integral right 4
        ADDS  IL
        SFR
        SFR
        SFR
        SFR
```

```
        ADDS  LTE1              ;Add P + I to acc to form P + I + D
        ADDH  HTE1

        LRLK  AR0,VN            ;Point AR0 to VN
        LARP  AR0
        CALL  ROUOF4            ;Round off and overflow check

        CALL  FUNCT             ;Actuator saturation function
        OUT   UN,PA1            ;Output control signal

;I-section

        LT    YSP
        MPY   BI

        LTP   YN
        MPY   BI

        LTS   UN
        MPY   BT

        LTA   VN
        MPY   BT
        SPAC

        ADDS  IL                ;Add old I with double precision
        ADDH  IH

        SACH  IH                ;Store integral
        SACL  IL

        BLZ   INEG              ;Overflow check (10 instr. cycles)
        SUB   MAXNUM,SCALE      ;Subtract maximum pos. number
        BLEZ  OUT4              ;If acc <= 0 then no overflow
        LAC   MAXNUM,SCALE      ;else store maximum number
        SACH  IH
        SACL  IL
        B     OUT5

INEG    SUB   MINNUM,SCALE      ;Subtract maximum neg number
        BGEZ  OUT4              ;If acc >= 0 then no overflow
        LAC   MINNUM,SCALE      ;else store minimum number
        SACH  IH
        SACL  IL
        B     OUT5

OUT4    NOP
        NOP
        NOP
```

```
        NOP
        NOP
OUT5    EINT                    ;Enable interupt
        NOP
        NOP
        DINT                    ;Disable interupt
        B     WAIT              ;Loop again

; Rounding, overflow and shifting function (19 cycles)

ROUOF4  BLZ   RNEG              ;Check if number negative
        ADD   ONE,SCALE-5       ;Round
        SACH  HTE1              ;Store value
        SACL  LTE1
        SUB   MAXNUM,SCALE-4    ;Subtract scaled max pos number
        BLEZ  RNO               ;If acc <= 0 then no overflow
        ZALS  MAXNUM            ;else store max num
        SACL  *
        NOP
        RET

RNEG    ADD   ONE,SCALE-5       ;Round
        SACH  HTE1              ;Store value
        SACL  LTE1
        SUB   MINNUM,SCALE-4    ;Subtract scaled min neg number
        BGEZ  RNO               ;If acc >= 0 then no overflow
        ZALS  MINNUM            ;else store min neg number
        SACL  *
        NOP
        RET

RNO     ZALH  HTE1              ;Shift number left 4 before store
        ADDS  LTE1
        SACH  *,5               ;+1 shift because of sign
        RET

;Saturation function (12 instr. cycles)

FUNCT   ZALH  VN                ;Load VN
        SUBH  UMIN
        BLZ   LOWER1            ;Branch if v < umin
        ZALH  VN
        SUBH  UMAX
        BLZ   SAME              ;Branch if v < umax
        ZALH  UMAX              ;v >= umax
        SACH  UN                ;u = umax
        RET

LOWER1  ZALH  UMIN
```

```
        SACH  UN                ;u = umin
        NOP                     ;Always same time
        NOP
        NOP
        NOP
        RET

SAME    ZALH  VN
        SACH  UN                ;u = v
        RET


;Interupt service routine. To read set point value

ISR     SST   STA0              ;Save status
        SST1  STA1
        IN    YSP,PA3           ;Load ysp
        LST   STA0              ;Restore status
        LST1  STA1
        RET                     ;Return
        .endæ
```

DSP Implementation of a Disk Drive Controller[†]

Hermann Steingrímsson
Graduate School of Business
University of Wisconsin
Madison, Wisconsin, USA

Karl Johan Åström
Department of Automatic Control
Lund Institute of Technology
Lund, Sweden

1. Introduction

The purpose of this paper is to study implementation of a controller based on state estimation and feedback from estimated states on a digital signal processor. Design of a control system for a disk drive is chosen as an example. The controller is implemented on a DSP that does not have floating point hardware. The control problem is described in Section 2, which also describes mathematical models of different complexity. Design of a controller is discussed in Section 3. This section contains a derivation of a continuous time controller and a discrete time controller. The continuous controller is used to choose design parameters and to estimate orders of magnitude. The discrete time controller is the algorithm implemented on the DSP. The section on control design also contains a discussion of design trade-offs. Implementation of the controller on a DSP is discussed in Section 4. Scaling of parameters and states is a major issue. An outline of the code is given. The complete code is listed in the appendix. Testing of the code is described in Section 5 and the paper ends with conclusions and references.

2. Disk Drive Control

Modern disk drive use fast voice coil actuators to position the magnetic heads on a track and to keep them on track under closed loop control. The task of the position control system is twofold: to position the heads over a desired track and to keep it there. The first task is a servo problem whereas the second task is a regulation problem. This paper treats the regulation problem.

Two methods are currently used for feedback measurements. In a *dedicated servo* an entire surface is used for position information, that could have been used for data. In an *embedded servo* the position information are embedded into the data track at the beginning of each sector, instead of using a separate surface. It is also possible to have dual layers so that the servo information is on a layer below the data layer.

The advantage of the dedicated servo is that position information available continuously. With a dedicated servo it is therefore possible to use a controller with a high bandwidth. In an embedded servo, position information is only obtained at a sector boundary. This limits the track following bandwidth and results in longer seek times, and more sluggish track following. A dedicated servo uses an extra surface for the position information. Thermal differences between the position surface and the data surfaces also give rise to errors.

Linear or rotary actuators with a permanent magnet and a voice coil are used to move the head across the tracks. The arm is ideally a rigid body which can be modeled as a double integrator. The large accelerations will, however, excite resonant modes. This makes it difficult to achieve a high bandwidth for positioning and track following.

Analog controllers have been used for servos. They contain amplifiers, compensation networks, notch filters, switches and passive components. The parameters of the analog components change with temperature and component aging can result in deteriorated performance of the servo.

There are several advantages in using a digital servo. Components having drift and aging are avoided, the number of components can be reduced and servo performance can be increased. a digital servo will, however, require high sampling rates. This makes a microcontroller less suitable. The inexpensive DSP's offer computational power an or-

[†] Part of this work was done when the first author was visiting professor and the second author a graduate student at the University of Texas at Austin.

der of magnitude greater than the microcontrollers and some, like the TMS320C14, do also have the hardware for input-output similar to a micro controller. Such components are ideally suited for implementation of fast servos of the type used in disk drives.

Position Detector

The head/track misalignment is the only information available to the controller. Control thus has to be based on error feedback. The position detector generates a voltage which is proportional to the misalignment of the head and track. The operating range is 23μm and the output voltage is in the range 0-5 V. After A/D-conversion one unit in the processor corresponds to a track/head misalignment of 11.5μm . The useful track width is approximately 4.3μm.

Control Signal

The D/A-converter generates a voltage in the range ± 5 V. This voltage is amplified by an amplifier which generates a current. The current passes through the voice coil and generates a torque to move the arm.

Physical Constants of the Drive

The drive system has the following parameters:
Pivot to head radius
R: 0.08 m
Power amplifier gain
K_{pa}: 0.5 A/V
Torque constant of the actuator
K_t: 0.09 Nm/A
Total moment of inertia
J: $50 \cdot 10^{-6}$ Kgm^2

Mathematical Model

A mathematical model describing the position of the arm as a function of the current trough the coil is a double integrator

$$J\frac{d^2\varphi}{dt^2} = K_t I \tag{1}$$

where φ is the angle of the arm. The transfer function from voltage u to arm position y is

$$G_p(s) = \frac{Y(s)}{U(s)} = \frac{K_p}{s^2} \tag{2}$$

where

$$K_p = K_{pa}K_t R/J \approx 72\text{m/s}^2\text{V}$$

The model given by Equation (1) neglects the fact that the arm has compliance. If this is considered, the plant transfer function becomes $G_{p1} = G_p G_1$, where

$$G_1 = \frac{\omega_1^2}{s^2 + 2\zeta\omega_1 s + \omega_1^2} \tag{3}$$

or $G_{p2} = G_{p1}G_2 = G_p G_1 G_2$, where

$$G_2(s) = \frac{s^2 + 2\zeta_3\omega_3 s + \omega_3^2}{s^2 + 2\zeta_1\omega_2 s + \omega_2^2} \tag{4}$$

Typical values of ω and ω_1 are 2 KHz and 3 KHz. The model given by Equation (1) is a good approximation of low frequencies. Because of the resonances this model does, however, not describe the system well at frequencies approaching one kHz. For those frequencies it is necessary to use models like (3) and (4) or even more complicated models.

Disturbances

The major disturbances acting on the system are low frequency load disturbance and a periodic tracking error. Load disturbances are due to the torque from the wires connected to the arm. This torque is almost constant at a given track, but it changes with the track. It may also change with temperature. The second disturbance is due to the eccentricity of the disk which translates into a periodical tracking error. Since the amplitude of this error is small, the disturbance can be approximated by a sinusoid with the rotational frequency of the disk. By introducing the state x_3, the load disturbance can be added to equation (1), giving

$$\begin{aligned}\frac{d^2\varphi}{dt^2} &= K_p u + x_3 \\ \frac{dx_3}{dt} &= 0\end{aligned} \tag{5}$$

3. Controller Design

Control algorithms for the disk drive will be derived in this section. A continuous time controller for the simple rigid body model is first derived.

This derivation gives insight into the control problem and guide lines for choosing the design parameters. The controller is obtained using a straightforward pole-placement method. See [Åström and Wittenmark, 1990]. A discrete time algorithm is then derived. This algorithm is the basis for the DSP implementation.

A state-space model of (5) is

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) \end{aligned} \tag{6}$$

where

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 0 \\ K_p \\ 0 \end{pmatrix}, \quad y = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}$$

and
- x_1: position [m]
- x_2: velocity [m/s]
- x_3: torque [Nm]
- K_p: gain [m/s^{2}V]
- u: control signal [V]

Continuous-Time Controller

It is easily verified that the states x_1 and x_2 of the model (6) are controllable. The disturbance state x_3 is naturally not controllable. All the states of the system are observable. A controller based on a state-feedback and an observer can therefore be designed.

State Feedback. The controller will now be derived in the straightforward manner. See [Åström and Wittenmark, 1990]. It is first assumed that all states are measurable. The state feedback

$$u = -Lx = -l_1x_1 - l_2x_2 - l_3x_3 \tag{7}$$

gives the closed-loop system

$$\dot{x}(t) = (A - BL)x(t) \tag{8}$$

The gains l_1 and l_2 are selected such that the characteristic polynomial of the closed loop system becomes

$$s(s^2 + 2\zeta_p\omega_p s + \omega_p^2) \tag{9}$$

Notice that the zero at the origin is due to the uncontrollable disturbance mode. The characteristic polynomial of (8) is

$$s\big(s^2 + K_p l_2 s + K_p l_1\big)$$

To obtain (9) the feedback gains l_1 and l_2 should thus be chosen as

$$\begin{aligned} l_1 &= \omega_p^2/K_p \\ l_2 &= 2\zeta_p\omega_p/K_p \end{aligned} \tag{10}$$

The gain l_3 is chosen to give perfect disturbance cancellation, i.e.

$$l_3 = 1/K_p \tag{11}$$

The control law (7) can be interpreted as a feedback from the process states x_1 and x_2 and a feedforward from the disturbance state x_3.

State Observer. A state observer is given by

$$\dot{\hat{x}}(t) = A\hat{x}(t) + Bu(t) + K\big(y(t) - C\hat{x}(t)\big) \tag{12}$$

where $\hat{x}$ is the estimate of the state vector x. The reconstruction error $\tilde{x} = x - \hat{x}$ is given by

$$\dot{\tilde{x}}(t) = (A - KC)\tilde{x}(t) \tag{13}$$

The characteristic polynomial of this system is

$$s^3 + k_1s^2 + k_2s + k_3$$

The observer gains k_1, k_2 and k_3 are chosen so that the observer has the characteristic polynomial

$$(s + a_o)(s^2 + 2\zeta_o\omega_o s + \omega_o^2) \tag{14}$$

The following observer gains are then obtained

$$\begin{aligned} k_1 &= 2\zeta_o\omega_o + a_o \\ k_2 &= \omega_o^2 + 2\zeta_o\omega_o a_o \\ k_3 &= \omega_o^2 a_o \end{aligned} \tag{15}$$

Discrete-Time Controller

To derive a discrete time controller the system (6) is sampled. This gives

$$\begin{aligned} x(k+1) &= \Phi x(k) + \Gamma u(k) \\ y(k) &= Cx(k) \end{aligned} \tag{16}$$

where

$$\Phi = \begin{pmatrix} 1 & h & h^2/2 \\ 0 & 1 & h \\ 0 & 0 & 1 \end{pmatrix}, \quad \Gamma = \begin{pmatrix} K_p h^2/2 \\ K_p h \\ 0 \end{pmatrix} \tag{17}$$

$$C = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}$$

and h is the sampling interval. The states x_1 and x_2 of the discrete time system (17) are controllable but disturbance state x_3 is of course uncontrollable. All states are observable.

First consider the case when all states are measured. With state feedback the closed loop system has the characteristic polynomial

$$(z-1)\left(z^2+\left(K_p\frac{h^2}{2}l_1+K_phl_2-2\right)z+1-K_phl_2+K_p\frac{h^2}{2}l_1\right) \tag{18}$$

Notice that the pole $z=1$ is due to the uncontrollable disturbance mode. The desired closed loop characteristic polynomial is obtained by sampling (9). This gives

$$(z-1)(z^2+a_{p1}z+a_{p2})$$

where

$$\begin{aligned} a_{p1} &= -2e^{-\zeta_p\omega_p h}cos\left(\omega_p h\sqrt{1-\zeta_p^2}\right) \\ a_{p2} &= e^{-2\zeta_p\omega_p h} \end{aligned} \tag{19}$$

Choosing the feedback gains l_1 and l_2 so that (19) and (18) are the same gives

$$\begin{aligned} l_1 &= \frac{a_{p1}+a_{p2}+1}{K_ph^2} \\ l_2 &= \frac{a_{p1}-a_{p2}+3}{2K_ph} \end{aligned} \tag{20}$$

State Observer. A state-observer of the form

$$\begin{aligned} \hat{x}(k|k) &= \hat{x}(k|k-1)+K\big(y(k)-\hat{y}(k|k-1)\big) \\ \hat{x}(k+1|k) &= \Phi\hat{x}(k|k)+\Gamma u(k) \\ \hat{y}(k+1|k) &= C\hat{x}(k+1|k) \end{aligned} \tag{21}$$

is chosen. The reconstruction error is then given by

$$\tilde{x}(k+1|k)=\Phi(I-KC)\tilde{x}(k|k-1) \tag{22}$$

This system has the characteristic polynomial

$$z^3+\left(k_1+hk_2+\frac{h^2}{2}k_3-3\right)z^2+\left(3-2k_1-hk_2+\frac{h^2}{2}k_3\right)z+k_1-1$$

Requiring that this polynomial be equal to

$$(z-a_{o3})(z^2+a_{o1}z+a_{o2})$$

where a_{o1} and a_{o2} are given by equation (19) and

$$a_{o3}=e^{-a_oh} \tag{23}$$

gives

$$\begin{aligned} k_1 &= 1-a_{o2}a_{o3} \\ k_2 &= \frac{a_{o1}-a_{o2}-a_{o3}+a_{o1}a_{o3}+3a_{o2}a_{o3}+3}{2h} \\ k_3 &= \frac{a_{o1}+a_{o2}-a_{o3}-a_{o1}a_{o3}-a_{o2}a_{o3}+1}{h^2} \end{aligned} \tag{24}$$

The Control Algorithm

Reorganizing the calculations to minimize the delay between the A/D- and D/A-conversions gives the following algorithm.

ALGORITHM 1

1.	Read	$y(k)$
2.	Compute	$\hat{x}(k\|k)=\hat{x}(k\|k-1)+Ke(t)$
		$e(t)=y(k)-\hat{y}(k\|k-1)$
		$v(k)=-L\hat{x}(k\|k)$
		$u(k)=f(v(k))$
3.	Output	$u(k)$
4.	Update	$\hat{x}(k+1\|k)=\Phi\hat{x}(k\|k)+\Gamma u(k)$
		$\hat{y}(k+1\|k)=C\hat{x}(k+1\|k)$
5.	Wait	

where the function f is a model of the actuator nonlinearity. □

Notice that the algorithm has been organized so that the computational delay between the A/D and D/A converters are minimized. Notice also that Step 2 of this algorithm can be expressed as

$$\begin{aligned} \hat{x}(k+1|k) &= \Phi_n\hat{x}(k|k-1)+\Gamma_n y(k) \\ v(k) &= C_n\hat{x}(k|k-1)+D_n y(k) \end{aligned} \tag{25}$$

where

$$\begin{aligned} \Phi_n &= \Phi-\Phi KC-\Gamma L+\Gamma LKC \\ \Gamma_n &= \Phi K-\Gamma LK \\ C_n &= -L+LKC \\ D_n &= -LK \end{aligned}$$

Sampling Frequency and Anti-Aliasing Filter

The following rule of thumb for the selection of sampling frequency for a digital controller with a zero-order hold, is given by [Åström and Wittenmark, 1990].

$$0.2 \leq \omega_c h \leq 0.6 \qquad (26)$$

where ω_c is the crossover frequency. With a sampling frequency of 20 KHz the crossover frequency can be at least 1 kHz. This was judged adequate for the application.

A prefilter in the form of a second order Bessel filter with the bandwidth 7500Hz was chosen to avoid aliasing.

Design Parameters

The controller has the design parameters: ω_p, ζ_p, ω_o, ζ_o, a_o and h that must be chosen. The choice of sampling interval has already been discussed. Parameters ζ_p and ζ_o, which represent relative damping, can easily be chosen. Then there remain three parameters ω_p, ω_o and a_o. Requirements on desired settling time and disturbance rejection have to be matched against constraints due to model uncertainty. Recall that the rigid body model used for the design was not valid for frequencies approaching 1 kHz. After some experimentation the following design parameters were chosen for the nominal case.

$$\omega_p = 1000\pi$$
$$\zeta_p = 0.80$$
$$\omega_o = 1500\pi$$
$$\zeta_o = 0.80$$
$$a_o = 200\pi$$

Figure 1 shows a simulation of the response of the system with the nominal design parameters. In the simulation a step command at 11.5 μm is first applied. After 3 ms a torque disturbance in the form of a step of 0.013 Nm is applied.

The simulation was performed assuming that the plant model is given by Equation (3), which has a resonance of 2 kHz. The settling time is about 1.5 ms and the resonant modes are not much excited by the command signal. With the rigid body process model given by Equation (2) the system has an amplitude margin of 3.2 and phase margin of 31°. The gain cross-over frequency is 460 Hz and the phase cross-over is 1036 Hz. This indicates that the design based on the rigid body model has acceptable margins.

The effects of the neglected dynamics on the margins can be estimated as follows. Assuming

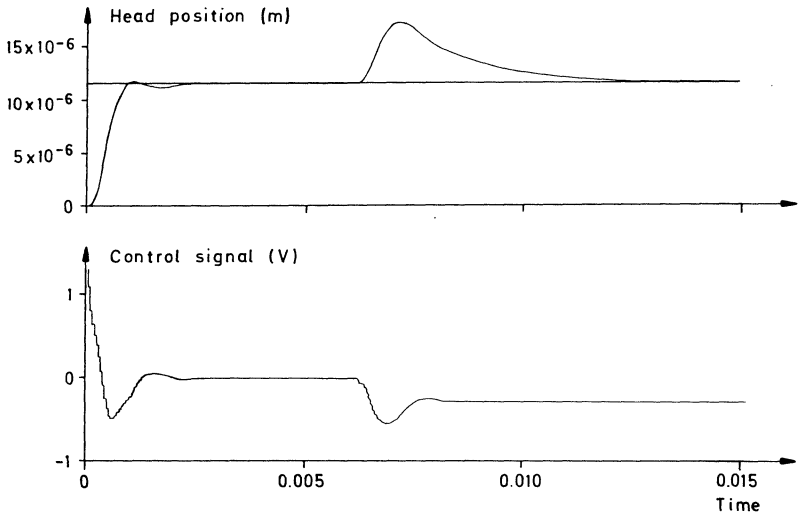


Figure 1. Step response of the closed loop system.

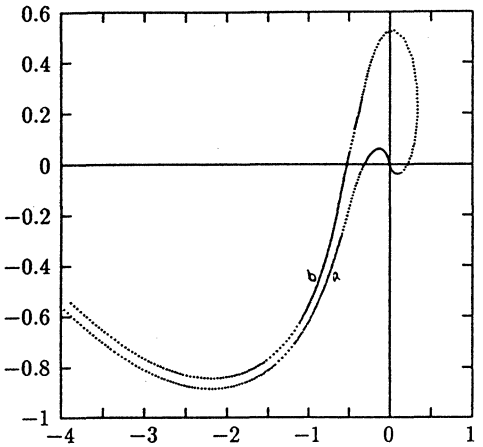


Figure 2. Nyquist curves for the loop transfer functions with the rigid body dynamics, Equation (2), and the dynamics with one resonant mode, Equations (2) and (3).

that the system dynamics is described by the model having one resonant mode, Equation (3). The additional dynamics is then given by

$$G_1(s) = \frac{\omega_1^2}{s^2 + 2\zeta\omega_1 s + \omega_1^2} \tag{27}$$

where ω_1 is the undamped natural frequency (2 KHz) and ζ is the relative damping (0.1). The magnitude M of the transfer function G_1 at ω is

$$M = \frac{1}{\sqrt{(1-\omega^2/\omega_1^2)^2 + (2\zeta\omega/\omega_1)^2}} \tag{28}$$

Introducing $\omega = \omega_{\phi c} = 1036$ Hz this equation gives $M = 1.36$. The gain margin is thus decreased to 1.77. The argument of the transfer function of ω is

$$\alpha = -\arctan\frac{2\zeta\omega/\omega_1}{1-\omega^2/\omega_1^2} \tag{29}$$

with $\omega_{gc} = 460$ Hz, which gives 2.8°. Figure 2 shows the Nyquist curves with the nominal process transfer function (2) and the transfer function with one resonant mode (3). These curves show that the essential effects of the resonant mode is to decrease the amplitude margin.

An additional illustration of the sensitivity to gain variations is illustrated in the simulation in Figure 3, which shows the time response of the closed loop systems, where the loop gain changes with ±20%. Compare with the nominal case in Figure 1.

Tracking Error

Misalignment errors is a common source of tracking errors. Such disturbances can be approximated by a sinusoidal. The sensitivity of the closed loop system to such errors can be modeled by the pulse transfer function.

$$H_{track}(z) = \frac{1}{1 - H(z)G(z)} \tag{30}$$

With the chosen controller we find that disturbances of 60 Hz are attenuated by a factor of 32. This agrees well with the simulation results that showed a reduction from 5 μm to 0.2 μm.

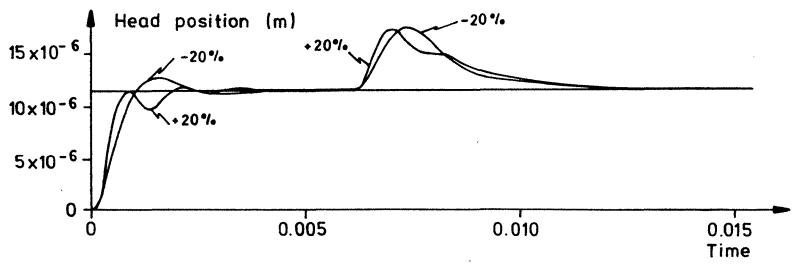


Figure 3. Responses of the closed loop system to a step command and a step change in the torque when the process gain changes by ±20%.

4. DSP Implementation

Implementation of the controller using a DSP with fixed point calculations will now be discussed. The key issues are scaling of coefficients and states. See [Roberts and Mullins, 1987], [Hanselmann, 1987], [Texas Instruments, 1986], [Texas Instruments, 1988a], [Texas Instruments, 1988b], [Texas Instruments, 1989a], [Texas Instruments, 1989b], [Texas Instruments, 1990a] and [Texas Instruments, 1990b].

The controller derived can be described by the matrices:

$$\begin{aligned}
\Phi &= \begin{pmatrix} 1 & 5\cdot 10^{-5} & 1.25\cdot 10^{-9} \\ 0 & 1 & 5\cdot 10^{-5} \\ 0 & 0 & 1 \end{pmatrix} \\
\Gamma &= \begin{pmatrix} 9.25\cdot 10^{-8} & 3.7\cdot 10^{-3} & 0 \end{pmatrix}^T \\
C &= \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \\
K &= \begin{pmatrix} 3.352917424019266\cdot 10^{-1} \\ 1.100808656418762\cdot 10^{3} \\ 5.695461161564441\cdot 10^{5} \end{pmatrix} \\
L &= \begin{pmatrix} 1.176909751519137\cdot 10^{5} \\ 6.300506379182784\cdot 10^{1} \\ 1.351351351351351\cdot 10^{-2} \end{pmatrix}^T
\end{aligned} \tag{31}$$

The elements of these matrices have numbers that are widely spread. To accommodate this on a DSP with fix point arithmetic it is necessary to scale the numbers appropriately.

I/O-Scaling

The range of the output signal in tracking mode corresponds to $\pm 11.5\mu$m. The scaling will be chosen so that this corresponds to ± 1 units in the DSP. The input scaling factor s_y is therefore

$$s_y = 11.5\ \mu\text{m}$$

Since the dimensions of l_1, l_2 and l_3 are [v/m], [sv/m] and [s^2v/m] respectively, it is advantageous to multiply L with s_y rather than dividing C with s_y. The output must also be scaled since the D/A-converter converts ± 1 into ± 5 V. The matrix L is thus multiplied by the output scaling factor

$$s_u = \frac{2}{10\ \text{V}} = 0.2\ \text{V}^{-1}$$

The vector Γ is in the same way as L. Hence

$$\begin{pmatrix} L & \Gamma \end{pmatrix} \Leftrightarrow \begin{pmatrix} s_y s_u L & \frac{1}{s_y s_u}\Gamma \end{pmatrix}$$

and the scaled vectors Γ and L become

$$\Gamma = \begin{pmatrix} 4.021739130434783\cdot 10^{-2} \\ 1.608695652173913\cdot 10^{3} \\ 0 \end{pmatrix} \tag{32}$$

$$L = \begin{pmatrix} 2.706892428494014\cdot 10^{-1} \\ 1.449116467212040\cdot 10^{-4} \\ 3.108108108108108\cdot 10^{-8} \end{pmatrix}^T \tag{33}$$

Coefficient Scaling

The coefficients of system (31), and (32) and (33) can not be represented in the DSP. A similarity transform $\hat{x} \Leftrightarrow T_c\hat{x}$ is used to scale the coefficients. This gives

$$\begin{aligned}
&\begin{pmatrix} \Phi & \Gamma & C & K & L \end{pmatrix} \Leftrightarrow \\
&\begin{pmatrix} T_c\Phi T_c^{-1} & T_c\Gamma & CT_c^{-1} & T_cK & LT_c^{-1} \end{pmatrix}
\end{aligned} \tag{34}$$

The elements of the matrices Φ, K, Γ and L are proportional to powers of h. It is therefore natural to use a scaling matrix of the form

$$T_c = \text{diag}\begin{pmatrix} cs_1 & cs_2 & cs_3 \end{pmatrix}$$

The following scaling matrix was obtained after some trial and error

$$T_c = \text{diag}\begin{pmatrix} \frac{1}{1.05k_1} & \frac{1}{5k_2} & \frac{1}{40k_3} \end{pmatrix} \tag{35}$$

State-vector scaling

With the chosen scaling of all controller coefficients have magnitudes less than one. It now remains to scale the state-vector. Simulations showed that overflow could occur when the head is positioned at the edge of the track and the disk controller is switched to track-following. The scale factors s_1, s_2 and s_3 were chosen from a simulation of this case. It was found that x_1 had to be scaled down and that x_2 could be scaled up. Scaling of x_3 depends on the maximum possible load disturbance. For a load disturbance of 0.3 Nm it was not necessary to scale

x_3. The following transformation was therefore chosen to scale the state vector

$$T_{sc} = \mathrm{diag}\begin{pmatrix} 1/2.8 & 1/0.3 & 1 \end{pmatrix} \qquad (36)$$

The following controller matrices were obtained after scaling:

$$\Phi = \begin{pmatrix} 1 & \phi_{12} & \phi_{13} \\ 0 & 1 & \phi_{23} \\ 0 & 0 & 1 \end{pmatrix}$$

$$\Gamma = \begin{pmatrix} 4.079845420409798 \cdot 10^{-2} \\ 9.742508490121855 \cdot 10^{-1} \\ 0 \end{pmatrix}$$

$$C = \begin{pmatrix} 9.857577226616642 \cdot 10^{-1} & 0 & 0 \end{pmatrix} \qquad (37)$$

$$K = \begin{pmatrix} 3.401360544217687 \cdot 10^{-1} \\ 6.666666666666667 \cdot 10^{-1} \\ 2.5 \cdot 10^{-2} \end{pmatrix}$$

$$L = \begin{pmatrix} 2.668340115802361 \cdot 10^{-1} \\ 2.392799926898983 \cdot 10^{-1} \\ 7.080843606269305 \cdot 10^{-1} \end{pmatrix}^T$$

where

$$\phi_{12} = 8.375348965918672 \cdot 10^{-2}$$
$$\phi_{13} = 2.888874735967583 \cdot 10^{-2}$$
$$\phi_{23} = 6.898517895130376 \cdot 10^{-1}$$

The system matrices are finally transformed to integers to fit the 16 bit fractional format of the DSP. The transformation is done by multiplying the coefficients with 2^{15} and rounding each coefficient to the nearest integer. The matrices then become

$$\Gamma = \begin{pmatrix} 1337 \\ 31924 \\ 0 \end{pmatrix}^T$$

$$\Phi = \begin{pmatrix} 1 & 2744 & 947 \\ 0 & 1 & 22605 \\ 0 & 0 & 1 \end{pmatrix}$$

$$C = \begin{pmatrix} 32301 & 0 & 0 \end{pmatrix} \qquad (38)$$

$$K = \begin{pmatrix} 11146 \\ 21845 \\ 819 \end{pmatrix}$$

$$L = \begin{pmatrix} 8744 & 7841 & 23203 \end{pmatrix}^T$$

The largest roundoff error 0.04% occurs in ϕ_{13}. To find how the poles of the controller are affected by the coefficient rounding, the characteristic equation of the controller was calculated. The largest pole deviation is 0.0013% from the design value.

5. The DSP Code

The control algorithm was implemented on the TMS320C25 by using the Texas Instruments Software Development System. The complete code is listed in Appendix A. The organization of the code is straightforward. It is composed of the following steps:

1. Perform A/D conversion.
2. Compute the state estimate.
3. Compute the new control signal.
4. Saturate control signal.
5. Perform D/A conversion.
6. Update equations for state estimate.

Compare with Algorithm 1. Approximately 32% of the computational power of the TMS320C25 used when the controller was running.

It can be estimated how processor loading increases with the order of the controller. Neglecting saturation arithmetic and anti-windup calculations, the number of multiply/accumulate instructions are proportional to $n^2 + 5n$ where n is the order of the controller. A 6th order controller would therefore exhaust the computational power of the C25. The saturation arithmetic routine must be

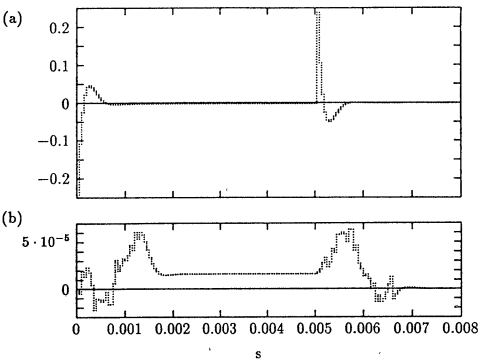


Figure 4. Impulse response of the controller (a) and the error compared to an ideal implementation (b).

called approximately 2n times. In our case, the saturation arithmetic consumes almost 50% of the total execution time. Therefore, if saturation arithmetic can be avoided by using more careful scaling, one can estimate that a 10th order controller can be implemented on the C25.

6. Testing

The open loop behavior of the controller was tested using the development system. The impulse response of the controller was generated and compared to the ideal impulse response. Figure 4(a) shows the responses of the controller to two impulses of magnitude 0.9 and –0.9. Figure 4(b) shows the error between the ideal and actual impulse response of the controller. The small error small is due to the roundoff in the controller. Notice that the quantization step is approximately $3 \cdot 10^{-5}$.

The observer was tested separately. A control signal was generated and the corresponding ideal response of the arm was calculated. The input signal was piecewise constant with jumps at $t = 0$, 0.001, 0.0018, and 0.0021. A load disturbance that was unknown to the observer was added at time $t = 0.0025$. All signals were scaled appropriately and fed to the observer whose response was recorded. Figure 5 shows the velocity estimate and its error. Figure 6 shows the position estimate and its error. The error is very small before time

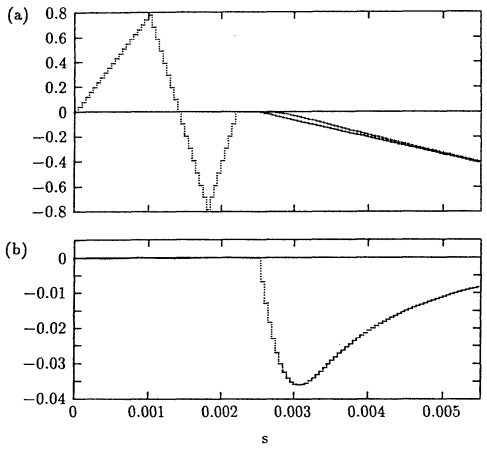


Figure 5. Actual and estimated velocity (a) and estimation error (b).

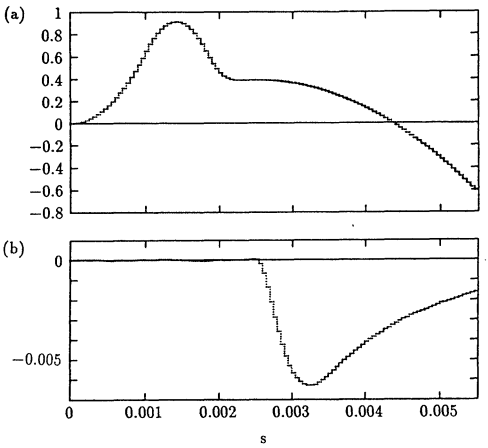


Figure 6. Actual and estimated position (a) and estimation error (b).

$t = 0.0025$, where the load disturbance was introduced. The load disturbance does, however, introduce significant errors both in velocity and position estimates. This is natural, because the observer does not have information about this load disturbance. The error will, however, decrease when the observer improves its estimate of the disturbance as is indicated in Figure 6.

Although open loop testing can never replace actual closed loop testing of the whole system, these results indicate that the controller works properly.

Remarks on a Roundoff Algorithm

The first tests of the algorithm used a roundoff scheme found in a programming example in [Texas Instruments, 1986]. This resulted in a large estimation error, see Figure 7. The problem was investigated, since the error was larger than estimates based on analysis of roundoff errors. The reason for this is an error in the roundoff algorithm. To reduce quantization errors the numbers are rounded off, rather than truncated, before they are stored as 16-bit numbers. This roundoff is done in software. To roundoff a positive number, a bit is added to the MSB of the lower half of the 32-bit number before it is stored away. At first sight it appears natural to subtract the bit from the number to roundoff a negative number. This was done in the coding example [Texas Instruments, 1986]. This is not correct

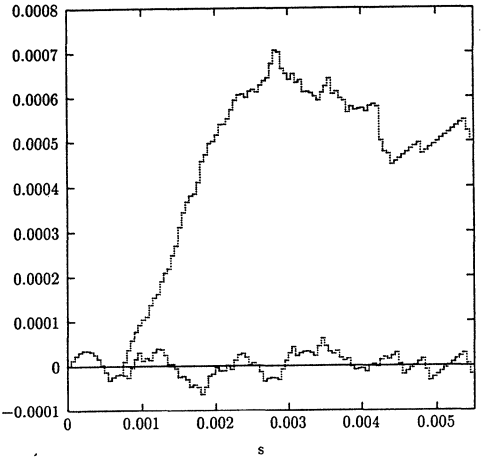


Figure 7. Position error with an incorrect round-off algorithm.

with the chosen number representation. The round-off algorithm gives –2 when applied to the number –1 because of the computational scheme used in the DSP. If the upper half of the number is complemented without considering the lower half, the result will not be the same as if the whole number is complemented. The correct code for the roundoff is given in Appendix A.

7. Conclusions

This paper shows that it is straightforward to implement a controller based on an observer and feedback from the observed states using a DSP with fix point calculations. Some effort is required to obtain proper scaling. The coefficient scaling is quite straightforward and can be automated. Scaling of the states is more difficult. It requires that the ranges of the states are known. This can be determined from simulation. Great care has to be exercised to find the worst cases. The code for the disk controller is much simpler than the code for the PID-controller discussed in [Åström and Steingrímsson, 1991]. The reason is that the disk controller is designed for a specific process while the PID-controller is designed as a general purpose controller. The coefficient ranges for the PID-controller are therefore much wider. This requires more complex scaling and saturation arithmetic, which is a large part of the code [Åström and Steingrímsson, 1990].

8. References

Åström, K. J. and H. Steingrímsson (1990): "Implementation of a PID controller on a DSP," Texas Instruments.

Åström, K. J., and B. Wittenmark (1990): *Computer Controlled Systems – Theory and Design*, Second edition, Prentice-Hall, Englewood Cliffs, NJ.

Dote, Y. (1990): *Servo Motor Control Using Digital Signal Processor*, Prentice Hall, Texas Instruments.

Hanselmann, H. (1987): "Implementation of digital controllers – A survey," *Automatica*, **23**.

Roberts, R. A., and C. T. Mullins (1987): *Digital Signal Processing*, Addison-Wesley Publ Co.

Texas Instruments (1986): *Digital Signal Processing Applications with the TMS320 Family – Theory, Algorithms, and Implementations*, Digital Signal Processing, Semiconductor Group.

Texas Instruments (1988a): *First-Generation TMS320 – User's Guide*, TI Digital Signal Processing, Prentice Hall.

Texas Instruments (1988b): *Second-Generation TMS320 – User's Guide*, TI Digital Signal Processing, Prentice Hall.

Texas Instruments (1989a): *TMS320C1x / TMS320C2x – User's Guide*, Digital Signal Processor Products.

Texas Instruments (1989b): *TMS320 Family Development Support – Reference Guide*, Digital Signal Processor Products.

Texas Instruments (1990a): *Digital Signal Processing – Applications with the TMS320 Family*, Application book volume 3, Digital Signal Processor Products.

Texas Instruments (1990b): *TMS320C3x – User's Guide*, Digital Signal Processor Products.

Appendix A: Disk Controller for TMS320C25

```
; Disk Controller for the TMS320C25
; Based on a Rigid Body Model of the Arm
; Version 1.0
; Author: Hermann Steingrimsson
; Date: 3-31-1990
;
;
; RESERVE SPACE IN DATA MEMORY FOR CONSTANTS AND VARIABLES
DTbeg .bss A12,1 ;The matrix A (or Phi)
.bss A13,1
.bss A23,1
.bss B1,1 ;The vector B (or Gamma)
.bss B2,1
.bss C1,1 ;C1
.bss K1,1 ;The vector K (in this case CK/2)
.bss  K2,1
.bss K3,1
.bss L1,1 ;The vector L
.bss L2,1
.bss L3,1
.bss    MAXNUM,1   ;Maximum number
.bss MINNUM,1   ;Minimum number
.bss UMAX,1 ;Saturation limits
.bss UMIN,1
.bss ONE,1 ;ONE=1
.bss MODE,1
DTend .bss CLOCK,1 ;End of parameters in data memory
.bss XE1,1 ;State vector x(k+1|k)
.bss XE2,1
.bss XK1,1    ;Vector x(k|k)
.bss    XK2,1
        .bss XK3,1
.bss YE,1 ;Estimate of ye
.bss Y,1 ;Input
.bss ERR,1
.bss V,1 ;Control signal before saturation
.bss U,1 ;Control signal after saturation U=SAT(V)

;Begin program memory

.sect "IRUPTS"
B START ;Branch to start of program

;Store parameters in program memory

.data
Ptable .set $
.word 2744,947,22608,1337,31924,32301,11146,21845,819,8744,7841,23203
```

```
.word   32767,-32768,32766,-32766,1,1,1
.word -1
Ptend .set $-1
SCALE .set 15

;Initialize

.text
START DINT ;Disable interupts
NOP
SOVM ;Set overflow mode
SSXM ;Set sign-extension mode
SPM   0 ;No shifting from P register

;Load coeff from prog. mem to data mem. use BLKP

LRLK  AR0,DTbeg  ;AR0 points to begining of data block
LARP  AR0
RPTK  Ptend-Ptable ;Set up counter
BLKP  Ptable,*+ ;Move data
;=> Coeff loaded into data memory

;Initialize variables

LDPK  A12 ;Point to correct data page
ZAC ;Clear variables
SACL  XE1
SACL  XE2
SACL  XK3
SACL  YE
SACL  U

OUT  MODE,PA4 ;Init analog board
OUT  CLOCK,PA5
LARP 0 ;Point to AR0

;Begin loop

;WAIT BIOZ  GET ;Wait for input
; B     WAIT
WAIT IN    Y,PA0 ;Change WAIT to GET when ; are removed


ZALH  Y ;Form ERR = y(k) - ye(k|k-1)
SUBH  YE
SACH  ERR

;Compute x(k|k) = x(k|k-1) + K*err
```

```
LAC   XE1,SCALE ;Calculate x1(k|k)
LT    ERR
MPY   K1
APAC
LRLK  AR0,XK1
CALL  ROUOF

LAC   XE2,SCALE ;Calculate x2(k|k)
LT    ERR
MPY   K2
APAC
LRLK  AR0,XK2
CALL  ROUOF

LAC   XK3,SCALE ;Calculate x3(k|k) (Estimate xe3 not needed)
LT    ERR
MPY   K3
        APAC
LRLK  AR0,XK3
CALL  ROUOF

;Calculate control signal u(k) = -Lx(k|k)

ZAC
LT    XK1
MPY   L1

LTS   XK2
MPY   L2

LTS   XK3
MPY   L3
SPAC

LRLK  AR0,V
CALL  ROUOF

;Saturation function (12 instr. cycles)

ZALH  V
SUBH  UMIN
BLZ   LOWER1 ;Branch if v < umin
ZALH  V
SUBH  UMAX
BLZ   SAME ;Branch if v < umax
ZALH  UMAX ;v >= umax
SACH  U ;u = umax
B     FIN ;Begining of loop
```

```
LOWER1 ZALH  UMIN
SACH  U ;u = umin
NOP ;Always same time
NOP
NOP
NOP
B     FIN

SAME ZALH  V
SACH  U ;u = v
NOP
NOP

FIN OUT   U,PA2 ;Output control signal

;Update the estimate xe(k+1|k) = Ax(k|k) + Bu(k)
;                    ye(k+1|k) = Cxe(k+1|k)

LAC   XK1,SCALE ;Calculate xe1

LT    XK2
MPY   A12

LTA   XK3
MPY   A13

LTA   U
MPY   B1
APAC

LRLK  AR0,XE1
CALL  ROUOF

LAC   XK2,SCALE ;Calculate xe2

LT    XK3
MPY   A23

LTA   U
MPY   B2
APAC

LRLK  AR0,XE2
CALL  ROUOF

;No need to update xe3 (xe3 = xk3)

LT    XE1 ;Calculate ye
MPY   C1
```

```
PAC

LRLK  AR0,YE
CALL  ROUOF

B     WAIT ;Loop

; Rounding, overflow and shifting function (11 cycles)

ROUOF BLZ   NEG ;Check if number negative
ADD   ONE,SCALE-1 ;Round
SACH  *,16-SCALE ;Store value
SUB   MAXNUM,SCALE ;Subtract scaled max pos number
BLEZ  NOOV ;If acc <= 0 then no overflow
ZALS  MAXNUM ;else store max num
SACL  *
RET

NEG ADD   ONE,SCALE-1 ;Round
SACH  *,16-SCALE ;Store value
SUB   MINNUM,SCALE ;Subtract scaled min neg number
BGEZ  NOOV ;If acc >= 0 then no overflow
ZALS  MINNUM ;else store min neg number
SACL  *
RET

NOOV NOP
NOP
RET

.endæ
```

PART IV
Applications of Digital Controllers with the TMS320

Automotive

Specialized Applications

Digital Control Applications with the TMS320

More designers are using DSPs to solve problems that commonly occur in control applications. DSPs now make practical some applications that were previously difficult to implement or were not cost-effective. As the cost of DSPs decreases, these processors are rapidly replacing microcontrollers and analog components in many control applications.

Some applications in which DSPs are already cost-effective are servo control for computer peripherals, power control in uninterruptible power supply (UPS) and DC power supply systems, motion control for numerical control (CNC) systems and robotics, suspension/engine/brake control for automotive systems, and vector control for AC and other brushless motors. Other applications are missile guidance and "smart" weapon control for military systems.

This introduction presents a few areas of DSP-controlled applications. Following it, papers discuss topics pertaining to those and other areas. Most of these documented applications have evolved into very successful commercial products.

Computer Peripherals

Many computer peripherals use DSPs for applications such as read/write head control in winchester disk drives, tape control in tape drives, pen control in plotters, and optical beam positioning and focusing in optical disks.

Disk Drives: Disk drives were early to adopt DSPs. DSPs are used for servo control of the actuator driving the read/write head. Disk drives employ a voice-activated coil motor with high bandwidth. Data is read from the disk at a very high rate; sampling rates of up to 50 kHz are sometimes used. In addition to implementing the compensator, DSPs can implement notch filters to attenuate undesirable frequencies that cause mechanical resonances or vibrations.

Tape Drives: In tape drives, DSPs are used to control the tape mechanism. A tape drive has two servo loops: one controls the tape speed, and the other controls the tension placed on the tape. Position feedback is obtained from an optical encoder, and tension information is fed from a tension sensor. DSPs are also used to filter undesirable frequencies that cause mechanical resonances.

Power Electronics

DSPs can be used in multiple applications in power electronics. These applications include AC servo drives, inverter control, robotics, and motion control.

AC Servo Drives: In AC servo drives, DSPs are used for vector control of AC motors. AC drives are less expensive and easier to maintain than DC drives. However, AC drives have complex control structures as a result of the cross-coupling of three-phase currents. Vector rotation techniques are used to transform three-phase axes into rotating two-phase "d – q" axes. This two-phase rotation technique greatly simplifies the analysis, making it equivalent to analyzing field-wound DC motors.

UPSs and Power Converters: In uninterruptible power supplies (UPSs) and power converters, DSPs are used for PWM generation along with power factor correction and harmonic elimination. Advanced mathe-

matical techniques can be used to control the firing angles of the inverter, creating low-harmonic PWM with unity power factors.

Robotics and Motion Control: DSPs are used in large-scale applications in robotics and other axis control applications. DSPs support high-precision control along with implementation of advanced techniques like state estimators and adaptive control. A single controller can handle speed/position control as well as current control. Time-varying loads can be handled with adaptive control techniques. Adaptive control techniques can also be used to create universal controllers that can be used with different motors. In addition to implementing controllers, DSPs implement notch filters to attenuate undesirable frequencies that causes resonances or vibrations.

Automotive

DSPs can be used for many automotive applications such as active suspension, anti-skid braking, engine and transmission control, and noise cancellation.

Active Suspension: Active suspension systems use hydraulic actuators. DSPs can take into consideration body dynamics, such as pitch, heave, and roll, and then use this information to control four actuators independently and dynamically for counteracting external forces and the car's attitude changes.

Anti-Skid Braking: In anti-skid braking systems, DSPs can read the wheel speed from sensors, calculate the skid distance, and control the pressure in the wheel's brake cylinder. Traction-regulating systems can be added to control the vehicle in adverse driving conditions, to prevent wheel(s) from locking or spinning, and to increase general vehicular stability, steerability, and drivability.

Engine Control: In engine control applications, DSPs can be used with in-cylinder pressure sensors to perform engine pressure waveform analysis. This information can be applied to determine the best spark timing, most effective firing angles, and optimal air/fuel ratios. The closed-loop engine control scheme can tolerate external turbulences, aging, and wearing, while maintaining optimum engine performance and fuel efficiency.

DSP helps keep disk drives on track

Using a sophisticated DSP chip to implement adaptive embedded servo control avoids the head-positioning errors that can plague high-density Winchester disk drives.

Conventional design approaches are inadequate to meet the demand for ever-higher track densities on Winchester disk drives. When densities exceed 1,200 tracks/in., drives relying on dedicated servo feedback for positioning accuracy become unpredictable parts of computer systems. Implementing embedded servo control with adaptive positioning features, however, allows the design and manufacture of adequately margined disk drives that provide a solid platform for higher densities.

Since designers can't predict exact performance, a disk drive with adequate margin requires reserve capability in all areas. Materials or components, for example, may not be within specifications, and environmental conditions may also exceed specifications or combine in unpredictable ways. For instance, electrical noise may combine with temperature changes in a peculiar way that even an exhaustive testing schedule could miss. In addition, materials and components change with time.

The search for ample safety margins led Vermont Research to use the 32020 digital signal processing (DSP) chip, from Texas Instruments (Dallas, TX), to incorporate adaptive embedded servo control into its Model 7030 hard disk drive. Digital signal processing of feedback signals offers immense flexibility for designers of many products, from disk drives to numerically controlled machine tools to aircraft control systems. Exploited to its fullest, the power of DSP can be used to expand reliability margins in numerous motion control applications.

James M. Corliss and Richard Neubert

Corliss is principal engineer and Neubert is a design engineer at Vermont Research (North Springfield, VT).

The dedicated servo approach

The most common method of locating a track on a Winchester disk drive has been the dedicated servo approach. The designer reserves one surface in the stack of platters where servo control information is written. If the head on that surface is correctly located, it's assumed that all other heads on the carriage are also on their tracks.

Sometimes dedicated servo drives work well, but higher track densities can make them hypersensitive to temperature changes, especially when combined with shock or vibration. The drives develop high error rates and may not retrieve data at all if conditions have changed since the recording. The problem is that positioning errors that may be manageable at lower track densities can cause significant positioning difficulty at higher track densities because the errors represent a larger percentage of the narrower tracks. If the heads aren't properly positioned, the analog signal-to-noise ratio plummets on readback, causing skyrocketing error rates and, sometimes, an unusable drive.

Embedded servo control provides feedback in the form of bursts of prerecorded positioning information embedded in data on the track that's being read. Adaptive positioning actively compensates for both external disturbances such as shock and vibration and internal changes such as the aging of shock mounts and creep of materials. Of course, the effectiveness of embedded servo control is

Shock Sensitivity			
Sample Rate (kHz)	Time Between Samples (μs)	Off-Track Shift (μin.)	
		2 G Shock	5 G Shock
10.0	100	4	9.7
5.0	200	16	38.6
3.3	300	36	86.9
2.5	400	64	154.0
2.0	500	100	241.0
1.6	600	144	347.0
1.4	700	196	473.0
1.2	800	256	618.0
1.1	900	324	782.0
1.0	1,000	400	965.0

The shock sensitivity of a drive with embedded servo is a function of the amount of time between sampling. At a 10-kHz sampling rate, a 2-G shock induces only a 4-μin. off-track error; at 1.2 kHz, it's 256 μin.—enough to accidentally destroy data on an adjacent track.

limited by the frequency at which positioning feedback is provided. If the sampling frequency is too low, the track-following errors that accumulate between samples will be larger than the errors a dedicated servo approach would have allowed. Inadequate feedback also makes positioning performance suffer.

Adaptive embedded servo positioning, as implemented in the Model 7030 disk drive, wasn't practical before the advent of sophisticated DSP chips, which can analyze rapid-fire bursts of servo information and make quick position corrections. Implementing adaptive positioning without sacrificing access time or user flexibility required a new level of servo information analysis that relies heavily on digital signal processing. Pre-DSP electronics wouldn't have been practical for the adaptive embedded servo approach at a satisfactory sampling rate. The cost and real estate requirements of discrete logic would have been prohibitive.

That's not to say that using an advanced DSP chip like the TI 32020 for multiple signal processing functions is completely straightforward. Since the functions can't be truly simultaneous, priorities must be carefully established. Also, there are some disadvantages to using adaptive embedded servo control. One is a recording overhead of 15 percent of a drive's capacity, compared to 10 percent for dedicated servo and 7 or 8 percent for embedded servo with lower sampling rates. Fortunately, though, this overhead cost is more than offset by the ability to reliably use higher track densities.

Living within a budget

Like every physical device, a disk drive has tolerances. Absolute perfection in head positioning isn't required for reliable drive performance, but there's a set limit on how much deviation is acceptable for each case. Disk drive designers commonly use a "tracking error budget" when analyzing all possible sources of track-following deviations. If the drive can't achieve an acceptable bit error rate unless the heads are, say, within 60 μin. of perfect positioning, then 60 μin. is the tracking error budget.

Suppose, for example, that differential thermal expansion may cause as much as 35 μin. of track-following error, despite the servo system's best compensation efforts. If shock and vibration contribute no more than 10 μin. and all other sources of error combined will be no more than 10 μin., the total possible error is 55 μin., and the 60-μin. error budget won't be exceeded.

As track widths diminish, however, error budget shrinks disproportionately. At 1,200 tracks/in., for example, a 60-μin. error budget is 10 percent of the track width. At a 1,500-track/in. density, though, with the accompanying decrease in absolute signal strength from the head, the error budget may have to shrink to 8 percent of the track width. In this case, the error budget becomes a mere 38 μin.

A matter of degrees

Temperature changes caused by operating or environmental conditions are a common source of trouble for reliable positioning. Differential ther-

mal expansion among the various materials in head support arms, disks, carriages, spindles, bearings and housings in the 5-in.-long chain of parts between the head and the disk is typically 5 μin./in./°C. At 1,200 tracks/in. on an 8-in. drive, that can mean that a track written when a drive is cold can shift half a track or more when the drive is warm. A mere change of 2.5 °C can consume an entire 60-μin. error budget.

Careful attention to air circulation in the drive can minimize temperature differences within it but can't eliminate them. While the temperature is changing, parts within the drive will expand or contract differently, even if they subsequently stabilize at a new temperature. A drive depending on dedicated servo information is blind to head shift during temperature changes. Thus, even though the servo head is positioned properly, the data head may not be. The drive will compensate for the dimensional changes affecting the reference head, but it can't compensate for the fact that the parts that locate the data heads are changing in a different way. In practice, temperature sensitivity means that a disk drive may need a warm-up period before it works reliably. It also may mean that information recorded last week is unavailable this week because of changes in the room temperature.

With adaptive embedded servo control, however, temperature sensitivity ceases to be an issue. The drive needs no warmup, data written cold can be read hot and vice versa, and changes in room temperature won't affect performance. System builders can ship software on the disk and be sure the disk will boot up. They'll see dramatic reductions in the number of dead-on-arrival drives and systems and won't have to carry a large inventory to compensate for failures.

Other sources of head-track misalignment that can cause positioning difficulty include creep and stress relief in materials, bearing runout, shock and vibration, head stack tilt, disk slip, and bending or twisting of the main frame chassis. All of these phenomena may affect one disk in the stack differently than the others, creating track-following errors for data heads even if the servo head is on track.

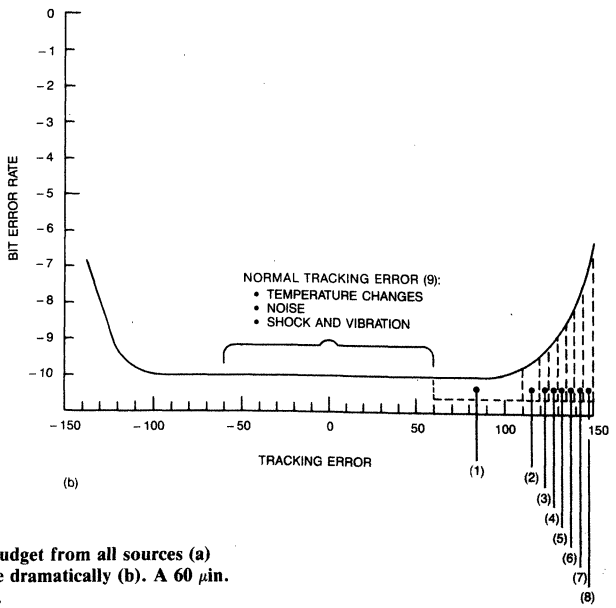
Coping with internal variables

One significant internal variable is head width, which typically varies ±10 percent and, thus, affects positioning accuracy. In the Model 7030 drive, head widths are measured by the DSP hardware during the factory configuration and test process and stored in nonvolatile RAM for use during operation. Other component characteristics, such as the actuator motor's magnet profile, are also programmed in firmware when the disk is built. One of

Tracking Error Components

Cause	Range (μin.)
Movement in Heads and Flexures (1)	50
Spindle Runout and Imbalance (2)	10
Flex Circuit Offset (3)	5
Disk Imbalance and Slip (4)	5
Stress Relief and Creep in Materials (5)	5
Bearing Runout and Centerline Drop (6)	5
Head Stack Tilt (7)	5
Bending and Twisting of Main Chassis (8)	5
Normal Tracking Error (9)	5

(a)



(b)

When a typical disk drive's tracking error budget from all sources (a) exceeds 100 μin., bit error rates begin to rise dramatically (b). A 60 μin. budget provides a greater margin for safety.

Embedded servo scheme guarantees accurate positioning

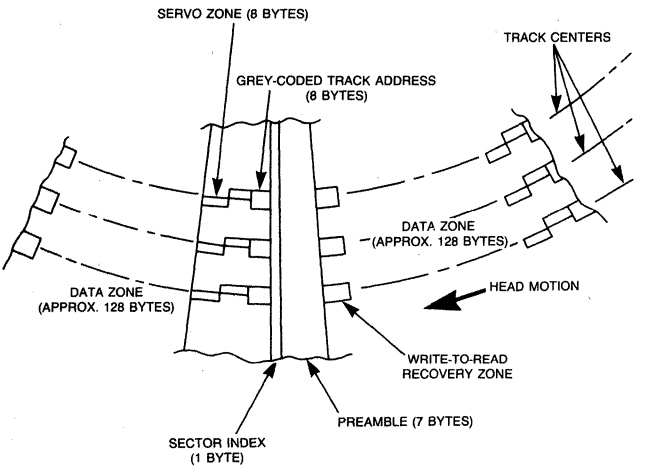
In Vermont Research's recently introduced 648-Mbyte, 8-in. Winchester disk drive (Model 7030), factory-recorded servo information bursts are embedded in data every 128 bytes, providing a position sampling rate of 9.6 kHz. This rate is, in fact, the practical equivalent of continuous feedback for track following and ensures quick, accurate positioning. Even a 1-G shock can move the heads only 3 μin. between samples, which isn't enough to perceptibly affect the data signal integrity.

The servo zone is divided into multiple parts. Before it enters a data zone, a head encounters a short preamble and a sector index mark, then a Gray-coded track address mark, and finally a pair of servo marks, offset in time, with one lying inside the track (toward the disk spindle) and one outside. The servo bursts are displaced in time so they can be distinguished. If the head is centered on the data track, the servo signals have the same amplitude; if not, one is larger than the other. Small variations in the timing of the servo bursts have no effect on the measurement as long as they occur within an allotted time window.

Each time the drive is started up, a self-test program is loaded into a Texas Instruments 32020 digital signal processor. The program verifies its own operation, the external data bus and the interrupt structure, and calibrates the analog-to-digital and digital-to-analog conversion circuitry and the signal path for positioning signals. It reads signal A into both channel A and B, and does the same with signal B, to measure the dc offsets and the gain ratio between the two channels. This guards against the possibility that the gains of the two position-indicator signal converters may have drifted apart.

To determine signal amplitude, the peak-to-peak value of each set of dipulses is averaged, which removes some high-frequency noise. The amplitude value of each position signal is digitized, and the DSP applies the measured values of dc offset and gain ratio to make any necessary corrections. The DSP then computes the difference of the two signals and divides the result by their sum to obtain a raw amplitude-compensated ratio that indicates the degree by which the head is off-track. This ratio is numerically filtered in a finite impulse response filter to remove further high-frequency noise.

The processed signal is now a position indicator. The measured rate of positional change also provides velocity and acceleration information, which the DSP uses to compute the amount of current to supply to the linear motor powering the head actuator.



the major strengths of adaptive positioning, however, is its ability to dynamically compensate for changing variables.

The force of the flex circuits connecting the heads to the drive electronics tends to move heads off-track. Mechanically, the flex circuit is a spring, so its force varies with the track address; during operation, the DSP computes the offsetting actuator current for each track address. As the flex circuit ages, its spring constant changes, but the drive adapts to the change with each startup. The DSP chip measures the force constant of the linear head actuator motor on startup by applying a pulse of current and measuring the resulting motion. If necessary, the drive can be rezeroed during operation to adjust for a temperature-changed force constant—or any other parameter.

A more complex adaptive feature is compensation for movement of the head-disk assembly on its shock mounts. The high forces needed to accelerate the head carriage displace the entire head-disk assembly on its shock mounts, typically by 0.020 in.—the equivalent of 20 tracks or more. The displacement becomes a damped oscillation in the 20- to 40-Hz range after the seek is completed.

Without compensation for the shock-mount oscillation, the drive's servo loop could follow about 99.7 percent of the displacement induced by the mounts, but that still leaves a 0.3 percent uncompensated displacement on the first cycle of the

damped oscillation, which represents 10 percent of a track width, or 60 μin.—the maximum that can be tolerated from all sources of error combined. Even after the first cycle, uncompensated oscillation would consume at least half of the drive's error allowance. That error represents margin that could hinder quick actuator settling time and, thus, adversely affect overall seek performance.

In the Model 7030, a mathematical model of the response of the shock-mounted assembly, which includes factors for frequency, amplitude and damping, is stored in the firmware of the 32020 DSP, which uses it to predict the damped oscillation and apply the inverse actuator current to cancel it. The DSP continually updates the parameters of the model, automatically adjusting for changes in the elastomer of the shock mounts and in other materials as they age or change temperature. Once per minute, the updated model is pulled into nonvolatile RAM, along with the updated flex circuit spring constant, as an accurate starting point in subsequent startups. A similar method is used to compensate for resonance of the actuator assembly after a seek, a problem that introduces some degree of track-following error in all disk drives.

In addition to improving performance, the inherent flexibility of adaptive embedded servo control lets a disk drive design take advantage of ongoing improvements in heads and disk coatings without requiring radical redesign. An additional benefit of insensitivity to component variations is that no adjustments are needed when boards are changed in the field, which saves time and requires less expertise from field service technicians.

The advanced DSP technology used for adaptive positioning also provides for sophisticated self-diagnosis and monitoring of environmental and power supply conditions. This capability can save hours of field service time by making it possible to pinpoint temperatures or voltages outside of specs as the source of difficulties that otherwise would cause a wild goose chase. **CD**

LQG-Control of a Highly Resonant Disk Drive Head Positioning Actuator

HERBERT HANSELMANN, MEMBER, IEEE, AND ANDREAS ENGELKE

Abstract—**A fast fine-positioning controller has been designed for a rotary actuator type magnetic storage disk drive. The controller was designed using the lqg (linear quadratic gaussian) methodology and has been implemented on a digital signal processor. It is shown that lqg design is a viable approach, and that various problems associated with the structural resonances of the actuator can be solved.**

Keywords—**magnetic disk storage, position control, microprocessor control.**

I. Introduction

MODERN DISK DRIVES use fast voice coil actuators for positioning magnetic heads on desired tracks and keeping them on track against various disturbances using closed-loop control. Two types of actuators are predominant in state-of-the-art drives: rotary and linear actuators, both driven by a current passing through a coil in a strong magnetic field.

In high-performance drives the head position is measured from a dedicated servo platter. Measurement electronics supply a head/track misalignment error voltage which is proportional to this error within track width. Current flowing through the coil generates torque or force so there must be closed-loop control.

We investigated fine-positioning control in an industrial prototype 8-in drive using a rotary actuator, as shown in Fig. 1. As described in some detail in [1], the studies were carried out on an experimental version of the drive with fixed disk-spindle. With an operating drive the investigations would have been hindered because of the required clean-room conditions. The position error measurement was achieved through an optical sensing device capable of measuring in the real position range of an operating drive (useful track width $+ -9\ \mu$m) with excellent resolution.

In [1] results of modal structural analysis as well as of control using a classical approach have been presented. The controller was of double PD-type with 3 notch-filters. It had finally been extended by a synthetic disturbance feedforward system.

It is the purpose of this paper to report on our effort to design and implement an appropriate controller using the lqg/ltr methodology. The plant is of the SISO (single-input single-output) type because only position error measurement is available. Thus designing a controller might seem to be fairly simple at first glance, but the strong structural resonance effects posed some problems, and the experience with the approaches taken might be of interest for others working on the control of mechanical systems.

Manuscript received March 27, 1987.

H. Hanselmann was with the Department of Automatic Control in Mechanical Engineering, University of Paderborn, Paderborn, W. Germany. He is now with dSPACE Digital Signal Processing and Control Engineering Gmbtt, Paderborn, W. Germany.

A. Engelke is with the Department of Automatic Control in Mechanical Engineering, University of Paderborn, Paderborn, W. Germany.

IEEE Log Number 8718148.

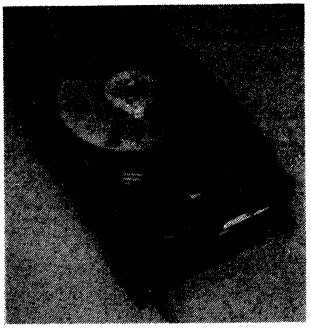


Fig. 1. Prototype disk drive.

II. Plant Model

A simple mathematical model would be a double integrator (torque to position). But the control bandwidth desired is so high that structural mechanics effects can by no means be neglected. Figure 2 shows the measured frequency response from input current to position in the 1- to 10-kHz frequency range. There are many resonances and notches, and zoomed analysis would show even more.

The second curve in Fig. 2 shows the frequency response as computed from a 30th order input-output (black-box) model, which has been formulated in state-space form. This model has basically been formed from resonance frequency, damping, and residue data obtained with the curve fitting facility of the structural dynamics analyzer we used. Additional tedious manipulations were however necessary in order to improve the model in both phase and amplitude response. The model is fairly good below 2 kHz and above 4 kHz, but less so between these frequencies. In particular the two deep notches between 3 and 4 kHz are not well represented in the model, and correspondingly there is considerable phase mismatch. The classical (notch-filter based) controller from [1], which was designed with a view to amplitude stabilization, was not sensitive to this model mismatch, in contrast to the lqg controller, as shown below. Certainly we should do some work on improving the model by better computerized model matching methods.

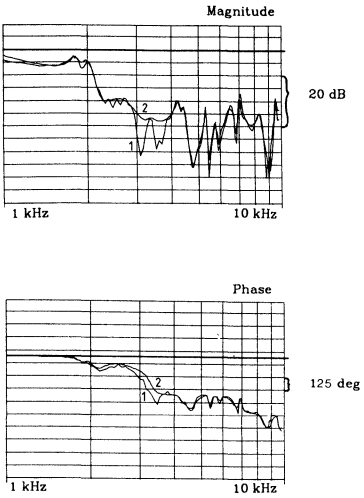


Fig. 2. Frequency response truth model (2)/measurement (1).

III. Control Design

Fine-positioning control may seem to be very easy because the plant is SISO and is virtually linear. Linearity is only lost when the current saturates. It turns out that the current limit is not reached with fine-positioning regulating control because the current range is designed for fast large-distance positioning at high torque. The plant is SISO because for economic reasons the only measurement presently available for control is the track position error itself.

From a design viewpoint, the difficulties of control arise mainly from the structural resonances ranging up to 10 kHz. A classical approach to coping with resonances is the well-known introduction of properly designed notch-filters. They compensate for resonance peaks to such an extent that these peaks drop significantly below 0 dB. In this case insensitivity to phase behavior of the plant is gained (amplitude stabilization). The controller from [1], which had been designed that way, yielded high bandwidth and performed well in the experiment. The design procedure was however not satisfactorily systematic. Fine-tuning of the filter parameters took a lot of time because we always had to look carefully at the total phase introduced by the filters around the projected crossover frequency. Tuning the controller was easier when we used a numerical parameter optimization program as reported in [1], but amplitude stabilization was lost, and design was still time-consuming.

This experience provided the motivation of trying the lqg approach, from which we hoped to get useful controllers in a systematic way with little effort. Some obstacles had however been anticipated, namely

1) a reduced-order design model was necessary, in contrast to the classical design which always worked with the full order 'truth model',

2) insensitivity to phase behavior is not guaranteed, so trouble with the phase mismatch of the model, as well as with phase deviations in the real plant (which have been observed and are due for instance to temperature-dependent stress) could be expected,

3) due to the 20 dB/decade slope of the open loop when true lqg state feedback is employed, problems with the high-frequency resonance peaks of the 'truth model' and the real plant were thought to be likely.

A. Design Model

The order of the design model determines directly the order of the final dynamic compensator/controller which consists of an observer or a Kalman filter with state feedback. In order to be comparable to the classical controller and to keep the control processor's workload reasonably low, a design model of 8th order was derived. Figure 3 shows the frequency responses of both the 'truth model' and the design model. There is good matching only up to 3 kHz. Even to achieve this, it was necessary to include the 7-kHz resonance in the design model, because it had much 'stray influence' into the lower frequency range.

Since the range of good matching is fairly small with respect to the projected crossover frequency range of 600–900 Hz, we had to be prepared to face robustness problems when applying the lqg controller to the 'truth model' or to the real plant.

This design model has been augmented by an integrator, the output of which adds to the control input. This integrator models a constant torque disturbance. This disturbance can be observed by the Kalman filter and the estimate fed forward to the control input. The final design model thus was of 9th order both for the feedback and the Kalman filter design.

B. Open-Loop Shaping

As noted above we expected to run into problems with the high-frequency behavior of the true plant, which is not well represented in the design model. We therefore aimed first at forcing rapid rolloff of the open-loop frequency response beyond the projected crossover-frequency into the controller design.

A viable method of doing this is

1) put a low-pass filter into the loop at the plant's input,

2) design state feedback and Kalman filter for the augmented plant,

3) implement the controller structure from Fig. 4.

This approach corresponds to the 'frequency-shaped cost functionals' technique given in [2] (particularly example 4).

It might seem to be a problem that the filter states themselves are also fed back and thus the filter characteristic changes. We chose the corner frequency of the 2nd-order filter near to the desired crossover frequency (usually a bit lower) and there was only weak feedback of the filter states. The filter's poles were only slightly shifted and the desired rolloff was achieved without significant loss of control bandwidth, as shown in Fig. 5. The small effect of filter feedback can be explained by the cost which any larger changes in the filter dynamics would introduce, which are therefore avoided by the lq-optimal feedback design as long as the control bandwidth is not forced to be much higher than the filter corner frequency.

It turned out however that open-loop shaping was not really

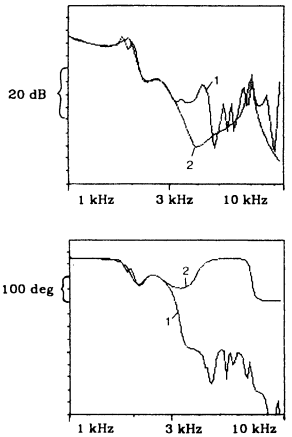


Fig. 3. Frequency response truth model (1)/design model (2).

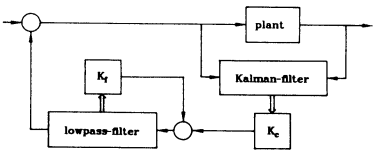


Fig. 4. Control structure with open-loop sharing.

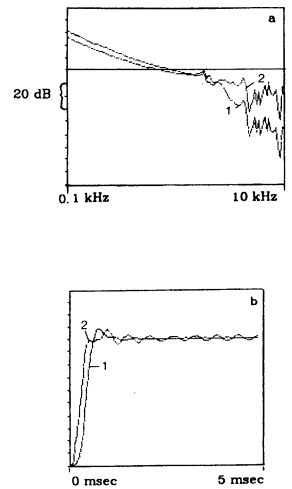


Fig. 5. Results of open-loop shaping: (a) Open-loop frequency response magnitude, (b) step response, (1) with open-loop shaping, (2) without open-loop shaping.

necessary for our drive actuator control. The controller described below had moderate but sufficient roll-off without this technique, yet open-loop shaping might be necessary when manufacturing tolerances or the like require a more safe design. The oscillation visible in Fig. 5(b), which results from the near 0-dB resonance peak at 2 kHz, could also be avoided by weighting a suitably defined variable in the lq cost functional (see below).

C. lgq/ltr Design

The ltr method of designing a state-feedback plus Kalman filter (observer) based controller is now well established and already appears in textbooks, for instance [3]. In effect it means introducing fictitiously high process noise at the control inputs (one in our case). Increasing this noise forces the Kalman filter to rely more and more on the measurements, thus using the control input information less and less. In the limit case this information is no longer used at all. The loop transfer functions of the original full state feedback without Kalman filter are then 'recovered'. The purpose of this strategy is to make the control loop less sensitive to certain mismatches between the plant model used in the Kalman filter design and the real plant.

The limit case is however not practically useful, because it yields an ultrafast filter with too noisy estimates. A compromise has to be found. Our strategy generally is to observe the filter poles when increasing the control input noise and to locate the poles somewhere in the region which corresponds to the desired dynamics of the control system. The pole corresponding to the disturbance model is also located in an appropriate region through a suitable disturbance noise intensity.

Figure 6 shows the control system structure. Due to the ltr procedure no problems arose with using the equivalent SISO compensator instead of the original 2-input controller with the control signal being explicitly fed into the Kalman filter. Note that without the fictitious ltr noise the Kalman filter would have relied far more on the control signal than on the measurement. In such cases the compensator may turn out to be unstable, so that the control system becomes only 'conditionally stable', which is very undesirable.

The result of the first attempt to design the state feedback and the Kalman filter is shown by the step response in Fig. 7(a). This response was simulated with the design model as plant, i.e., without any model mismatches. The state feedback had been designed with cost function weight on the control signal and the head position error.

The problem with this design was the 2-kHz oscillation visible in the step response. It is typical of lqg designs that lightly damped plant modes do not always come out well damped in the closed loop. Even if the damping achieved here were considered to be sufficient, it was unfortunately not retained when the controller was applied to the truth model of the plant. In fact, the oscillation built up, and the closed loop was unstable. To remedy this it was necessary to force the lqg design to yield more damping of the critical mode.

D. Modal Weighting

In resonant mechanical systems it is often necessary to achieve higher damping than given by the lqg design when weight is put only on the controlled or related variables [4]. Frequently the mechanical motion dominantly associated with a critical mode can be identified, such as the relative motion of

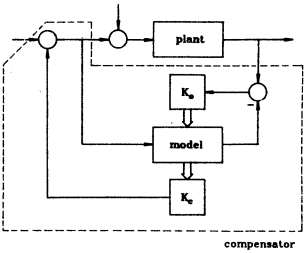


Fig. 6. Control structure.

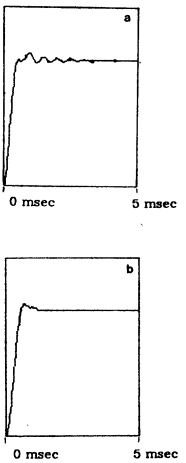


Fig. 7. Step responses simulated with design model: (a) without modal weighting, (b) with modal weighting.

two masses connected by a spring. Then it may suffice to put cost function weight on appropriate variables related to this relative motion in order to get good damping without affecting the eigenfrequency or other eigenvalues too much.

In the disk drive application we unfortunately do not have an appropriate mechanical plant description, we only have a black-box input/output model. It is however still possible to apply 'modal weighting', by introducing an auxiliary output variable y_M (one per critical mode) which only reflects the critical mode in the transfer function from the control input u to y_M, and does this in a certain manner, i.e.

$$\frac{Y_M(s)}{U(s)} = \frac{ks}{s^2 + 2\zeta\omega_0 + \omega_0^2}. \tag{1}$$

Finding this auxiliary variable here requires computing the eigenvectors of the plant state space system matrix. Combining the two conjugate complex left eigenvectors l_c^T and l_c^{*T} of the critical mode by

$$c^T = c_1 l_c^T + c_2 l_c^{*T} \tag{2}$$

with appropriate constants c_1, c_2 and defining

$$y_M = c^T x \tag{3}$$

(x being the state vector) yields the required auxiliary output variable.

It is crucial to take the constants in such a way that the 's-term' is in the numerator of (1). It can be shown [4] that this ensures that damping is achieved by weighting y_M without affecting the eigenfrequency too much. Strictly spoken, it is possible to move the critical eigenvalues exactly along the 'constant eigenfrequency/more damping-path' by weighting y_M, provided that not the eigenvectors of the uncontrolled plant are used, but those of the already controlled (but insufficiently damped) plant. If the critical eigenvalues have not been affected significantly in the previous control design, it may be assumed that the eigenvectors are almost unchanged too. Then it is more convenient, and sufficient, to use the plant's eigenvectors, and this we did.

With this 'modal-weighting' technique we achieved sufficient damping very easily (Fig. 7(b)), and this then carried over to the truth model based simulation, and later on to the real implementation (see below).

IV. Implementation Results

For the implementation of fast controllers we routinely use our own TMS 32010-based digital signal processing system along with a set of design and implementation software tools, including an automatic code generator [5]. We carried out the design in the analog domain, and discretized the controller with methods briefly described in [6]. The discretized controller given in state space form was then transformed to 'real modal form' and was scaled automatically for 16-bit fixed point arithmetic by a l_1-scaling technique [6]. After checking for the effects of discretization, computational delay, AD- and DA-quantization, and finite wordlength arithmetic by simulation, the signal processor code was automatically generated and downloaded. The sampling rate was about 34 kHz.

Figure 8 shows the step response result obtained with the truth model of the disk drive by simulation with the all-digital controller as mentioned above. Figure 9 shows the same response measured from the real drive. In contrast to the results in [1], where the experimental and the simulated response were very close, we observe considerable deviations here. The reason lies in the higher sensitivity of the lqg controller mainly with respect to plant phase as compared to the 'amplitude-stabilization'-type design of [1]. The mismatch between our design and truth models and the true plant is too high for a controller whose design relies too much on the fidelity of the model. Nevertheless, the controller works, and it has some advantages, as discussed below.

First of all, to our own surprise, the lqg controller was much quieter than the controller from [1] in the experiment. The high reduction of audible noise can be explained by the controller's magnitude frequency response given in Fig. 10. The high-frequency gain is much lower than that of the PDPD-notch-type controller from [1]. Secondly, the disturbance response is about twice as fast as with the controller from [1]. When the controller gain was increased by 20 percent, the disturbance response was even faster (Fig. 11), and reference signal step response was also improved somewhat, without running into robustness trouble.

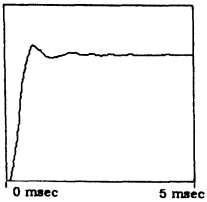


Fig. 8. Simulated step response with truth model and digital controller.

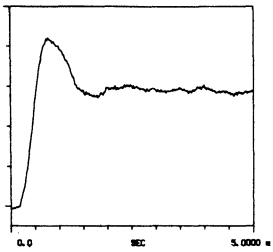


Fig. 9. Measured step response.

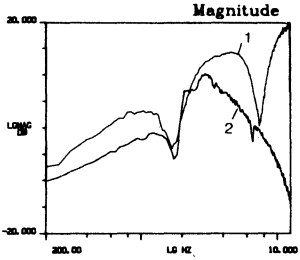


Fig. 10. Controller frequency response comparison: (1) double PD & notch-type controller from [1], (2) lqg controller.

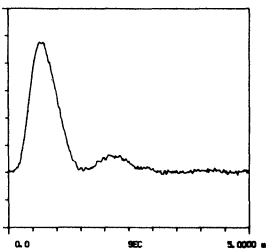


Fig. 11. Disturbance step response.

In order to show that the design efforts were worthwhile compared to a quick design of a simple controller we show the step response obtained with a PD controller in Fig. 12. Note

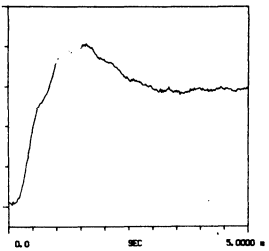


Fig. 12. Step response with simple PD controller.

that the dynamics of this control system would become worse if an integrator were added for achieving constant disturbance rejection, which the controller from [1] and the lqg controller already had built in.

V. Conclusions

We have shown that lqg design, if done properly, can be used to obtain reasonable controllers for highly resonant mechanical systems such as the disk drive. Designs can be completed systematically and quickly, as long as there are no problems with model fidelity. There are means to achieve robustness against high-frequency mismatch (the low-pass filter technique) but in the medium frequency range the model should be good.

There is still much more potential in the lqg design of fine-positioning control. Accurate modelling of disturbances (harmonic ones from disk rotation [7], stochastic and shock disturbances from the environment) promises to be beneficial. The resulting increase in controller order should not be a severe obstacle with today's low-cost high-performance signal processor devices and suitable software support.

References

[1] H. Hanselmann and W. Moritz, "High bandwidth control of the head positioning mechanism in a Winchester disk drive," in *Proc. IECON'86*, Milwaukee, 1986; also accepted for the special IECON issue of *Control Syst. Mag.*, Oct. 1987.

[2] N. K. Gupta, "Frequency-shaped cost functionals: Extension of linear-quadratic-Gaussian design methods," *J. Guidance Contr.*, vol. 3, no. 6, pp. 529-535, 1980.

[3] B. Friedland, *Control System Design*. New York: McGraw-Hill, 1986.

[4] H. Hanselmann, "Achieving damping in lq control of resonant mechanical systems," to be published.

[5] H. Hanselmann, "Using digital signal processors for control," in *Proc. IECON'86*, Milwaukee, 1986.

[6] H. Hanselmann, "Implementation of digital controllers—A survey," *Automatica*, vol. 23, no. 1, pp. 7-32, 1987.

[7] T. Tsujisawa, H. Murayama, and H. Inada, "Modern control theory application in high track density FDD," in *Proc. IECON'86*, Milwaukee, 1986.

HIGH BANDWIDTH CONTROL OF THE HEAD POSITIONING MECHANISM IN A WINCHESTER DISC DRIVE

Herbert Hanselmann and Wolfgang Moritz

University of Paderborn, Department of Automatic Control in Mechanical Engineering
4970 Paderborn, Federal Republic of Germany

ABSTRACT

Modern disc drives use fast voice coil actuators for the positioning of magnetic heads onto desired tracks and keeping them on track against various disturbances by closed-loop control. Problems stem from high desired control bandwidth (ca. 1 kHz) requiring digital signal processing rates above 10 kHz and from complexity due to structural mechanics effects.

1. INTRODUCTION

Modern disc drives use fast voice coil actuators for positioning magnetic heads on desired tracks and keeping them on track against various disturbances by closed-loop control. Two types of actuators are predominant in state of the art drives: rotary and linear actuators, both driven by a current passing through a coil in a strong magnetic field.

In high performance drives the head position is measured from a dedicated servo platter. Measurement electronics supply a head / track misalignment error voltage which is proportional to this error within track width. Current flowing through the coil generates torque or force so there must be closed-loop control.

Head positioning control comprises two tasks: Positioning on a target track (maybe across many tracks), and fine-positioning. The former task constitutes a servo problem (nonlinear for large initial / target track distances), whereas the latter is a regulator problem.

We concentrated on fine-positioning control in an industrial prototype 8" drive using a rotary actuator, as shown in Fig. 1.

2. SETUP

Initial studies had been performed on the actuator assembly separated from the drive housing. Because of considerable interaction of the actuator with the housing (base-plate) and platter/spindle assembly it was however necessary to use a more complete drive.

The setup now is an almost complete drive without top plate, with fixed spindle, and with an optical sensor for head position. Fixing the spindle of course means that several effects which exist in the operating drive are not accounted for. These are: (a) vibration of the (Whitney-type) slider / head assembly (Mizoshita et al., 1985); (b) aerodynamic suspension of the flying head; (c) vibration from spindle / bearing inaccuracies (Naruse et al., 1983); (d) noise from misalignment detection electronics.

The head position range in our setup is however realistic. The optical sensor consists of a small plate with a borehole in place of the head which moves between a differential photo-diode and an LED mounted on adjacent platters. Resolution and linearity is very good so that we can operate in the original track width position range (+-18 μm maximum, +-9 μm used) with sensor noise in the 0.2 μm rms range.

The magnetic actuator is driven by linear current driver electronics so that the torque is directly proportional to the control input voltage to the driver.

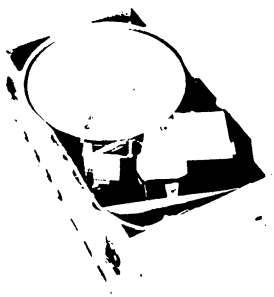


Fig. 1 disc drive setup

3. MODELLING

In order to study control, a mathematical model of the plant is necessary. A simple mathematical model would be a double integrator (torque to position). But the control bandwidth desired is so high that structural mechanics effects can by no means be neglected.

Frequency response. Fig. 2 shows the measured frequency response from input current to position in the 1 Hz to 10 kHz frequency range. There are many resonances and notches, and zoomed analysis would show even more. An interesting fact is that obviously the transfer function of such a mechanical system can be of nonminimum phase type. This can be concluded from the phase behaviour around 3 kHz, where the deep amplitude notches are not accompanied by a phase lift. In fact phase goes down 360 degrees. We did not expect this but it can been shown that some kind of mechanical vibration mode can indeed lead to such behaviour, which is expressed by pairs of conjugate complex transfer function zeroes in the right half-plane. Such behaviour is known to be undesirable for control.

The second curve in Fig. 2 shows the frequency response as computed from an input-output (black-box) model with a 30th order transfer function. This

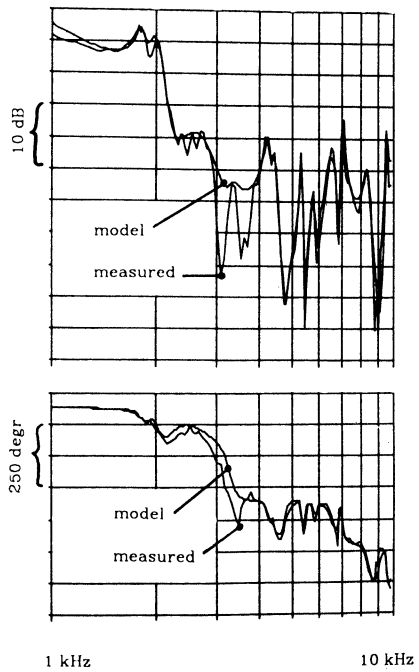


Fig. 2 plant frequency response

model has basically been formed from resonance frequency, damping, and residue data obtained with the curve fitting facility of the structural dynamics analyzer we used. Additional tedious manipulations have however been necessary in order to improve the model in both phase and amplitude response. The model is fairly good below 2 kHz and above 4 kHz, but less so between these frequencies. In particular the two deep notches between 3 and 4 kHz are not well represented in the model, and correspondingly there is considerable phase mismatch. Because we focussed on gain stabilization (see next section) we accepted this for the time being. But certainly we should do some work on improving the model by better computerized model matching methods.

For frequency response computation, control design and simulation, the model has been formulated in state-space parallel form.

Modal analysis. It is interesting to know with which vibrational modes the significant resonances are associated. So we performed experimental modal analysis using a structural dynamics analyzer (in fact modal analysis was the first step in the study and gave important clues as to where to improve the mechanical construction).

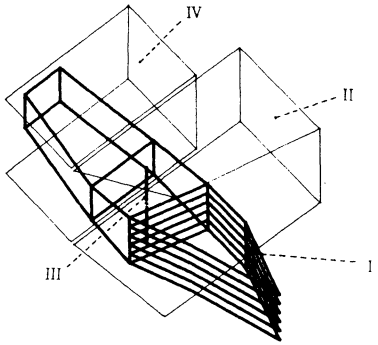
The first step is the measurement of the frequency response between the control input (torque or current) and several accessible points of the mechanical structure. Fig. 3 a) shows such points for a 3D model of the actuator. This model was used in previous studies of the actuator where it was not built into the drive housing. The rotary arm for 6 platters is shown in bold lines along with the voice coil (I). The mounting frame (II) is shown in thin lines. It carries the bearing for the rotary arm's shaft (III) and is screw-mounted onto the baseplate. The magnet (IV) is mounted on the frame.

Vibrational deflection was measured via subminiature accelerometers, weighing only 0.6 grams. The modal analysis for the whole disc drive was restricted to a 2D study because only the top level of the arm is then available for measurements. Figs. 3 b) - e) show the 2D model used with dotted lines for the undeflected state and bold lines for the deflections associated with each vibrational mode.

Each resonant peak in the frequency response of such a weakly damped mechanical structure is associated approximately with a real vibrational mode. Such modes can be determined by the structural dynamics analyzer via frequency response curve fitting around the resonances. It is possible to get animated pictures of the individual modes on the screen, which do not however show the actual movements of the structure under some excitation, but only the form of the contributions of the individual modes.

Some of the modes which are relevant in control design are shown in Figs. b) - e). The lowest resonance frequency (1.78 kHz, Fig. 3 b)) is associated with a bending mode of the arm, as one might expect. But frame and magnet, although being very solid and heavy, also contribute to this mode. The nearby resonance at 2.08 kHz (Fig. 3c)) also belongs to a bending

mode, but magnet, frame and bearing do not contribute significantly to this movement. It is typical of the high frequency modes (Figs. 3 d) and e)) that the deflections at the tip of the arm are very small compared to those of the voice coil on its carrier construction.

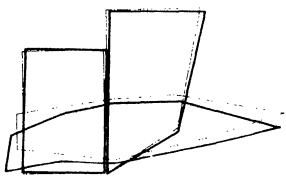


a) view of actuator with bearing and magnet

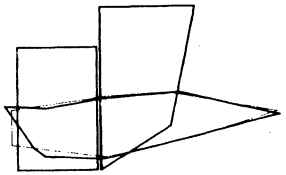
Possible deflections of the top platter were also investigated, because track misalignment depends on the movement of the slider at the tip of the arm in relation to the platter. Previous analyses, where the platter spindle had not been fixed as in this setup, indicated that there can be such relative movements of the platters at low frequencies, even though the spindle is mounted several inches away on the baseplate. In this setup we could not however observe significant platter movements. Deflections of the individual arms (there are 6) relative to each other were found in previous analyses, but were not investigated here.

Some discrepancies between the frequency response from the control input to the slider's position and the responses measured via the accelerometer (after double integration in the frequency domain) must be attributed to the mass of the accelerometer.

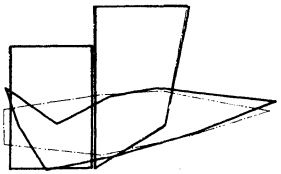
It should be clear that experimental modal analysis reveals most efficiently the coupling of the components of the mechanical structure. It gives information needed for incorporating the important effects in rigid body or finite-element modelling, and gives clues as to where improvements to the construction could be made. In our case it should be beneficial to fix the magnet-to the baseplate more rigidly, and to have a more rigid voice coil construction, which would presumably prevent large amplitude high frequency resonances which pose difficulties for high bandwidth control (see below).



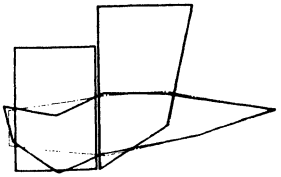
b) 1.78 kHz mode



d) 7.13 kHz mode



c) 2.08 kHz mode



e) 9.55 kHz mode

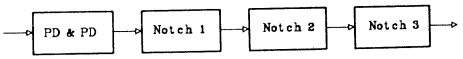
Fig. 3 vibration modes of actuator

4. CONTROL

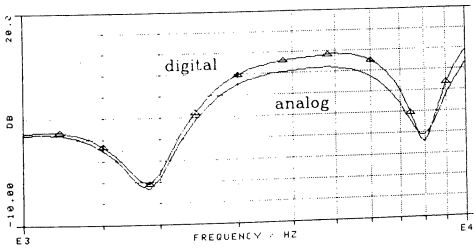
Fine-positioning control may seem to be very easy because the plant is SISO (single-input single-output) and is virtually linear. Linearity is only lost when the current saturates. It turns out that the current limit is not likely to be violated with fine-positioning regulating control because the current range is designed for fast large-distance positioning at high torque. The plant is SISO because for economic reasons the only measurement presently available for control is the track position error itself.

The difficulties of control arise mainly from the structural resonances, the very uneven phase response, and the nonminimum phase behaviour. High control bandwidth, which is desired for disturbance suppression and quick response, can only be achieved by insertion of phase lead (PD-type controller) near the desired crossover frequency. This in turn makes the structural resonances more significant due to increased high frequency gain in the loop, thus leading to stability problems. Resonance peaks coming close to the 0 dB level in the open-loop amplitude response (gain stabilization) should be avoided, because it cannot be guaranteed that phase will stay off the $-\nu*180$ ($\nu=1,3,\ldots$) degree levels at the resonance peaks. This applies even if the measured frequency response shows acceptable phase margins, because plant phase is quite sensitive to slight alterations in the mechanical structure even with one given drive. Problems would be magnified if the controller would be applied to series drives with manufacturing tolerances.

Classical control. A solution within the scope of classical control comes from structural notch filters. This technique is becoming common now, particularly with the control of flexible mechanical structures in space. The controller shown in Fig. 4 is composed of



a) block diagram



b) frequency response

Fig. 4 classical controller

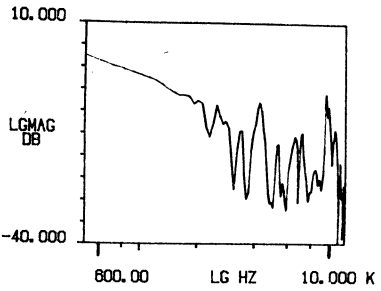


Fig. 5 loop frequency response

two PD-type blocks and three notch filters in series. The notch filters make the resonances at 2, 4, and 7 to 9 kHz sufficiently "invisible" in the loop (Fig. 5). The loop gain has been chosen so that crossover frequency is around 900 Hz. However the phase margin is then quite low, so the command step response in Fig. 6

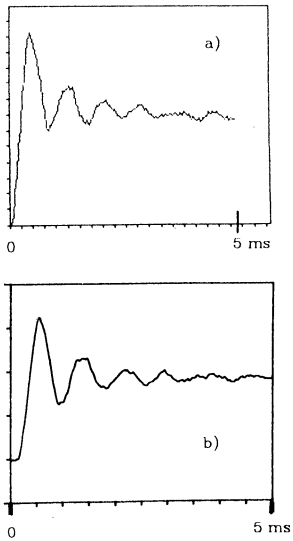


Fig. 6 step response
a) simulated
b) measured
(note slightly different time scale)

shows large overshoot and oscillation. This might be considered unsatisfactory, but is of secondary importance because the focus is on regulator behaviour (not servo), and, if desired, a prefilter (with complex zeroes) in the command path could easily eliminate both overshoot and oscillation while retaining fast rise time. A slight gain reduction also eliminates oscillation, but overshoot remains large.

Note that the simulation result in Fig. 6 corresponds fairly well to the measured response. This is because of the high quality of the plant model and our 'near-to-reality' simulation concept, including implementation effects from processor arithmetics, non-simultaneous sampling, AD- and DA-conversion, and measurement noise.

The controller has been implemented in state-space form

$$x_{k+1} = A\,x_k + B\,u_k$$
$$y_k = C\,x_k + D\,u_k \qquad (1)$$

on a TMS 32010 signal processor at 30 kHz sampling frequency using the CACE-system described in a companion paper (Hanselmann, 1986). A natural form of implementation would have been to realize each of the 2nd order blocks from Fig. 4 a) in a series connection (cascade structure). In order to minimize computational delay between sampling and output we prefer form (1). It proved however beneficial to choose a series connection of 2nd order blocks for the structure of A. The parallel form (Hanselmann, 1987) which we picked first was suffering heavily from state-variable quantization noise, although the eigenvalues of A seemed to be sufficiently spaced from each other (otherwise problems would have been no surprise).

Synthetic disturbance feedforward. Although there are two integrators in the plant, this does not ensure rejection even of constant disturbances. There is for instance some almost constant torque in the rotary actuator from a spring moving the actuator to the landing position when power is off. This torque acts as a disturbance at the plant's input and leads to some μm of constant deviation of head position.

Instead of incorporating an integrator into the controller from Fig. 4 a), which would cause stability problems, we chose to make use of synthetic disturbance feedforward. This means definition of a disturbance model (an integrator for constant disturbances), design of a disturbance observer, and determination of the gain factors (only one in the case of the simple integrator), with which the observed disturbance is fed forward in order to compensate for the real disturbance.

The usual way to do all this unfortunately requires building a full observer for the plant plus disturbance model. Doing this for the 30th order drive model for the sole purpose of disturbance feedforward is clearly undesirable, and certainly also not a trivial task. Therefore we simply built the synthetic disturbance feedforward system around the original closed-loop system (Fig. 7). This disturbance compensation system

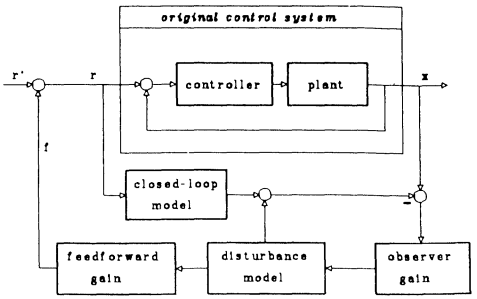


Fig. 7 synthetic disturbance feedforward

contains a model of the closed-loop control system, an observer observing the disturbance as it seems to act at the output, and feedforward to the former reference input of the original control system. Note that the nature of the physical disturbance and the point where it acts on the plant in reality need not be known. It is sufficient to know and model the effect the disturbance has at the original control system's output.

The closed-loop model can be quite simple. This is because in the closed-loop response there is not much influence of the resonances of the plant, due to the notch type controller. We used a 2nd order model according to the step response shown in Fig. 6. The quality of this model determines how fast the disturbance feedforward can be. If the model matches the actual closed-loop response perfectly, then the disturbance observer is never affected by the original reference input variable r, and the observer which determines the speed of the disturbance compensation can be designed to be arbitrarily fast. This is of course not the case here, but, as Fig. 8 shows, the disturbance feedforward is nevertheless satisfactorily fast, much faster than necessary for compensating the abovementioned torque.

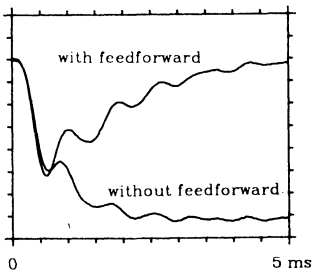


Fig. 8 step disturbance response (measured)

Note that for the constant disturbance considered here the model is a simple integrator; so feedforward gain, disturbance model and observer gain together form a 1st order lag system. And since there is unity gain in the original closed-loop transfer function from r to x, the disturbance compensating transfer function from x to f is simply

$$G(s) = \frac{-1}{(1+Ts)} \qquad (2)$$

where the single design parameter T determines how fast the disturbance compensation works. The timeconstant selected should not be too small because due to mismatch between the closed-loop model and the real closed-loop system the poles of the original closed-loop system would be affected. This would not be the case with perfect model match.

Controller optimization. We also subjected the controller from Fig. 4 to a parameter optimization using the program described briefly by Lückel et al. (1985). The main aim of this optimization was to get information on what we sacrificed by applying gain stabilization when we designed the original controller. Thus we let the optimization program vary all 17 controller parameters in order to minimize the rms value of the return difference (tracking error), where the reference input was white noise filtered by a 1 s timeconstant lowpass filter (thereby approximating the spectrum of a step input).

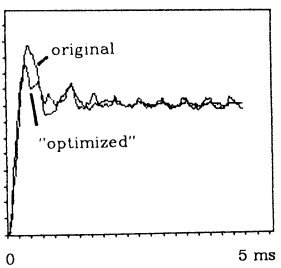


Fig. 9 effect of controller "optimization"

Fig. 9 shows the step responses of the control system both with the original and the "optimized" controller (analog versions). Because in the parameter optimization no restriction to gain stabilization was made, the optimizer was free to exploit the phase (phase stabilization) and lifted some resonances considerably. This saved some negative phase contributions from the controller and made somewhat faster control possible. The effect is clearly seen in the high frequency contents of the "optimized" step response due to resonances from the plant, which had been kept low in the original design.

5. CONCLUSIONS

It has been shown that the strongly resonant disc drive head positioning system can be digitally controlled with high bandwidth. The requirements are: a good model, appropriate controller design, and implementation tools for fast controllers.

From an analysis of the vibrational modes which posed difficulties in the controller design, improvements to the mechanical construction can be suggested, which may lead to a further increase in control bandwidth.

Acknowledgement

Many of the computer and measurement results presented here have been supplied by cand. ing. A. Engelke, who carried out the computer studies and realizations during his diploma thesis work.

REFERENCES

Hanselmann, H. (1986). Using digital signal processors for control. *Proc. IECON'86*, Milwaukee.

Hanselmann, H. (1987). Implementation of digital controllers. *Automatica*, survey paper, accepted for publication.

Lückel, J., R. Kasper and K. Jäker (1985). Interactive optimization of controller and plant parameters in the case of multiple design objectives. *Proc. 3rd IFAC/IFIP Symp. Computer Aided Design in Control and Egineering Systems.*, Lyngby, Denmark.

Mizoshita, Y., K. Aruga and T. Yamada (1985). Dynamic characterstics of a magnetic head slider. *IEEE Trans. Magnetics*, **MAG-21**, 1509.

Naruse, J., M. Tsutsumi, T. Tamura, Y. Hirano, T. Hayama and O. Matsushita (1983). Design of a large capacity disk drive with two actuators. *IEEE Trans. Magnetics*, **MAG-19**, 1695.

Fast Access Control of the Head Positioning Using a Digital Signal Processor

S. Hasegawa, Y. Mizoshita, T. Ueno, K. Takaishi

Fujitsu Laboratories Ltd. File Memory Laboratory
10-1 Morinosato-Wakamiya, Atsugi 243-01, Japan

ABSTRACT

We have developed a new digital servo controller for a 5" hard disk drive which has average access time of 10 ms for a 25 mm stroke. To obtain this fast access speed, we used a state estimator with a new acceleration trajectory model. The estimator and trajectory generator are implemented using a digital signal processor.

There are two problems for fast access control: motor coil inductance and the mechanical resonance of the actuator and disk enclosure. To solve these problems and to achieve precise head positioning, we developed the following control method.

To solve the voice coil motor inductance and actuator resonance problems, we used a new acceleration trajectory model which is not affected by the coil inductance when the head moves quickly. This design is based on an optimal control theory which minimizes the square of differentiated acceleration. By using this new trajectory model, the high harmonics of actuator drive are damped and the residual vibration of actuator immediately after access is decreased.

1. INTRODUCTION

Direct access storage devices (DASD) are required to have faster and faster access speeds, be smaller and smaller, and have larger storage capacity. To satisfy these requirements, we have to figure out how to achieve the high speed access and precise positioning at the same time. Mechanical resonance of the DASD becomes important and limits the file access speed.[1,2]

The mechanical resonance problem is divided into two parts associated with the frequency band : low and high frequency vibration. Vibrations below the servo bandwidth (300 ~ 700Hz) are caused by forces through the shock absorbers which support the head disk assembly base and the reaction force of the actuator when it moves quickly.

High frequency vibration (bandwidth above servo bandwidth) is caused by mechanical resonance of the actuator, which is composed by magnetic heads, sliders, gimbals and head arms.

Low frequency vibration can be controlled by using a state estimator to estimate the mechanical resonance of the plant. Unfortunately, it is impossible to make an active controller using only feedback for high frequency vibration. Smooth acceleration and deceleration in the seek are not the cause of the high harmonics.

We propose a new acceleration trajectory which is based on the movement of the human body.[3] In this model the trajectory is determined as minimizing the square of differentiated acceleration. There was an example of application to the adaptive control of a robot arm.[4]

Conventionally, the controller of the head positioning in hard disk drives has used a general purpose microprocessor. Recently the requirements for the advanced control and additional functions are increasing, and high speed digital signal processing is the way to achieve this control because the ability of the digital controller depends mainly on its sampling frequency. Thus, using a digital signal processor (DSP) ,which is fast, in the head positioning system has become common.[5,6,7,8]

2. CONFIGURATION OF THE DIGITAL SERVO SYSTEM

Figure 1 shows the configuration of the digital control system for the head positioning system we developed.

Double phase position signals (POSN,POSQ) are generated by the position transducer using the servo pulse from servo head. Each linear part of these position signals is selected by an analog switch and input to the A/D convertor whose conversion time is 3 μs. Track cross pulse is generated from these position signals and input to the counter.

At every sampling period, DSP calculates the current drive and outputs it to a power amplifier through the D/A converter. The TMS320C25 was chosen as the DSP because of its speed (one instruction per 100 ns) and price.

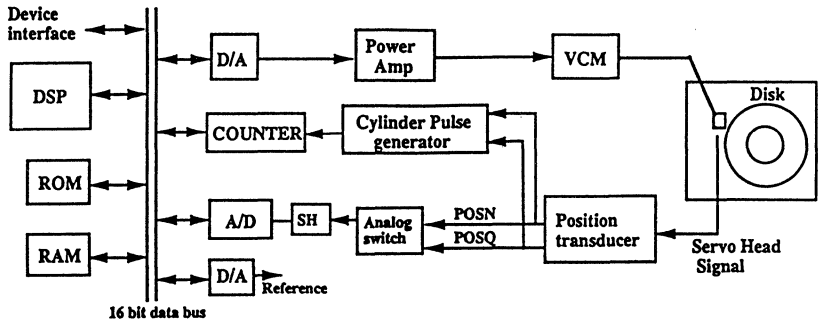


Fig.1 Digital servo system for head positioning

Table 1 lists characteristics of the control disk drive.

Table 1. Characteristics of the Control Disk Drive

Disk diameter	5"
Servo type	dedicated servo
Stroke	25 mm
Actuator type	rotary
VCM force constant	2.1 N/A
Actuator moving mass	10.1 g (equivalent)

Figure 2 shows mechanical transfer function of the actuator. It has mechanical resonance at 1.5 kHz, 2.1 kHz and 6.4 kHz, so we cannot set the servo band width over 700 Hz in this drive.

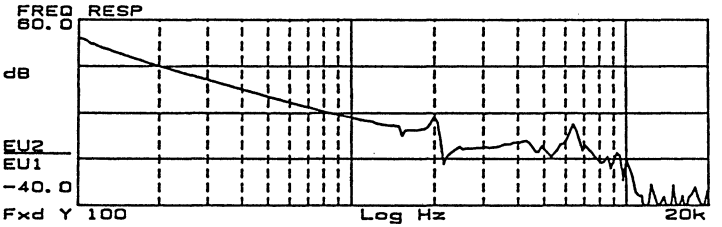


Fig.2. Mechanical Transfer Function

3. CONTROL SYSTEMS

3.1. State estimator model

We designed state space controller for head positioning. We assumed the plant as simple double integrator model. In this way, feedback states which are necessary for control are position and velocity. Among them, velocity is not measurable state. So we used state estimator model.

In the double integrator model, the digitized space equation can be stated in matrix form.

$$x(k+1) = \Phi\, x(k) + \Gamma\, U(k) \tag{1}$$

$$y(k+1) = H\ \ x(k+1) \tag{2}$$

where

$$x(k) = \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} \quad \Phi = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \quad \Gamma = \begin{bmatrix} g \\ 2g \end{bmatrix} \quad L = \begin{bmatrix} L_1 \\ L_2 \end{bmatrix} \quad H = [1 \quad 0]$$

We used following translation :

$$x_2(k) \ \text{<---} \ T\ x_2(k) \qquad g \ \text{<---} \ \frac{T^2}{2\ b}$$

T is sampling period and b is its mechanical gain. The estimator model can be stated as :

$$\hat{x}(k+1) = [\ \Phi\ - LH\ ]\ \hat{x}(k)\ + \Gamma\, U(k)\ + L\, x(k) \tag{3}$$

where $\hat{}$ means an estimated value. Matrix L is a correction factor, it is designed based on the response speed of the estimator.

3.2. Track follow operation

Figure 3 is a block diagram of the track follow control. To reduce average offset , integrated position x_0 is added to the feedback loop. The x_0 is the running sum of measured position. The control law is :

$$U(k) \ = \ - K_1\ \hat{x}_1(k) - K_2\ \hat{x}_2(k) - K_0\ x_0(k) \tag{4}$$

The feedback gain series K_1, K_2, K_0 was determined by pole placement.[9] Sampling frequency is 30 kHz, so the influence of phase lag, which depends on delay, is small.

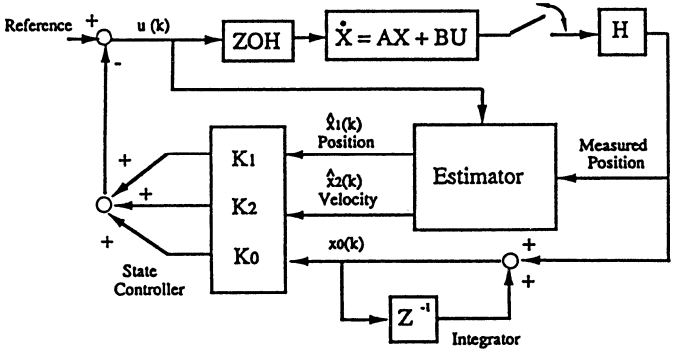


Fig.3 Tracking control

Figure 4 shows the open loop transfer function in tracking operation. Zero cross frequency is 530 Hz and the phase lead at this frequency is about 36 degrees.

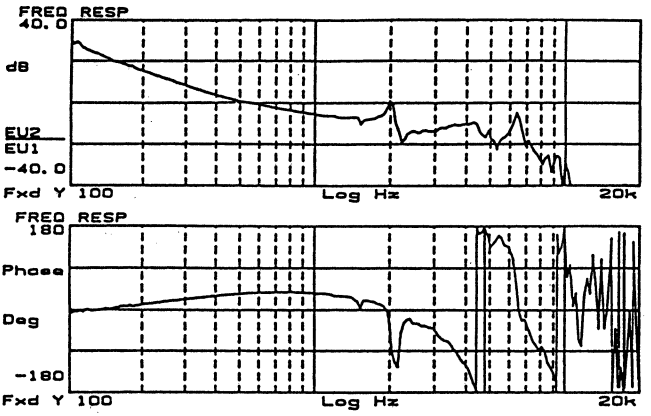


Fig.4 Open loop transfer function

3.3. Seek operation

Figure 5 is a block diagram of the seek operation. The sampling frequency is the same as track following. This operation creates a velocity feedback loop. The control law is

$$U(k) = K_v (V_{target} - \hat{x}_2(k)) + Kf\ x_3(k) \qquad (5)$$

where V_{target} is velocity trajectory profile and $x_3(k)$ corresponds to the feedforward signal.

In a conventional controller the velocity trajectory is taken from a calculated table. This table usually represents a desired velocity at a given distance from target track.[10]

In this control system, we did not use this table. We used a new velocity and acceleration trajectory to reduce excitation of the resonance during seek operation. We will explain the design of this new trajectory in the next section.

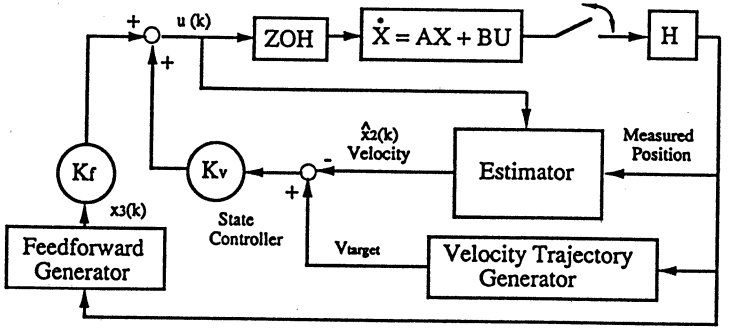


Fig.5. Seek control algorithm

3.4. Implementation to DSP

Figure 6 is a basic flow chart of the control program we developed. The designed sampling time (33.3 μs) was achieved by the timer interrupt function of the DSP chip. Tasks are scheduled by two subroutines : tracking and seek routines. In both routines, the DSP calculates the states from formula (3), and in the seek routine, it also calculates the velocity trajectory and the feed forward.

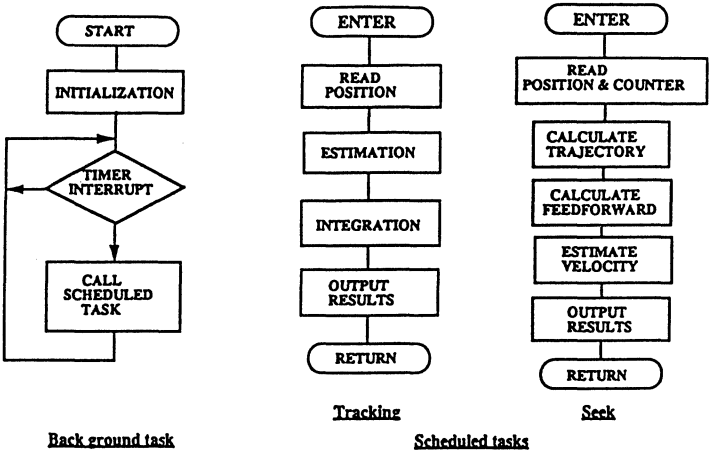


Fig. 6. Digital servo system using the DSP

4. DESIGN OF NEW ACCELERATION TRAJECTORY

4.1. Principles

First we consider the simple third order model of the actuator (Fig.7). The differentiated acceleration (dα/dt) is added to the basic double integrator model. u(t) is redefined as shown in Fig.7.:

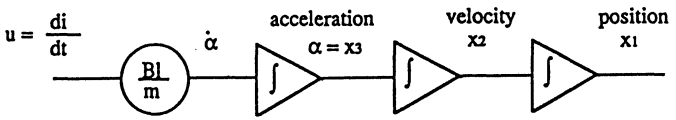


Fig.7. Third order model of the actuator

Thus the state equation is

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ Bl/m \end{bmatrix} u \qquad (6)$$

We then try to get solutions for minimizing the cost function P (equation(7)) with the initial condition (8a) and the terminal condition (8b).

$$P = \int_0^{T_0} u^2 \, dt \quad \left(= \int_0^{T_0} \left(\frac{d\alpha}{dt}\right)^2 dt \right) \tag{7}$$

$$x(0) = \begin{bmatrix} a \\ 0 \\ 0 \end{bmatrix} \qquad x(T_0) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{8a) (8b}$$

where a is the seek distance and T_0 is seek time.

Using adjoining state vector P, we use the Hamiltonian H

$$H = P^T (A x + B u) + u^2 \tag{9}$$

where the matrices are

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 \\ 0 \\ Bl/m \end{bmatrix}$$

Then optimal input u(t) is given as:

$$u = -\frac{1}{2} B^T P \tag{10}$$

and canonical equations are

$$\begin{bmatrix} \dot{x} \\ \dot{p} \end{bmatrix} = D \begin{bmatrix} x \\ p \end{bmatrix} \qquad D = \begin{bmatrix} A & -\frac{1}{2} B B^T \\ 0 & -A^T \end{bmatrix} \tag{11}$$

We define the characteristic equation of A and D as gA(s) and gD(s), their relation is

$$gD(s) = g A(s) \; g A(-s) \tag{12}$$

$$gD(s) = - s^6$$

Therefore the eigenvalue of Hamiltonian matrix D is

$$s = 0 \qquad (6\text{th root})$$

Consequently solutions for optimal state become 6th order time variable functions, and the unknown coefficients in the functions can be determined from the initial and terminal conditions (8a),(8b).

position $$x_1(t) = -60\, a \left\{ \frac{1}{10} \left(\frac{t}{T_0}\right)^5 - \frac{1}{4} \left(\frac{t}{T_0}\right)^4 + \frac{1}{6} \left(\frac{t}{T_0}\right)^3 \right\} \tag{13}$$

velocity $$x_2(t) = -60 \frac{a}{T_0} \left\{ \frac{1}{2} \left(\frac{t}{T_0}\right)^4 - \left(\frac{t}{T_0}\right)^3 + \frac{1}{2} \left(\frac{t}{T_0}\right)^2 \right\} \tag{14}$$

acceleration $$x_3(t) = -60 \frac{a}{T_0^2} \left\{ 2\left(\frac{t}{T_0}\right)^3 - 3 \left(\frac{t}{T_0}\right)^2 + \left(\frac{t}{T_0}\right) \right\} \tag{15}$$

Figure 8 shows calculated state trajectory from (13)(14)(15). Acceleration becomes sinusoidal and its peaks are located at t/T_0= 0.21, 0.79.

Figure 9 shows trajectories of position,velocity and other variables when actuator accesses the average access tracks (702 tracks). We verified that the average access time was 10 ms.

In this figure, current drive, feedforward drive, and velocity agree with the designed value of Fig.8 very well. The normalized time (t/T_0),which is output every sample period for reference increases linearly from the start of the seek. It suggests that the following movement occurs during seek operation.

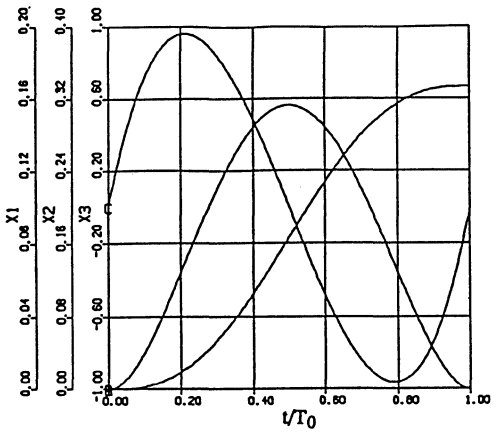


Fig. 8. Optimal state trajectories

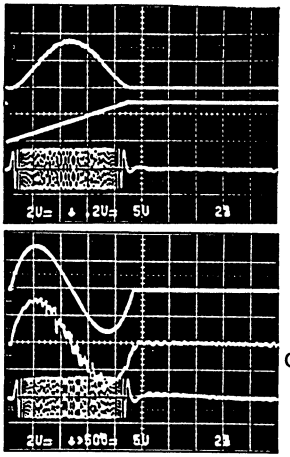


Fig. 9. Seek operation

4.2. Comparison with conventional trajectory

Figure 10 shows the difference between conventional trajectory and new developed trajectory. These trajectories are output from the reference DAC every sampling period.

In conventional seek control, the actuator is accelerated with full power amplifier until its speed reaches the desired velocity trajectory given in the table. In this case, therefore, the amplifier saturates at first stage of acceleration and the transient stage from acceleration to deceleration in specific track seek.

In our new seek control, desired trajectory is calculated in real time . And through full stage, smooth transient of current drive can be observed.

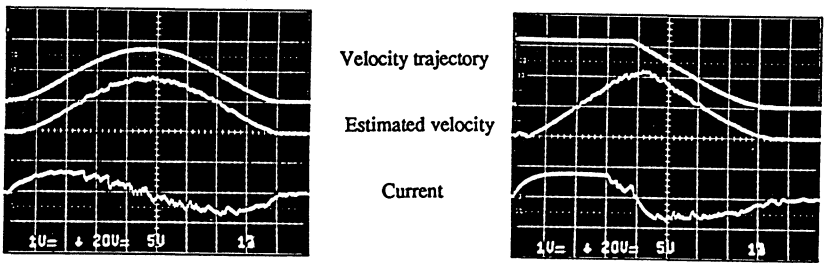


Fig. 10 New and conventional trajectories

4.3. Vibration reduction

Figure 11 shows simulation of acceleration and power spectrum against current drive for the three kinds of control : minimum time control (bang-bang control), conventional trajectory control, and our new trajectory control. The ideal minimum time cannot achieved because of the motor coil inductance.

In our trajectory control, the vibration is reduced. effectively. At 2 kHz, it can reduce power spectrum gain 20 to 30 dB lower than the others.

Figure 12 shows the vibration reduction in the time domain between these three controls. The residual vibration after access can be most effectively reduced by our trajectory (Fig. 13).

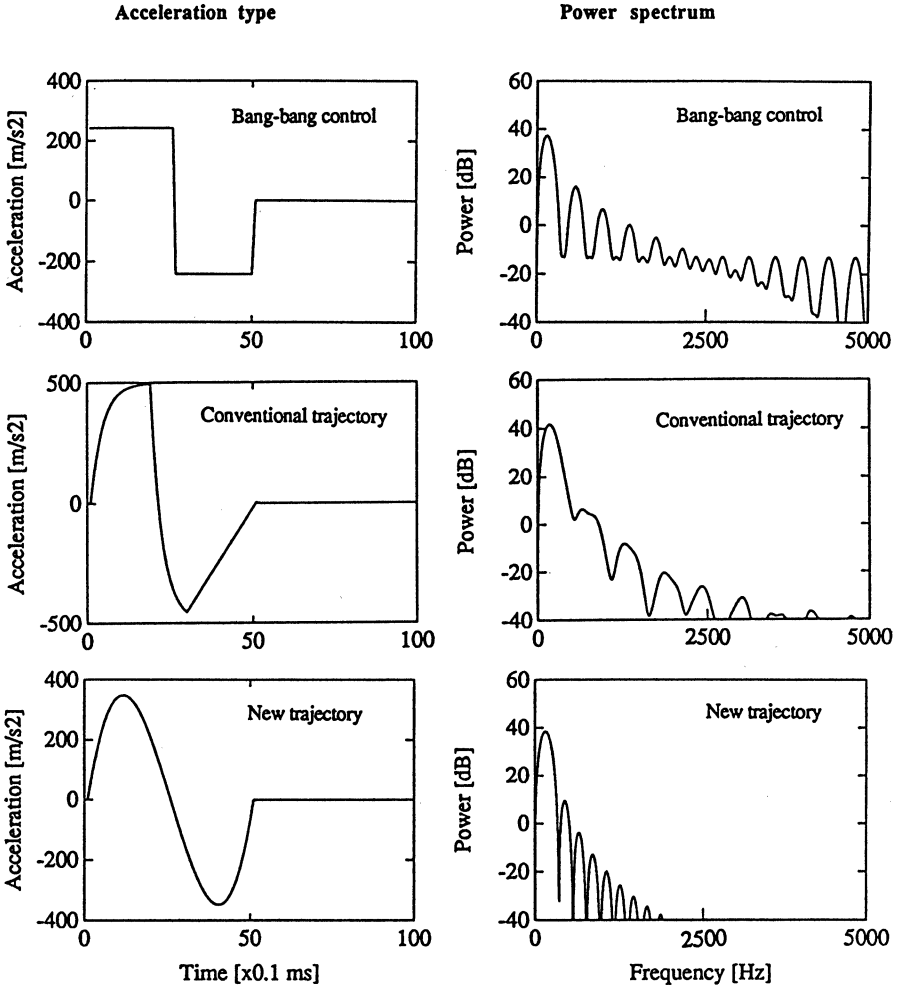


Fig. 11. Comparison with conventional trajectories (power spectrum)

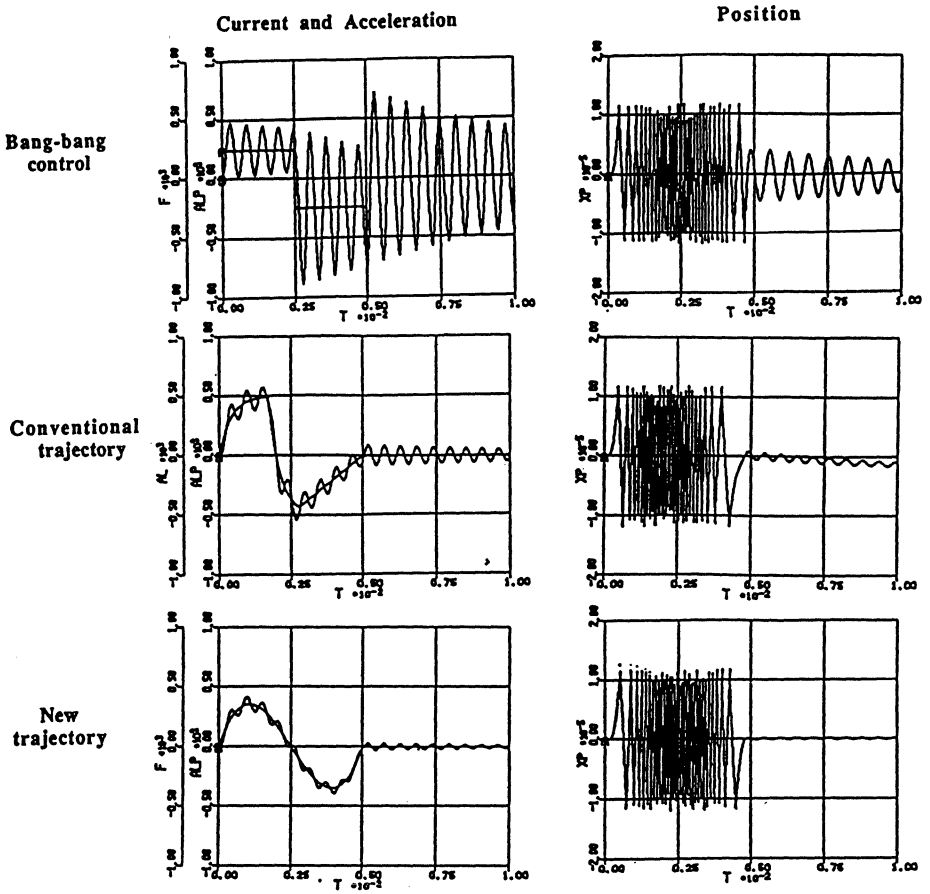


Fig. 12 Comparison with conventional trajectories (time domain : resonance = 1.8 kHz)

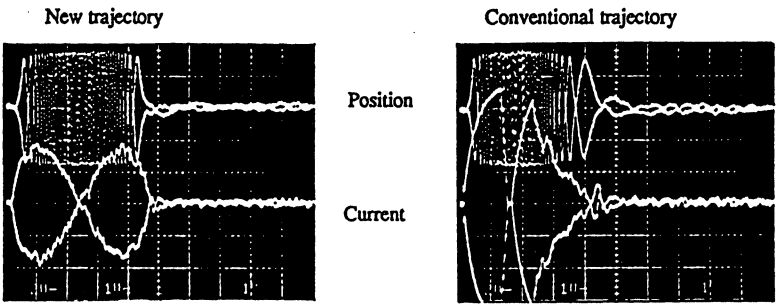


Fig. 13 Comparison of measured vibration after access (repeat seek : 100 tracks)

5. CONCLUSION

We have developed a new digital servo controller for a 5" hard disk drive. The state control with estimator was made using a digital signal processor. A stable tracking control and seek operation results in an average access time of 10 ms.

To avoid the high frequency mechanical resonance, which depends on current driving during fast access, we proposed a new velocity trajectory calculated to minimize the square of differentiated acceleration.

6. REFERENCE

1. I. Yamada and M. Nakagawa, Positioning Control of a Servomotor Mechanism with an Oscillatory Load, J. the Society of Instrument and Control Engineers, vol.18, 1,1982
2. K. Aruga, et. al, Acceleration Feedforward Control for Head Positioning in Magnetic Disc Drives, Trans. of ICAM 89, pp.19-24, 1989
3. T. Flash and N. Hogan, The Coordination of Arm Movements: An experimentally confirmed mathematical model, J. Neurosci. vol.5, pp. 1688-1703, 1985
4. M. Kawato, Mathematical Sciences, pp 76-83, no.289, July 1987
5. M. C. Stich, Digital Servo Algorithm for Disk Actuator Control, Conference on Applied Motion Control 1987
6. S. Hasegawa, et. al, Trans. of no.944 Symposium, pp.16-18, JSME, 1987
7. I. Ahmed and S. Lindquist, Digital Signal Processors Simplifying High-Performance Control, Machine Design, Sep. 10, 1987
8. H. Hanselman, Using Digital Signal Processors for Control, IECON'86, IEEE, 1986
9. G. Franklin and J. Powell, Digital Control of Dynamic Systems, Addison-Wesley, 1980
10. Y. Mizoshita and A. Futamata, Head-Positioning Servo Design for Disk Drives, Fujitsu Sci. & Tech. J., 18, 1 pp. 101-115, 1982

IMPLEMENTATION OF A MRAC FOR A TWO AXIS DIRECT DRIVE ROBOT MANIPULATOR USING A DIGITAL SIGNAL PROCESSOR

G. Anwar
Graduate Student

R. Horowitz
Assistant Professor

M. Tomizuka
Professor

Department of Mechanical Engineering
University of California, Berkeley, CA 94720

ABSTRACT

This paper is concerned with the digital implementation of a Model Reference Adaptive Control (MRAC) algorithm on a Texas Instrument TMS32010 Digital Signal Processor (DSP). The MRAC was designed to control a two axis direct drive SCARA type robot manipulator. The primary purpose of the adaptive controller is to compensate for the inertial variations due to changes in configuration and payload. Experimental results presented clearly illustrate the need for adaptive control over conventional PID controller for the type of arm structure used in the experiments. Discussion on the use of DSP in controls is presented in terms of their capabilities and the influence their performance will have on the sampling time of digital control systems.

1. INTRODUCTION

With the advent of direct drive robot manipulators, there has been a rising interest in the implementation of adaptive control to this particular class of robot arms [1],[2],[3], and [4] †. Direct drive robots, unlike indirect drive robots, are much more sensitive to configuration and payload changes, making them ideal candidates for adaptive control. Due to real-time computational speed limitation, much of the studies in adaptive control have been limited to mathematical analysis and computer simulation, however, it is now possible to implement adaptive control on direct drive manipulators with the availability of affordable high speed digital signal processors (DSP). It is the aim of this paper to present a digital implementation of a Model Reference Adaptive Control (MRAC) for a two axis SCARA-type robot manipulator using the TMS32010 from Texas Instruments. The paper will concentrate on the details of implementation and actual experimentation rather than the derivation of the adaptive controller. The details of the adaptive control design are referenced from our previous works [4].

The remaining sections of this paper are organized as follows, section 2 will briefly describe the adaptive control algorithm used, followed by a detail discussion of the implementation of the algorithm on the TMS32010 in section 3. Section 4 will discuss the experimental results, and the paper will conclude with section 5 discussing some of the advantages of using the TMS32010 DSP for real-time control applications.

2. ADAPTIVE CONTROL SCHEME

The two axis direct drive robot arm used for the experiments is shown in Fig. 1.0. The dynamic equations for such a two axis manipulator can be expressed as [1],[2],and [3],

$$\dot{\mathbf{x}}_p = \mathbf{x}_v \tag{1}$$

$$\mathbf{M}(\mathbf{x}_p)\dot{\mathbf{x}}_v + \mathbf{v}(\mathbf{x}_p,\mathbf{x}_v) + \mathbf{d} = \mathbf{q} \tag{2}$$

where $\mathbf{M}(\mathbf{x}_p)$ is the 2x2 inertia matrix which is symmetric and positive definite, $\mathbf{x}_p$,$\mathbf{x}_v$ and $\mathbf{q}$ are respectively the two dimensional joint displacement, joint velocity, and torque vectors. The vector $\mathbf{v}$ represents the nonlinear terms due to Coriolis and centripetal accelerations. The Coulomb friction torque vector is represented by $\mathbf{d}$.

† Number in the brackets designate the reference at the end of the paper.

Assuming that the torque vector $\mathbf{q}$ is preceded by a zero order hold, the dynamic equations (1) and (2) are discretized to

$$\mathbf{x}_p(k+1) = \mathbf{x}_p(k) + T\mathbf{x}_v(k) + \frac{T^2}{2}\mathbf{M}(k)^{-1}(\mathbf{q}(k)-\mathbf{v}(k)-\mathbf{d}(k)) \tag{3}$$

$$\mathbf{x}_v(k+1) = \mathbf{x}_v(k) + T\mathbf{M}(k)^{-1}(\mathbf{q}(k)-\mathbf{v}(k)-\mathbf{d}(k)) \tag{4}$$

where T is the sampling period.

Based on Eqs. (3) and (4) a Series Parallel Model [4] can be defined as

$$\mathbf{x}_{vm}(k+1) = \mathbf{x}_v(k) + T\mathbf{u}(k) \tag{5}$$

and the torque input vector is described by

$$\mathbf{q}(k) = \hat{\mathbf{M}}(k)\mathbf{u}(k) + \hat{\mathbf{v}}(k) + \hat{\mathbf{d}}(k) \tag{6}$$

where $\hat{\mathbf{M}}(k)$, $\hat{\mathbf{v}}(k)$, $\hat{\mathbf{d}}(k)$, are the estimates of $\mathbf{M}$, $\mathbf{v}$, $\mathbf{d}$, respectively. Defining the adaptation error as

$$\mathbf{e}(k) = \mathbf{x}_{vm}(k) - \mathbf{x}_v(k) , \tag{7}$$

the parameter adaptation algorithm for $\hat{\mathbf{M}}(k)$, $\hat{\mathbf{v}}(k)$, $\hat{\mathbf{d}}(k)$, are given by

$$\hat{\mathbf{M}}(k) = \hat{\mathbf{M}}(k-1) + T\mathbf{K}_m\mathbf{e}(k)\mathbf{u}^T(k-1) \tag{8}$$

$$\hat{\mathbf{v}}(k) = \left[\hat{v}_1(k)\ \hat{v}_2(k)\ \ldots\ \hat{v}_n(k)\right]^T \tag{9}$$

$$\hat{\mathbf{d}}(k) = \hat{\mathbf{d}}_m(k)\mathbf{s}(\mathbf{x}_v(k),\mathbf{u}(k)) \tag{10}$$

where

$$\hat{v}_k(k) = \mathbf{x}_v^T(k)\hat{\mathbf{N}}^{(k)}(k)\mathbf{x}^*(k) \tag{11}$$

$$\hat{\mathbf{N}}^{(k)} = \hat{\mathbf{N}}^{(k)}(k-1)+T\mathbf{K}_{N^{(k)}}\mathbf{e}_k(k)\mathbf{x}_v(k-1)\mathbf{x}_v^T(k-1) \tag{12}$$

$$\hat{\mathbf{d}}_m(k) = \hat{\mathbf{d}}_m(k-1) + T\mathbf{K}_c\mathbf{s}(\mathbf{x}_v(k),\mathbf{u}(k))\mathbf{e}(k) \tag{13}$$

The Coulomb friction function, $\mathbf{s}(\mathbf{x}_v(k),\mathbf{u}(k))$, is given by

$$\mathbf{s}(\mathbf{x}_v(k),\mathbf{u}(k)) = \begin{cases} sign[\dot{x}_{vk}] & \text{if } |\dot{x}_{vk}| > \epsilon_{\dot{x}_{vk}} \\ sign[u_k] & \text{if } |\dot{x}_{vk}| \le \epsilon_{\dot{x}_{vk}} \end{cases} \tag{14}$$

where $sign[\dot{x}_{vk}] = 0$ if $|\dot{x}_{vk}| = 0$, and $\epsilon_{\dot{x}_{vk}}$ is a velocity resolution deadband. $\mathbf{K}_m$, $\mathbf{K}_{N^{(k)}}$, and $\mathbf{K}_c$ are constant positive adaptation gain matrices, and $\mathbf{e}_k(k)$ is the k-th element of the vector $\mathbf{e}(k)$. The block diagram of the model reference adaptive control scheme is shown in Fig. 2.0. Interested readers should refer to Horowitz et. al. [4] for the details of derivation and stability analysis of this algorithm.

3. TMS32010 IMPLEMENTATION

The NSK-UCB robot illustrated in Fig. 1.0 is a SCARA-type arm driven by two NSK direct drive motors. Axis 1 is driven by a model 1410 motor with a maximum torque capability of 245 Nm. Axis 2 is driven by a smaller motor, model 608, capable of delivering up to 39.2 Nm torque. Both motors are powered by switching amplifiers from NSK, Series 1.5 and Series 1.0 for the model 1410 and model 608, respectively. A block diagram of the real-time control system is illustrated in Fig. 3.0.

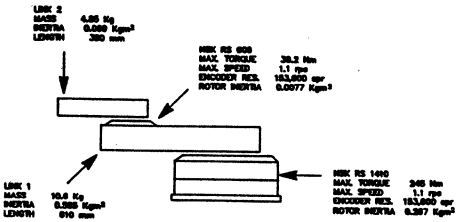


FIG. 1.0 NSK-UCB Two Axis Direct Drive Manipulator

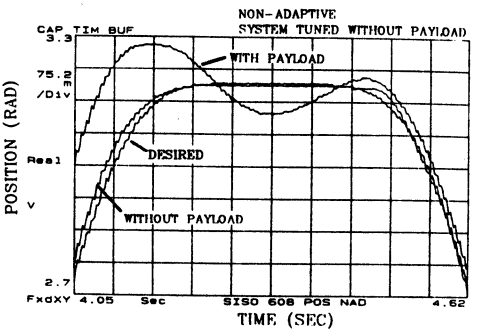


FIG. 4.0 Non-Adaptive Control

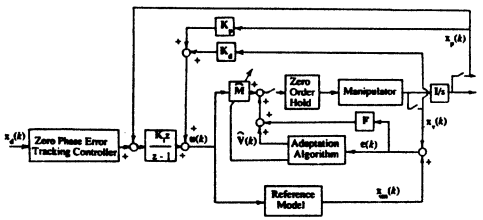


FIG. 2.0 Adaptive Control System

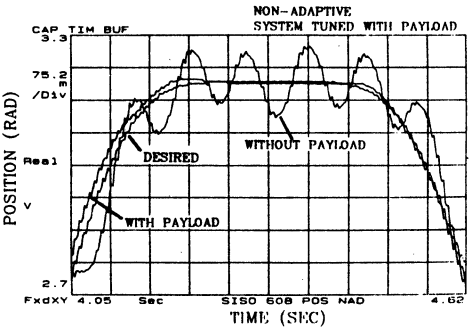


FIG. 5.0 Non-Adaptive Control

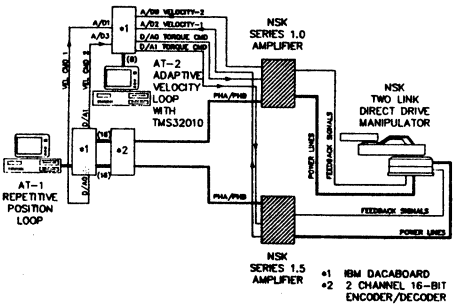


FIG. 3.0 Experimental Set-up

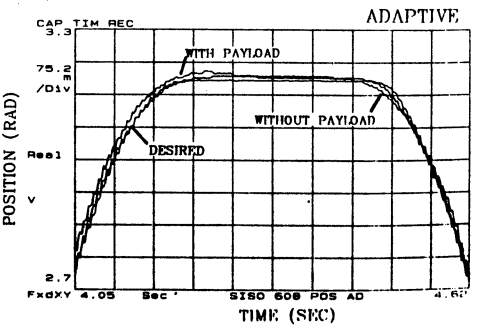


FIG. 6.0 Adaptive Control

Two IBM-AT's are used to implement the algorithm described in the previous section. The first IBM-AT is used to close the proportional position loop for both axes in 7 *ms* . The NSK amplifiers provide a two phase quadrature signal for position feedback. Both motors provide a resolution of 153,600 pulses per revolution. The quadrature signals are decoded to a 16 bit integer which is sampled by the IBM-AT and internally converted to a 32 bit integer by software. The IBM-AT calculates the appropriate velocity command signal for each axis and delivers the command to the second IBM-AT through two digital to analog converters (D/A).

The second IBM-AT which houses the TMS32010 DSP board from Atlanta Signal Processors, Inc., samples the velocity command from the first IBM-AT via two Analog to Digital converters (A/D). The minor adaptive velocity loop for each axis resides on the TMS32010 board. The second IBM-AT serves only as a data acquisition computer for the TMS. The IBM-AT is responsible for sampling four A/D's and controlling two D/A's, during real-time execution. The four A/D's are two for the velocity command, and two for the velocity feedback signal. The velocity feedback signals for both axes are provided by the NSK amplifiers as analog signals ranging from +10 volts to -10 volts, which corresponds to 1.0 RPS to -1.0 RPS. The two D/A's are used to deliver the computed torque commands from the TMS to the NSK amplifiers. For our system the NSK amplifiers have a gain of 47 Nm/V for axis 1, and 25 Nm/V for axis 2. The system is configured such that the TMS is a high speed numeric processor for the IBM. The real-time program is interrupt driven through the system timer of the IBM. The IBM controls the sample, and when data is ready delivers them to the TMS through a common shared memory space between the IBM and the TMS. The IBM in turns signal the TMS to begin execution. Upon completion the TMS delivers the computed torque command to the IBM through the same shared memory space and signals the IBM that the computation for that time slice is complete. The IBM then delivers the torque command to the appropriate NSK amplifiers via the two D/A's.

The adaptive velocity loop for both axis were implemented in TMS assembly. The resulting code was 755 bytes, with a minimum possible loop time of 151 μs . However, the overall algorithm was limited to about 700 μs, due to the limiting speed of the IBM-AT and the IBM Data Acquisition Board used. A similar version of the algorithm was written in assembly for the 80286 on the IBM-AT which ran at a minimum rate of 2 *ms* . Note, that if it was not for the limitation of the I/O drivers, by virtue of pure software execution time, the use of the DSP over conventional general purpose CPU's can decrease the sampling time by almost one order of magnitude.

The entire algorithm was coded with fix point arithmetic in mind. For this particular adaptive control algorithm, a fix point format of 7 bit integer and 8 bit fraction, which corresponds to a numerical range of $\pm 2^{7}.2^{-8}$, was sufficient. The task of scaling was simplified by the TMS, since it provides a 0 to 15 bit barrel shifter which can shift the data as it is being loaded from the memory to the arithmetic logic unit (ALU). Another feature of the TMS which makes it performance superior to most general purpose processors is the parallel hardware multipier which allows the TMS to perform a 16x16 bit multiply in 200 ns. An important feature of the TMS, which is beneficial to most control application and is not available in general purpose processors is the *overflow mode*, which when set prevents numeric overflows and underflows. Another point which should be mentioned is that the macro capability of the TMS assembly language has made the programming task bearable and actually rather simple.

4. EXPERIMENTAL RESULTS

The adaptive control algorithm was implemented on the NSK-UCB robot, and the results are illustrated in Figs. 4.0 to 6.0. The robot was subjected to a payload change for approximately 7.5 Kg. Both axes were tracking a third order trajectory which required each axis to traverse over 180° and return. The plots shown are closeups of the second axis position response as it reaches its 180° destination. Fig. 4.0 illustrates the non-adaptive case in which the system was tuned without the payload. The performance is fair without payload. however, the system experience tremendous overshoot when the payload was added. Fig 5.0 illustrates the opposing case, where the system is tuned with the payload and becomes highly oscillatory when the payload is lost. Fig. 6.0 shows the adaptive case which is nearly indistinguishable for either payload configuration. The results are for a sampling rate of 7 *ms* for both the position and adaptive velocity loops.

5. CONCLUSIONS

The digital implementation of a MRAC to a two axis direct drive robot using the TMS32010 Digital Signal Processor in conjunction with two IBM-AT's was presented. From the actual experience gained through implementation, few features of the TMS were found to be extremely beneficial to controls, these are :

* Macro capability of TMS assembly language
* Small and simple instruction set
* 0 to 15 bit barrel shifter for scaling
* 200 ns 16x16 bit hardware multiplier
* Single cycle instruction for simple timing analysis
* 32 bit Accumulator
* Overflow Mode for automatic numerical wrap-around prevention.

A key to the success of this implementation was the careful scaling and unscaling of intermediate values throughout the calculations. It should be noted that this may no longer be a concern with today's availability of floating point digital signal processors, such as AT&T's DSP32.

ACKNOWLEDGMENT

This work was supported by the National Science Foundation under grant MSM-8511955, the IBM Corporation under the U.C. Berkeley grant for distributed academic computing environment and the NSK corporation.

REFERENCES

[1] Horowitz, R. and Tomizuka, M.(1980). "An Adaptive Control Scheme for Mechanical Manipulators---Compensation of Nonlinearity and Decoupling Control", ASME Paper #80-WA/DSC-6 also in ASME Journal of Dynamics System, Measurement and Control, Vol 108, No. 2, June 1986, pp 127-135.

[2] Tomizuka, M. Horowitz, R. and Anwar, G. (1986) "Adaptive Techniques for Motion Controls of Robotic Manipulators", Japan-U.S.A. Symposium on Flexible Automation, July 1986, pp 117-224.

[3] Sadegh, N. and Horowitz, R.,(1987) "Stability Analysis of an Adaptive Controller for Robotic Manipulators", Proceedings of the 1987 IEEE Int. Conf. on Robotics and Automation, pp. 1223-1229, April 1987.

[4] Horowitz, R., Tsai, M.C., Anwar, G., Tomizuka, M.,(1987) "Model Reference Control of a Two Axis Direct Drive Manipulator Arm", Proceedings of the 1987 IEEE Int. Conf. on Robotics and Automation, pp. 1216-1222, April 1987.

[5] Texas Instruments, "TMS32010 User's Guide", Texas Instruments Incorporated, 1983.

Implementation of a Self-Tuning Controller Using Digital Signal Processor Chips

K. H. Gurubasavaraj

ABSTRACT: This paper describes implementation aspects of a self-tuning motion controller, which uses the Texas Instruments TMS32010 digital signal processor (DSP) chip. The potential advantages in using a DSP chip include reduced operation time, reduced development time, and reduced cost. The self-tuning controller can track variations in system parameters as well as system disturbances. Algorithms are described, experimental results are presented, and implementation strategies to overcome limitations of such systems are discussed.

Introduction

In many applications of electromechanical systems, parameters such as inertia and load torque may vary over time. Variation of load torque, manufacturing variations, and aging can degrade system performance. The design framework of self-tuning control is suitable for adjusting control parameters as well as compensating for disturbances [1], [2]. However, cost considerations have tended to limit the implementation of adaptive control to process control applications, where the control costs can be justified. The advances in microprocessor technology with reduced cost have made it possible to apply adaptive control to electromechanical systems because digital signal processing (DSP) chips reduce cost and development time. In particular, the implementation of adaptive control presented in this paper is currently being considered for use in a commercial product.

Digital control normally is implemented with a microcontroller, and microcontroller architecture is well suited for handling inputs and outputs for motion control systems. However, the arithmetic logic unit of such devices is slow, due to the general-purpose microprocessor architecture. For example, 16-bit multiplications normally require 5–20 μsec. These slow times preclude using these devices for simultaneous identification and control of an electromechanical system. On the other hand, the architecture of a DSP chip is quite suitable for intensive computation. Multiplication times for 16-bit DSP chips are in the range of 60–200 nsec. This time improvement makes possible real-time on-line adaptive control.

This paper discusses the implementation of self-tuning adaptive control using a DSP. Many vendors, such as Texas Instruments, Fujitsu, AT&T, Motorola, National Semiconductor, and NEC, produce a wide range of DSPs. The AT&T DSP32, Texas Instruments TMS32030, and NEC μPD77230 have 32-bit floating-point hardware architecture. They are capable of producing multiply and accumulate floating-point operations within 60–150 nsec. Other DSPs have fixed-point data architecture. The cycle time of these first-generation devices is in the range of 100–200 nsec. As a result of hardware multipliers, multiply and accumulate fixed-point operations are performed in 160 nsec compared to 12–16 μsec using Intel 8086 or 8096 devices. The cost of these devices is comparable to other microcontrollers, costing less than $10. The Texas Instruments TMS32010 device is used here for self-tuning controller development because of cost considerations and availability of development systems. Furthermore, all control functions of a microcontroller are integrated with the TMS32010 central processing unit (CPU) in the new device, referred to as digital signal controller DSC32014. This chip can be considered as a true single-chip controller capable of performing identification, control, and input-output signal processing in real time.

The TMS320 family of processors have Harvard-type architecture with separate data and address lines. The instructions are suited for implementation of digital filters. For example, the combination of LTD and MPY instructions load a coefficient in a register, multiply and accumulate with previous products, and move the data memory to the next higher memory address space. Hence, implementation of each additional pole and zero can be performed with two instructions. More information about these devices can be found in Refs. [3]–[5].

There are implementation constraints with these chips because of the fixed size of random access memory (RAM) space and hardware architecture suited for fixed-point operations. Our objective is to design the adaptive control with the capability to estimate the maximum number of parameters and control the system. Simultaneously, the implemented controller can track the velocity or position of a given electromechanical system. Experiments have been conducted to measure the effect of mismatch between the assumed model and the actual system.

System Model

The dynamics of many electromechanical systems can be represented using well-known models. For illustrative purposes, the permanent magnet DC motor driving a load with total inertia J can be represented by the following equations, where R, L, K_t, K_e, B, v, i, and T_d indicate resistance, inductance, torque constant, back electromotive-force constant, damping coefficient, voltage applied, current through the armature, and disturbance torque, respectively.

$$L\, di(t)/dt + Ri(t) + K_e\, \omega(t) = v(t) \quad (1)$$

$$J\, d\omega(t)/dt + B\omega(t) = K_t i(t) - T_d \quad (2)$$

The operation of the Laplace transform yields the following transfer-function relationships, where K^*, a^*, b^*, and c^* are determined from Eqs. (1) and (2).

$$\omega(s) = [K_1^* V(s) - K_2^*(s + c_1^*) T_d(s)] \div (s^2 + a_1^* s + a_2^*) \quad (3)$$

The equivalent Z-domain transfer function using zero-order hold gives the following relationship among velocity, voltage, and torque disturbance.

$$\omega(z) = [K_1(z + b_1) V(z) - K_2(z + c_1) T_d(z)]/(z^2 + a_1 z + a_2) \quad (4)$$

The adaptive control methodology presented here develops the control parameters as if the system parameters were known. A suitable identification procedure is used to

K. H. Gurubasavaraj is with the Xerox Corporation, Business Products and Systems Group, 1350 Jefferson Road, Rochester, NY 14623.

tune the initial control parameters. The following section describes the identification procedure.

System Identification

The discrete dynamic equations for the system in Eq. (4) can be written in the time domain using the following recursive equation, where the system parameters are represented by θ_i.

$$\omega(k+1) = \theta_1\omega(k) + \theta_2\omega(k-1) + \theta_3 v(k) + \theta_4 v(k-1) + \theta_5 T_d(k) + \theta_6 T_d(k-1) \quad (5)$$

The system equations can be written in vector notation, where the vector Θ represents system parameters, the Φ vector constitutes all known signals, and superscript T denotes transpose of the vector.

$$\omega(k+1) = \Phi^T(k)\Theta(k) \quad (6)$$

If the torque disturbance is constant, then $T_d(k)$ equals $T_d(k-1)$, and the last two terms in Eq. (5) can be combined to give a single bias term. This unknown bias can be included in the system parameters θ_i. A straightforward recursive least-squares (RLS) estimation procedure can be used to identify the system parameter vector Θ. However, when the torque disturbances vary over time and the torque disturbance sequence is not known, the estimation process becomes nonlinear due to unknown torque disturbance terms $T_d(k)$ and $T_d(k-1)$ in the Φ signal vector, which multiply the unknown parameter vector Θ.

For the preceding class of problems, the elements of the parameter vector Θ as well as the unknown elements of the signal vector Φ need to be estimated. The estimation problem can be solved by using either the extended least-squares (ELS) method or the approximate maximum likelihood (AML) estimation method [6]. If the properties of the disturbance noise distribution are known, the AML estimate has superior convergence properties compared to the ELS method. In the absence of such knowledge, both schemes exhibit similar convergence properties. The simplicity of the ELS algorithm and the absence of knowledge about the disturbance prompted us to use ELS estimation. The recursive estimation scheme is given by the following equations, which are similar to Kalman filter equations for linearized estimation, where the superscript on $\hat{\Theta}$ indicates the estimate and the vector B represents the gain.

$$\hat{\Theta}(k+1) = \hat{\Theta}(k) + B(k+1)(\omega(k+1) - \Phi^T(k)\hat{\Theta}(k)) \quad (7)$$

Since not all elements of the Φ vector are known in this equation, the unknown elements $T_d(k)$ and $T_d(k-1)$ are replaced by their residual sequence. The residual sequence of $T_d(k)$ is obtained from the following version of Eq. (5), where the parameters are replaced by estimates obtained from Eq. (7).

$$\hat{T}_d(k-1) = (1/\hat{\Theta}_5)[\omega(k) - \hat{\Theta}_1\omega(k-1) - \hat{\Theta}_2\omega(k-2) - \hat{\Theta}_3 v(k-1) - \hat{\Theta}_4 v(k-2) - \hat{\Theta}_6\hat{T}_d(k-2)] \quad (8)$$

The $T_d(k)$ estimates in the Φ vector of Eq. (7) are replaced by $\hat{T}_d(k-1)$ from Eq. (8). The preceding substitution assumes that the disturbances are continuous in nature and the bandwidth of such disturbances is much lower than the sampling rate. The recursive equations to determine the vector gain B are similar to the Kalman filter equations.

$$B(k+1) = P(k)\Phi(k) \div [1 + \Phi^T(k)P(k)\Phi(k)] \quad (9)$$

$$P(k+1) = [I - B(k+1)\Phi(k)]P(k) \quad (10)$$

Here $P(k)$ is the covariance matrix, which is initialized by setting $P(0)$ equal to a diagonal matrix. Elements of the parameter vector Θ are initialized by some initial guess.

The design of the controller is carried out by using estimates of the system parameters instead of actual values. The single-step-ahead prediction is used to generate control signals. The desired reference velocity during the next sample time is equated to the single-step-ahead velocity prediction by using the following version of Eq. (5):

$$v(k) = (1/\hat{\theta}_3)[\omega_{\text{ref}}(k+1) - \hat{\theta}_1\omega(k) - \hat{\theta}_2\omega(k-1) - \hat{\theta}_4 v(k-1) - \hat{\theta}_5\hat{T}_d(k) - \hat{\theta}_6\hat{T}_d(k-1)] \quad (11)$$

Implementation Considerations

The preceding real-time identification and control laws have been implemented using an Intel 8051 family controller with a Texas Instruments TMS32010 DSP as a coprocessor. The block diagram of the hardware schematic is as shown in Fig. 1. The DSP is used to generate velocity profiles and perform estimation and control calculations to generate the controlled input. The timers and counters on the 8051 are used to perform bookkeeping functions. All interface logic, input and output processing, and the TMS32010 CPU are integrated in the device DSC32014. This device provides the needed input and output processing capabilities as well as the fast computation capabilities of a DSP. Hardware design based on the DSC32014 is in progress.

The implementation of the preceding equations should consider the internal hardware architecture of the device. The architecture of the TMS320 devices is optimized to implement digital filters. For example, the TMS32010 can implement loading the register, adding the value to the accumulator, and moving the signal value into the next memory location. These capabilities are well suited for any classical filter implementation. However, the estimation routines need matrix or vector manipulations. Subroutines can be written for doing these manipulations. The time needed for these calls can be saved if the operations are performed in scalar form. Scalar manipulations decrease the RAM size requirement, while increasing the read-only memory (ROM) requirement due to additional coding. At the present juncture, this trade-off is advantageous due to limited RAM space (144 words) compared to ROM space (1536 words) available on these chips.

Estimation Routine and Control Design

The estimation routine used in Eqs. (7)–(10) can be directly implemented for estimating a small number of parameters. Estimating a larger number of parameters requires larger memory space. The covariance matrix P in Eqs. (9) and (10) should be positive definite for assuring convergence. The matrix P can lose positive definiteness due to subtraction operations in Eq. (10), leading to divergence. To provide numerical stability, the update of the covariance matrix can be accomplished with the square-root version of the P matrix instead of the P matrix itself, which is known as square-root filtering in the literature. However, square-root filtering is computationally expensive. Bierman's UDU^T method [7] provides the advantage of less memory space and does not need square-root calculations, while accomplishing the same objective. Bierman's method requires $n(n-1)/2$ locations for covariance matrix manipulation instead of n^2 locations in a regular filter implementation.

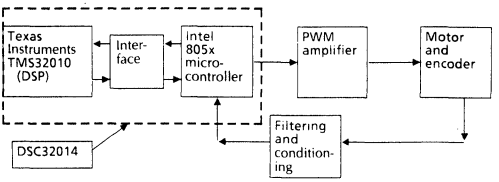


Fig. 1. Block diagram of implementation hardware.

Hence, Bierman's UDU^T method is employed to provide numerical robustness and for its applicability to estimation of other higher-order systems. Details about this algorithm can be found in Ref. [7].

The norm of the covariance matrix (and, hence, the filter gain) decreases as time increases and eventually goes to zero. This is desirable if the system is indeed time invariant. If the parameters are time varying, the decrease causes the loss of adaptive capability. To keep the filter active, the covariance matrix elements are divided by a constant less than 1, which is known as a forgetting factor. At the same time, Eq. (9) is also modified slightly. The TMS320 architecture needs a subroutine to perform divisions. The reciprocal of the forgetting factor is used to multiply the covariance matrix elements. Experiments were conducted to observe the effects of different forgetting factors. In some cases, the mismatch of the forgetting factor can increase the covariance matrix values to cause numerical instabilities or decrease them to small numbers. (Forgetting factors of less than 0.9 lead to increases in the elements of the covariance matrix. Forgetting factors of 0.98 and 0.99 provided better results for our specific applications.) To provide some protection against these situations, bounds on the minimum and maximum norms were employed. Resetting of the covariance matrix was performed at these boundaries, and this strategy worked fairly well in practice.

The convergence of RLS estimation can be assured while tracking only system parameter variations. The convergence is independent of the initial parameter estimates. When significant disturbances are present, the equation error due to the disturbance sequences leads to biased estimates. The consistency of RLS relies on the uncorrelated residual sequence, which requires a special noise structure. A correlated residual sequence leads to biased estimates. The ELS estimation is used to estimate disturbance torques and associated parameters. In this case, convergence is dependent on the initial parameter estimates. For many electromechanical systems, the parameter bounds are known. The estimator converges to the true values when the initial estimates of the parameters are in the proximity of the actual values. The estimates of the product θ_5 and $T_d(k)$ are known to a greater degree of certainty than the individual components.

In actual implementation, the gain term θ_3 can be normalized. For the case presented, the total number of parameters that need to be adapted is five. Overflow and underflow situations may occur due to fixed-point representation of numbers. Appropriate scaling becomes important to avoid these problems. Scaling is a continuous conflict between the dynamic range and resolution of signals or coefficients. Sign-plus two's complement arithmetic is used to represent numbers. Coefficient scaling is accomplished by estimating the maximum value of the coefficient estimates. Then, all coefficients can be normalized within the available word length. Similarly, signals are scaled. Setting of the overflow mode saturates the coefficient value at the maximum. This feature recovers the estimator from soft saturation without leading to damaging consequences. Appropriate safeguards need to be provided for eventual saturation problems. Many different ad hoc strategies can be used, depending on the type of saturation. The ELS algorithm using fixed-point arithmetic requires approximately 200 μsec for computation.

For a set-point regulator problem, the absence of persistent excitation may cause the filter to diverge. For the cases studied, the frequency components of torque disturbances appear to provide the needed frequency components and prevent divergence. Some divergence-related problems are noticed in linear systems without disturbances. Many investigations are being carried out to determine the cause of such divergence. At present, it is ascribed to insufficient excitation of input signals. If these disturbances are absent, then some perturbation may have to be provided in the input signal to prevent filter divergence [8]. Providing the needed excitation for the estimator and good regulatory performance seem to be a challenge.

Experimental Results

An Electrocraft E543 motor is used in the laboratory experiments. Using a magnotrol brake, torque disturbances of varying magnitudes are induced. The amplitude of such torque disturbance variation is limited to 31 oz.-in. Different torque magnitudes are used in the experiments. A sample period of 400 μsec is used.

Identification and Predictive Control for Constant Disturbance Torques

The predictive control using RLS estimation is shown in Fig. 2. The servo is tracking a series of trapezoidal profiles. As can be seen, the adaptation is complete during the ramp-up period. Ten or twelve samples of data are needed for convergence. Similar results confirm adaptation to different sets of system parameters by using different motors, inertias, and frictional loads. However, this estimation scheme leads to biased results when significant variations of torque disturbances are present. Figure 3 shows the performance for triangular torque disturbances at a frequency of 5 Hz. The velocity variation is limited to ± 5 counts/sample. The nonadaptive compensator performance for torque variation is similar to the RLS method.

Performance Under Varying Torque Loads

The performance of the controller based on ELS identification is compared with a well-tuned proportional-integral-derivative

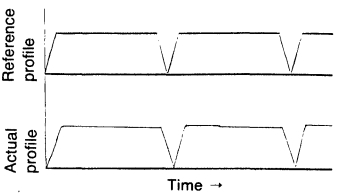


Fig. 2. Velocity profile tracking using RLS identification (under no torque disturbance).

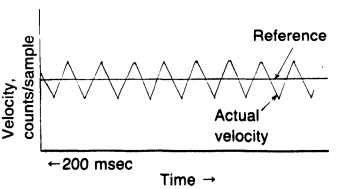


Fig. 3. Response during constant velocity using RLS identification (under 5-Hz torque disturbance).

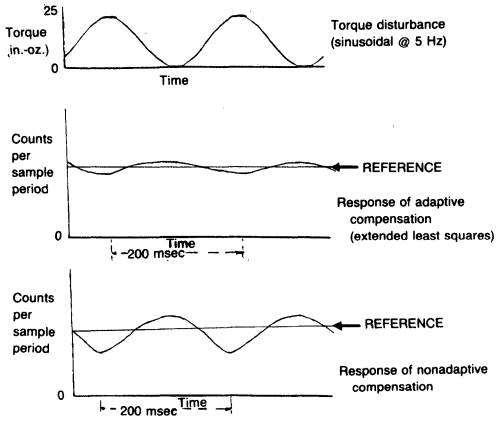


Fig. 4. Comparison of controller response to torque disturbances.

(PID) controller in the presence of sinusoidal torque disturbance with an amplitude of 25 oz.-in. at a frequency of 5 Hz. These low-frequency disturbances within the closed-loop bandwidth are difficult to handle using conventional controllers. The experimental setup remains the same for this identification and control procedure as that used previously. The results are shown in Fig. 4. The PID control errors are limited to ± 4 encoder counts/sample, whereas the adaptive control errors are within ± 1 encoder count/sample. The error bounds are invariant to the variation in the commanded reference level. Hence, the resolution of the encoder becomes a limiting factor in attaining lower steady-state error.

Conclusions

Adaptive control has become a viable alternative for controlling electromechanical systems. The computation power of the digital signal processor can be conveniently exploited to provide performance significantly higher than possible with conventional microprocessors.

Performance, Adaptability, and Reliability

These controllers update their information regularly. This increase in the knowledge base contributes to the adaptability of the system. The real-time performance improves significantly if the system has significant disturbances compared to any other conventional scheme. The major strength of adaptive control is its superior performance under system parameter variations and disturbances. The adaptive capability provides the desired performance throughout the life of the mechanism. Reliability is enhanced since the system meets the expected performance in spite of aging of the mechanisms.

Cost Implications

The monotonically decreasing cost of electronics offers the capability of modifying the dynamics cost-effectively in electronic hardware and software. One of the benefits may be the relaxation of tolerance specifications of mechanical components. After a certain point, precise tolerances increase the component cost exponentially. The optimal trade-off point for a given system has to be explored in greater detail. Since the controllers are self-tuning in nature, they will not need tuning in the field. This reduces the service calls and improves reliability. In addition, the state of the system can be estimated. These estimated states can be used as diagnostics for replacement of the components prior to actual mechanism failure. This aspect can be used to plan and schedule maintenance activities.

Reduction in Development-Cycle Time

The design basically encompasses the following aspects. The identification of the system, controller design, and implementation are performed in real time. The identification, compensator design, and transfer of the designed parameters into a workable hardware or software are eliminated in the design cycle. The faster computation capability of signal processors can be used advantageously to control higher-order dynamic systems. This feature allows sensor placement at the load end and includes all dynamics within the loop. The preceding adaptive controllers are used successfully as fixture development controllers at Xerox during development, when the system parameters are partially known and/or vary over the development period.

Limitations and Future Research

Adaptive controllers are more complex than traditional feedback control systems, and they are nonlinear in nature. There are some recent results to prove overall stability of such systems under restrictive conditions. Additional research is needed to prove overall convergence and stability of such systems under relaxed conditions. Software integrity becomes an important issue for the employment of these controllers. The overflow and underflow conditions make the scaling problem more difficult. The convergence of many recursive estimation schemes depends on persistently exciting input signals. However, during regulation periods, the estimation process diverges due to the absence of persistent excitation. Dithering of the input signals or pausing the estimation process during regulation provides a partial solution at the cost of reduced performance or adaptability. More research in this direction is needed.

Acknowledgments

The author acknowledges the invaluable support of Vincent Williams in all aspects of this project. The author also acknowledges anonymous reviewers for their constructive comments.

References

[1] K. J. Astrom and B. Wittenmark, "On Self-Tuning Regulators," *Automatica*, vol. 9, pp. 195–199, 1973.

[2] K. J. Astrom and B. Wittenmark, "Self-Tuning Controllers Based on Pole-Zero Placement," *IEE Proc.*, vol. 127, pp. 120–130, 1980.

[3] L. R. Morris, "Digital Signal Processing Microprocessors: Forward to the Past?," *IEEE Micro*, pp. 6–8, Dec. 1986.

[4] K. S. Lin, G. A. Frantz, and R. Simar, "The TMS320 Family of Digital Signal Processors," *Proc. IEEE*, vol. 75, no. 9, pp. 1143–1159, 1987.

[5] "TMS32010 Users Guide," Texas Instruments, 1983.

[6] G. S. Goodwin and K. S. Sin, *Adaptive Filtering Prediction and Control*, Prentice-Hall, 1984.

[7] G. J. Bierman, *Factorization Methods for Sequential Estimation*, New York: Academic Press.

[8] B. D. O. Anderson, "Exponential Convergence and Persistent Excitation," 21st Conf. Decision and Contr., 1982.

[9] W. L. Brogan, "Kalman Filtering in Identification and Control Problems," unpublished short course notes presented at the University of California, Los Angeles, 1984.

[10] K. H. Gurubasavaraj, "Self-Tuning Motion Controller for Varying System Parameters," Amer. Contr. Conf., 1988.

K. H. Gurubasavaraj received the B.S. degree in electrical engineering from Mysore University and the M.S. degree in electrical engineering from Bangalore University, Bangalore, India, in 1971 and 1973, respectively. He received the Ph.D. degree in electrical engineering from the University of Nebraska, Lincoln, in 1983. From 1973 to 1980, he was with Karnataka Electricity Board and Bangalore University. From 1983 to 1985, he was a faculty member at Union College, Schenectady, New York. During that period, he consulted with the General Electric R&D Center. Since 1985, he has been working as a Technical Specialist/Project Manager of Control Technology Development for the Xerox Corporation. He is also an adjunct faculty member at Rochester Institute of Technology. His current interests are in the implementation of adaptive control, Kalman filtering, and robust control concepts to improve performance and reliability of commercial products cost-effectively.

Motion Controller Employs DSP Technology

Robbert van der Kruk and John Scannell
Philips Centre for Manufacturing Technology

Several control strategies are considered to improve the performance of a digital motion controller, including: feedback design, velocity and disturbance observers, trajectory generator and feedforward compensation.

Development of a digital motion controller must carefully consider the sampled data nature of the system. For example, a stable position servo system must provide electronic damping, which often means tachometer feedback or else a simple derivative action in the position feedback loop. Using a tachometer increases the cost of the servo system, whereas a simple digital differentiation technique amplifies the quantization noise on the digital position signal. This causes excessive current ripple in the motor together with unpleasant audible noise. This new design uses a velocity observer to drastically reduce the quantization problem associated with simple digital velocity estimators.

Elimination of steady state errors has long been performed using an integrator in the error path. However, this technique has several disadvantages (e.g. wind-up, tuning) and tends to reduce the stability margins. This new design employs a more advanced technique, a disturbance observer, to eliminate steady state errors.

Many servo systems exhibit low frequency mechanical resonances due to the finite stiffness of the coupling between the motor shaft and the load. We will show that set point functions with programmable jerk dramatically improve the performance of such systems.

Modern automation applications place ever increasing demands on the tracking accuracy of servo controllers. Velocity and acceleration feedforward techniques can be employed to minimize tracking errors. The new design incorporates feedforward techniques.

From Analog to Digital Control

Introduction of the microprocessor, and more recently the signal processor, have radically altered the field of high performance servo control over the past decade. The advent of digital techniques has presented the designer with tremendous flexibility in the control algorithm design[1,2,3,4]. In addition, the provision of extensive diagnostics and status information has become a relatively simple operation thus easing the tasks of system development and support. However, this migration from analog to digital has several problems associated with it. In particular, the design of the control algorithm must take account of the sampled data nature of the system. Problems due to the delays introduced by the sample period and the calculation time must be carefully considered in the design of the feedback parameters. The quantization noise due to the digital nature of the position information must also be carefully analyzed and its effects minimized.

Consider the simple block diagram of the digital servo system in *Figure 1*. Assume that the power amplifier has a large bandwidth compared with the servo loop and may therefore be modeled as a gain element. The motor model is a double integrator and neglects friction and mechanical resonances.

The open loop transfer function of the continuous elements, including the sample and hold effect and the calculation delay is:

$$H(s) = \frac{X_{enc}(s)}{U(s)} = K\frac{e^{-sT_c}\left(1-e^{-sT}\right)}{s^3}, \quad (1)$$

s = Laplace operator

where

$K = K_{dac} K_a K_m K_{enc}/J$

K_{dac} = gain of the D/A converter

K_a = gain of the amplifier

K_m = motor constant

K_{enc} = resolution of the position encoder

J = motor inertia.

The most convenient method of analyzing this sampled data system is to convert H(s) into its discrete time equivalent, H(z), where z is the discrete time operator[5,6]. However, the z plane analysis only provides information at the sample instants, i.e., fractional delays are not allowed. Thus, in order to examine the effect of the calculation delay, T_c, the two extreme cases are considered: no calculation delay, $T_c = 0$, and maximum calculation delay, $T_c = T_s$.

Calculation of the feedback parameters is first considered for zero calculation delay. For $T_c = 0$, Equation (1) becomes:

$$H(z)\,|_{T_c=0} = \frac{X_{enc}(z)}{U(z)} = Z\{L^{-1}[H(s)]\} = \frac{1}{2}KT_s^2\frac{(z+1)}{(z-1)^2}. \quad (2)$$

A suitable value of the velocity feedback gain, K_v, is calculated by considering the loop transfer function. From *Figure 1*, the motor velocity is approximated by using the pulse count technique. The open loop transfer function of the velocity loop is:

$$V(z)\,|_{T_c=0} = \frac{z-1}{z}H(z)\,|_{T_c=0} = \frac{1}{2}KT_s^2\frac{(z+1)}{z(z-1)}. \quad (3)$$

Using the root locus technique suitable value of K_v may be derived[7]. A robust selection, giving sufficient design freedom for the outer position loop, is $K_v = 0.343/KT_s^2$. This gives a damping ratio of 1.0. Using this value of K_v the open loop transfer function of the position loop is:

$$P(z)\,|_{T_c=0} = \frac{1}{2}KT_s^2\frac{z(z+1)}{(z-1)(z-0.414)^2}. \quad (4)$$

As before, the root locus technique is used to calculate a value for the position feedback gain, K_p. Selecting a damping ratio of 0.7 gives $K_p = 0.072/KT_s^2$.

In a similar manner, the loop parameters may be determined when a calculation delay of one sample period is assumed. In this case Equation (2) becomes:

$$H(z)\,|_{T_c=T_s} = \frac{1}{2}KT_s^2\frac{(z+1)}{z(z-1)^2}. \quad (5)$$

The corresponding values of K_v and K_p, for the same damping ratios, are $K_v = 0.180/KT_s^2$.

The results obtained are summarized in *Table 1*, where F_s is the sample frequency

Reprinted, with permission, from *PCIM*, Sept. 1988.

($F_s = 1/T_s$). Generalizing these results for calculation delays between 0 and T_s gives:

$$K_v = \frac{0.35}{KT_s(T_s+T_c)}, \quad K_p = \frac{0.072}{K(T_s+T_c)^2}. \tag{6}$$

	$K_v.KT_s^2$	$K_p.KT_s^2$	Total Delay	Bandwidth
$T_c = 0$	0.343	0.072	T_s	$F_s/17.5$
$T_c = T_s$	0.180	0.018	$2T_s$	$F_s/35.0$

$$\text{Bandwidth} = \frac{1}{17.5(T_s+T_c)}. \tag{7}$$

These generalized results have also been verified for fractional calculation delays, i.e. $0<T_c<T_s$. From the results it can be concluded that the sample frequency must be at least 17.5 times higher than the required bandwidth of the position loop.

Velocity Observer

The straightforward pulse count method of velocity estimation results in poor resolution at low speeds. The quantization error in the velocity signal becomes worse with increasing sample frequency and can cause excessive current ripple in the motor together with audible noise (*Figure 3a*). This problem can be reduced by using a position encoder with a greater resolution, but this is a rather expensive solution. In addition, increasing encoder resolution unnecessarily increases the data speed, which can lead to a decrease in the maximum servo velocity. A velocity observer, estimating the servo velocity with a higher resolution than the pulse count method, can be used to overcome this quantization problem[7,8]. The discrete time transfer function from the servo command, U(z), to the position output, $X_{enc}(z)$, is given by Equation (5), ($T_c = T_s$). The transfer function from the servo command, U(z), to the velocity output, V(z), is:

$$G(z) = \frac{V(z)}{U(z)} = Z\{L^{-1}[sH(s)]\} = KT_s^2\frac{1}{z(z-1)}, \quad T_c = T_s. \tag{8}$$

(continued)

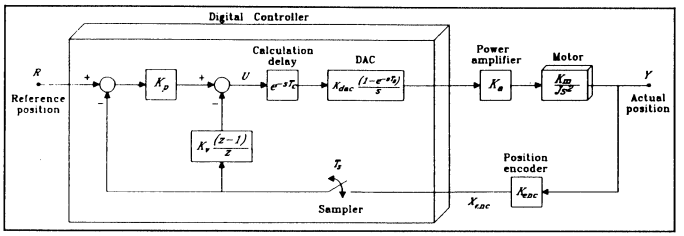


Figure 1. *Simple Second-Order Digital Controller.*

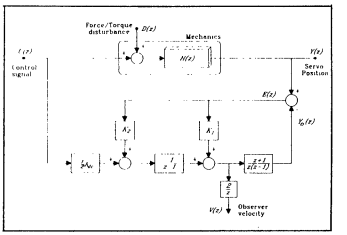


Figure 2. *Velocity Observer Block Diagram.*

DSP Controller

Equations (5) and (8) yield the observer structure shown in *Figure 2*, where $K_{dc} = KT_s^2$. Two feedback terms are used to correct for deviations between the observer and the system. The choice of K_1 and K_2 involves a trade-off between the bandwidth of the observer correction loop and the quantization of the estimated velocity, V(z). Since the objective of the velocity observer is to reduce the velocity quantization level, the choice of K_1 and K_2 are determined using this criterion. The value of K_1 must be less than ½ in order to provide a better resolution than the pulse count method. Practical values are $K_1 = 0.04$ and $K_2 = K_1^2$, resulting in a resolution enhancement factor of $½K_1 = 12.5$ and an observer -3dB bandwidth of $F_s/105$.

For very stiff servo systems, where the second order model is valid, the relation between the measured position, Y(z), and the observer estimated velocity, V(z) is:

$$V(z) = 2\frac{(z-1)}{(z+1)}Y(z). \tag{9}$$

This shows that the observer is essentially an implementation of Tustin's rule[9]. This rule is an approximation of the differential operator and is without additional phase shift. However, it cannot be programmed directly due to stability problems caused by the pole at z = -1.

Using the velocity approximation of Equation (9), the feedback analysis may be repeated for a servo system with the velocity observer. The corresponding general results are:

$$K_v = \frac{0.5}{KT_s(T_s+T_c)}, \quad K_p = \frac{0.125}{K(T_s+T_c)^2} \tag{10}$$

and

$$\text{Bandwidth} = \frac{1}{13.8(T_s+T_c)}. \tag{11}$$

From these results it can be seen that the sample frequency of a system using the velocity observer must be at least 13.8 times higher than the desired position loop bandwidth. Thus, for the same sample time and calculation delay, a system using the velocity observer has a 27% higher bandwidth than the same system using the pulse count method. The velocity observer has one machine-dependent parameter, K_{dc}, which can be simply tuned by monitoring the observer error.

The performance improvement yielded by the velocity observer is demonstrated in *Figure 3*. A reference velocity signal of 2.5 position increments per sample period ($2.5inc/T_s$) is applied to a brushless linear motor used in the Philips chip mounting machines. *Figure 3a* shows the current in the motor when the pulse count method of velocity estimation is used. In contrast, when the velocity is estimated using the observer the high frequency current components are eliminated as shown in *Figure 3b*. This reduction in the current ripple results in less motor heating, less dissipation in the power amplifier and a significant reduction in the audible noise level.

Disturbance Observer

The observer error signal, E(z), in *Figure 2* represents the reconstruction error between the observer and the servo system. The input signal, D(z), represents an external force or torque disturbance. If we assume that the observer model is exact

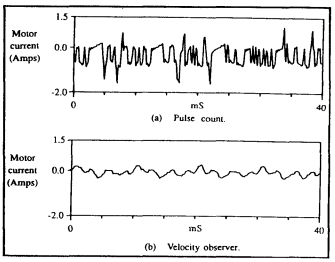


Figure 3. *Comparison of Motor Current for Brushless Linear Motor Running at 2.5 incs/Ts.*

then the expression for the error is:

$$E(z) = \frac{1}{2} K_{dc} \qquad (12)$$

$$\frac{(z+1)}{z(z-1)^2 + K_1(z-1)(z+1) + K_2(z+1)} D(z).$$

This expression for E(z) is the disturbance, D(z), low pass filtered by the observer poles. Thus, the observer error signal provides an estimate of the disturbance. This may be used to compensate the system and correct for errors caused by the disturbance. When applying the observer error signal to the system (*Figure 4*), two conditions must be met:

1. In steady state, the magnitude of the compensation signal, C(z), must equal the magnitude of disturbance:

$$\lim_{z \to 1} \frac{C(z)}{D(z)} = 1 ; \qquad (13)$$

2. The closed poles of the observer must remain unchanged.

Fulfilling these two conditions gives the disturbance observer block diagram shown in *Figure 4* [8,10].

In steady state we have:

$$\lim_{z \to 1} C(z) = \frac{2K_2}{K_{dc}} \lim_{z \to 1} E(z) = \bar{d} \qquad (14)$$

where $\bar{d}$ is the steady state value of the disturbance D(z). For a stable, well damped observer the equation $K_2 = K_1^2$ is valid where $K_1 \leq 0.17$. It can be shown that the bandwidth of the disturbance observer is directly proportional to the value of K_1 while the quantization noise fed to the motor is proportional to K_2, i.e. K_1^2. Thus, the choice of K_1 is a trade-off between the bandwidth of the disturbance compensation and the quantization noise. Note, however, that the value of K_1 is independent of the velocity and position loop gains and therefore the tuning of the disturbance observer may be performed independently of the outer control loops.

Due to the similarity between the velocity and disturbance observer structures it is possible to use a single observer for both velocity estimation and disturbance compensation. However, for the tuning of the velocity observer the reduction of quantization noise is generally more important than the bandwidth of the feedback loop. For the disturbance observer the bandwidth of the disturbance rejection is the more important tuning criterion. Therefore, for optimal performance, two separate observers are necessary.

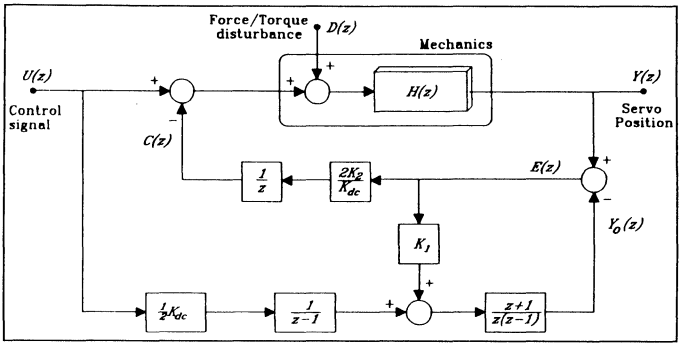


Figure 4. *Disturbance Observer Block Diagram.*

Figure 5 shows the response of a linear servo motor when a step disturbance of 20 Newton is applied. The bandwidth of the position loop is 44Hz with a damping ratio of 0.8. Using integral compensation the maximum position error is 92μm (1μm = 1 inc) and the steady state condition, with zero position error, is recovered at 90msec. Using the disturbance observer the maximum position error is only 19μm and the recovery time is 36msec. The performance improvement yielded by the disturbance observer stems from its relatively high bandwidth, 70Hz. By comparison, the bandwidth of the integrator is limited to 8Hz in order to maintain the position loop stability margins.

In the design of the disturbance observer the effect of friction is neglected. This design procedure has been validated by tests, in which the effect of friction forces has been found to have a negligible effect on the disturbance observer performance.

Trajectory Generator

Response of a servo system to a step input can be used as a measure of the system performance. However, in practical applications position steps are rarely used as set points as they can cause controller saturation resulting in nonlinear servo behavior. This results in large tracking errors, significant overshoot and poor settling times. Higher order set point functions may be used to overcome these saturation problems. More common position set points are the ramp (first order), the parabolic profile (second order) and the cubic profile (third order). Here, attention is paid to the parabolic and the cubic set points[7].

A parabolic position set point results in a triangular velocity profile with discontinuous acceleration. If this set point is applied to a servo system with a low

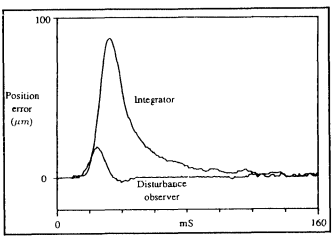


Figure 5. *Comparison of Disturbance Rejection for Step Disturbance of 20N Applied to a Brushless Linear Motor.*

frequency mechanical resonance then the discontinuous acceleration (and hence discontinuous torque) excites the resonant frequency resulting in undesirable overshoot and tracking errors. This is shown in *Figure 6a* where a triangular velocity profile is applied to a servo system with a 5.4Hz mechanical resonance frequency. This measurement is performed using a stiff servo motor connected to a mechanical load via a flexible coupling. A position encoder on the motor shaft provides feedback signals for the digital controller. A second encoder after the flexible coupling measures the load velocity responses shown in *Figures 6a* and *b*.

In contrast, a third order profile has no torque discontinuities and hence does not tend to excite the system resonance as

demonstrated in *Figure 6b*. It can also be shown that the position error at the moment that the set point trajectory is finished (the lag error) is smaller for the cubic profile than for the parabolic profile. However, for the same displacement and time, the cubic profile requires a 50% higher maximum acceleration and hence a larger maximum current.

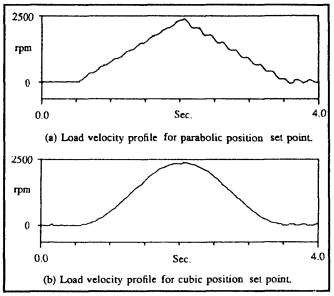


Figure 6. *Velocity Profiles of a Mechanical Load Connected to a Servomotor Via a Flexible Coupling (Mechanical Resonance Frequency = 5.4Hz)*

Generation of cubic set point profiles may be performed by a trajectory generator. This unit is programmed with the desired displacement, velocity, acceleration and jerk. The necessary position set point profile is then calculated by a series of numerical integration operations as shown in *Figure 7*.

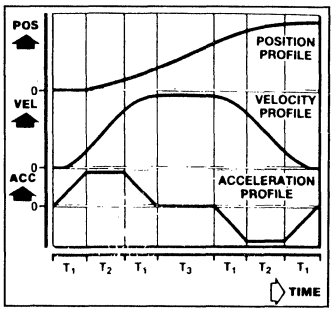


Figure 7. *Generation of Cubic Position Profile.*

Feedforward

Use of feedback as a control stategy yields a servo system with good disturbance rejection (stiff system). However, the servo response to set point commands is not optimum. Set point feedforward can be used to reduce the tracking error[7]. *Figure 8* shows a block diagram of a servo mechanism controlled by velocity and position feedback loops. H_{ff} is the feedforward transfer function. If no saturation or external disturbances occur, and the system is linear and time-invariant, then H_{ff} is calculated by requiring $E(z) = 0$.

$$\Rightarrow E(z) = R(z) - H(z)[K_p E(z) - K_v G(z) Y(z) + H_{ff}(z) R(z)] = 0 \quad (15)$$

Solution of this equation (ignoring the trivial solution of $R(z) = 0$) gives:

$$H_{ff}(z) = H^{-1}(z) + K_v G(z). \quad (16)$$

Thus, the feedforward transfer function consists of the inverse transfer function of the servomechanism and the compensation of the velocity feedback loop. For

$$G(z) = \frac{2(z-1)}{(z+1)}$$

(Tustin's method) and $H(z) =$

$$\frac{1}{2} K T_s^2 \frac{z+1}{(z-1)^2} \text{ (stiff servo)} \quad (17)$$

then

$$H_{ff}(z) = \frac{2K_{fa}(z-1)^2}{z+1} + \frac{2K_{fv}(z-1)}{z+1} \quad (18)$$

where

$K_{fa} = 1/KT_s^2$ and $K_{fv} = K_v$

are the feedforward acceleration and velocity gains, respectively.

However, Equation (18) is non-causal and cannot be programmed directly. For servo systems using trajectory generators, the reference position can be calculated by a numerical integration technique. This is illustrated in *Figure 9* where the reference position is generated from a jerk profile. By using the intermediate results of the reference acceleration and reference velocity the feedforward can be implemented.

Figure 10 shows the effect of acceleration feedforward on the following error during a set point movement. The velocity feedforward parameter is optimized for both traces giving an average following error of zero when the servo is moving with constant velocity. The effect of the acceleration feedforward term is clearly seen in the reduction of the following error during acceleration and deceleration.

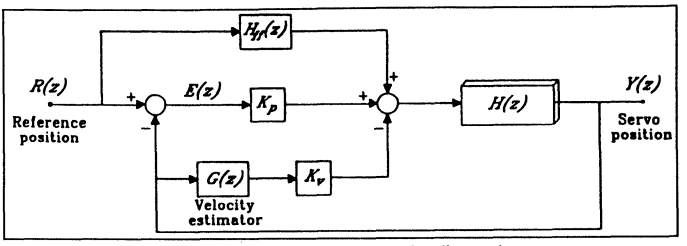


Figure 8. *Discrete Time Control Using Feedback and Feedforward.*

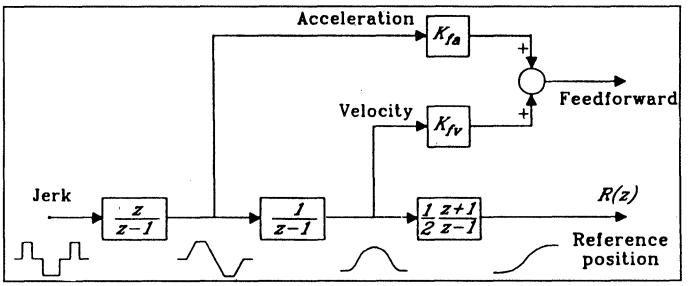


Figure 9. *Trajectory Generator and Feedforward.*

The MCV60

The control strategies described in the preceding sections are implemented on a motion control card, the MCV60. This card communicates with a host computer via the VMEbus. Software on the host provides a user-friendly environment for commissioning of the system and adjusting of the parameters during operation. This card is designed to control two brush-type DC motors or one brushless DC motor. Up to sixteen cards may be used together on the same bus. A functional block diagram of the MCV60 is in *Figure 11*.

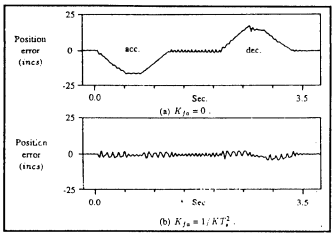


Figure 10. *Position Error During Cubic Set Point Movement [DC motor: velocity = 29.3 rev/sec, acceleration = 39 rev/sec^2, jerk = 78 rev/sec^3].*

The MCV60 hardware is based around the TMS320C25 signal processor running at 40MHz. This high clock frequency, together with the arithmetic capabilities of the signal processor, yield a high sample frequency and a short calculation delay (T_s = 150μsec, T_c = 40μsec). The maximum encoder data speed is 7MHz. On-board memory includes 16k words of program memory and 8k words of data memory of which 2k words is in dual-ported RAM. This provides high speed, bidirectional communication with the VME host computer. The data RAM has a battery back-up for the retention of system parameters when the card is not powered up. Three different position encoders are supported. An incremental encoder interface is standard on the card while piggyback interfaces for resolvers and sine-wave encoders may be simply mounted. Two 16-bit DACs deliver drive signals to the current amplifiers while a second pair of DACs provide monitor information. Hardware synchronization between several cards is also provided with the facility for master-slave control. In addition, five optically isolated outputs and four optically isolated inputs are provided for interfacing with programmable logic controllers.

The motion control software may be divided into card and host levels. At the card level, the controller algorithm implements the various strategies already described. In addition, continuous path movements can be implemented using cubic spline interpolation techniques. Software for auto-homing is provided and, for brushless motors, an automatic magnetic alignment routine together with commutation software is available[11]. Extensive hardware and servo diagnostics are performed at power-up and critical hardware checks are performed each sample period. System monitoring is performed each sample period generating the two monitor DAC variables. Other performance indicators are also recorded such as the maximum error during a movement, the overshoot, the settling time, etc.

At the host level, drivers are provided for communicating with the MCV60. A menu-driven, user-friendly test environment initializes and tunes system parameters. Self-tuning facilities initialize the controller parameters to suitable values based on the system characteristics. More refined parameter tuning may then be simply carried out by a series of well-defined tests. Communication with the card occurs each time a parameter is changed. This facilitates on-the-fly variation of system parameters for use in systems employing gain scheduling techniques.

Figures 12 and *13* demonstrate the speed and accuracy of the MCV60 motion controller. In *Figure 12* the response of a brushless linear motor to a set point displacement of 10mm (10,000 increments) is shown. *Figure 13* shows the motor velocity profile and the position error of the same servo when running at maximum velocity (data speed = 1.5MHz). The maximum position error is only 12 increments (12μm).

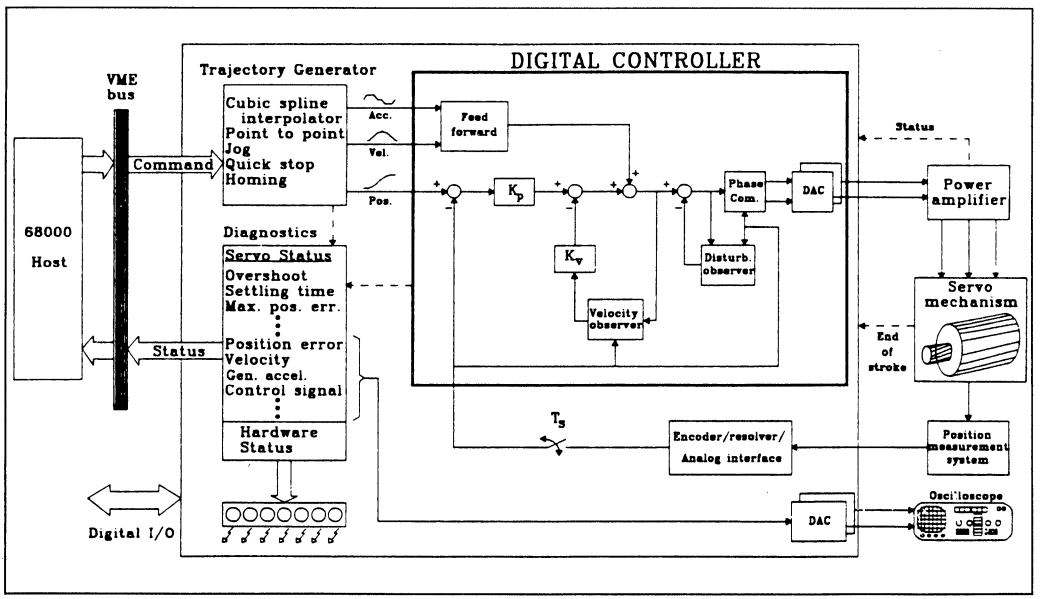


Figure 11. *Block Diagram of the MCV60 Connected to a Three-Phase Brushless Motor.*

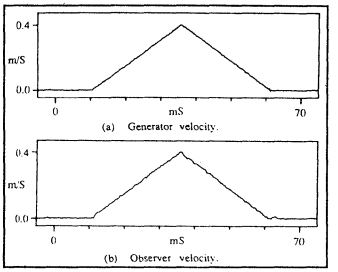


Figure 12. *Profiling Accuracy for a Displacement of 10mm in 50ms.*

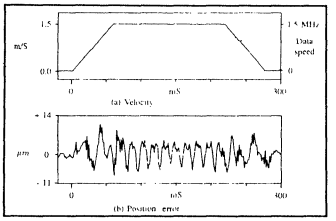


Figure 13. *High Speed Following Accuracy - Data Speed = 1.5MHz.*

Note: Patents are pending for the work described in this article.

References

1. Fritz, D., "High Precision Dynamic Position-Controlled Servo-Drive System," Proc. MOTOR-CON, April 1987, Hannover, pp. 344-357.
2. Squires, D., Kmetz, J., Cox, B., "A High Level DC Servo Controller Implemented as a New Generation of ASIC," Proc. MOTOR-CON, April 1987, Hannover, pp. 175-182.
3. Tal, Jacob, "Motion Control by Microprocessors," Mountain View, CA., Galil Motion Control, 1984.
4. Dimitri S. Dimitri, "Trends in Motion Control Monolithic Versus Modular Designs Continuous Path, Linear and Circular Interpolation," Proc. MOTOR-CON, April 1987, Hannover, pp. 238-247.
5. Åström, K.J. and B. Wittenmark, "Computer Controlled Systems: Theory and Design," Prentice-Hall International Edition, 1984.
6. Franklin, G.F. and J.D. Powell, "Digital Control of Dynamic Systems," Addison-Wesley, 1981.
7. Kruk, R.J. van der, "Motion Control System Concepts," Philips CFT Report, 32/87 EN.
8. Kruk, R.J. van der, "Digital Control of Position Servo Systems," Journal A, 88/1.
9. Tustin, A., "A Method of Analyzing the Behaviour of Linear Systems in Terms of Time Series," JIEE (London), 94, pt. IIa, pp. 130-142, 1947.
10. Scannell, J.R.M., "A Disturbance Observer for the CFT Motion Control System," Philips CFT Technical Note 01/88 EN.
11. The Netherlands Patent Number 8701438, "Drive Arrangement and Motor Energizing System for Use in the Arrangement." □

Using DSPs in AC Induction Motor Drives

DR. S. MESHKAT, Motion Research Inc., Plymouth, Minn., and I. AHMED, Texas Instruments, Houston, Tex.

Although simple to manufacture, ac induction motors require very complex control techniques if they are to be used in servo applications. High performance DSPs can solve the ac motor control problem.

Dc drives still account for a large portion of drives used in industrial control, even though they are less reliable and more expensive than their ac counterparts. This is mainly due to the fact that dc drives have fairly simple control structures and allow precise control.

Ac drives, on the other hand, are less expensive and more reliable, especially in harsh industrial environments. However, ac drives require very complex control techniques and this has prevented them from replacing a large number of dc drives in robotics and motion control.

Ordinary microcontrollers lack the computing performance necessary to carry out these complex control schemes. Faster devices like 32-bit microprocessors and bit-slice processors are too expensive.

But, due to unprecedented performance offered by digital signal processors, this may be changing. DSPs provide more speed than the fastest 32-bit microprocessors, and they do this at a fraction of their cost. They make it possible to implement the complex structures needed to make ac drives the workhorse of industrial control applications.

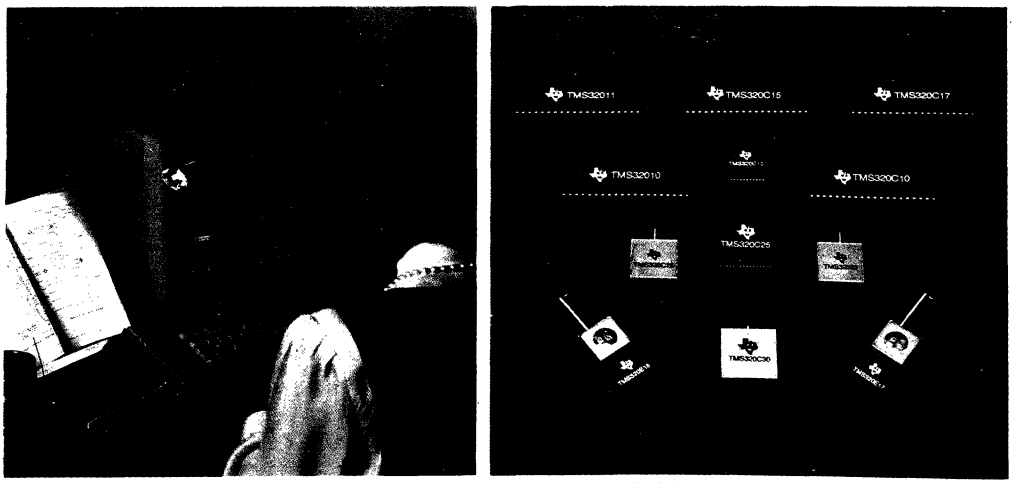
Dynamics of an ac motor

In a conventional field-wound dc motor, there are two independently controllable currents: the field current and the armature current.

In an ac motor, however, there are three phase currents, and the three are tightly coupled together. This means that none of them is independently controllable.

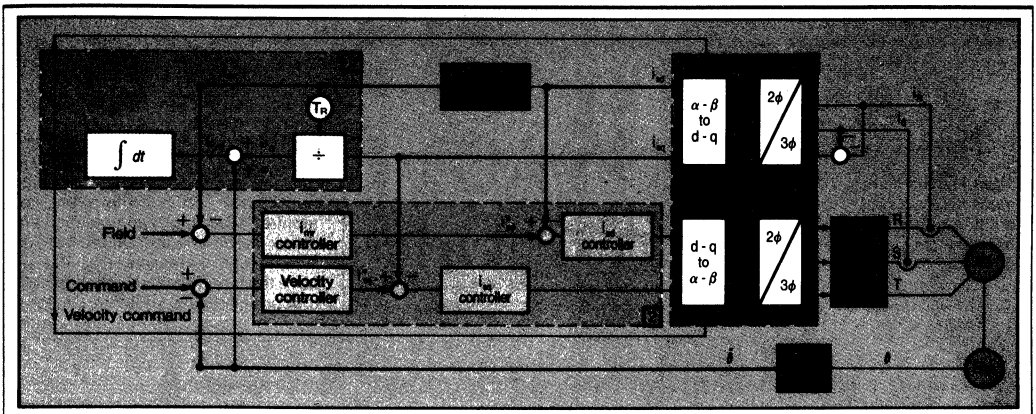
The three currents are represented by the stator current vector $\mathbf{i}_s$. Unlike the case of the dc motor, there is no linear relationship between $\mathbf{i}_s$ and either the torque or the flux.

The torque vs. current relationship,



Personal computer based software tools allow low-cost development of DSP software. At right is shown Texas Instruments' entire family of DSP chips. The back two rows are first generation, the second row is second generation, and the third generation—released now in sample quantities—is in the foreground. Also in the foreground are two DSP packages that contain 4K of EPROM, used in software development.

Reprinted, with permission, from *Control Engineering*, Feb. 1988.



All functional components of the control system for a field oriented ac induction drive can be implemented in software on a single-chip DSP.

in a conventionally controlled ac induction motor, is nonlinear and can be represented as equation (1):

$$T = \frac{3P}{4}\frac{L_m^2}{L_r}|i_s|^2 \frac{\omega_s T_r}{1 + (\omega_s T_r)^2}$$

Where L_m is the stator/rotor mutual inductance; P is the number of poles; L_r is the rotor induction (referred to the stator); T_r is L_r/R_r (R_r is the rotor resistance, referred to the stator); i_s is magnitude of stator current vector; and ω_s is the slip frequency. The slip frequency is the amount the rotor lags behind the synchronous speed ω_m i.e., $\omega_m = \omega_s + \omega_r$.

Equation (1) shows a nonlinear relation between torque and slip frequency ω_s. This simply means that no servo loop can be closed around a traditionally driven ac induction motor. This is the reason why ac induction motors have been used primarily in constant speed applications.

Over the past decade, several different control methods for "squirrel cage" induction motors have been proposed. The most popular of the alternatives is the field orientation scheme. In this control scheme, the magnetic flux is measured (or calculated) and fed back to the control unit as a basis for the commutation of the stator current vector. This method is called "vector control."

The main objective of vector control is to decouple the torque generating component of the stator current from the field producing one. In our notation, we will use i_{qs} and i_{ds} to represent the torque generating component and the flux generating component, respectively, of the stator current.

The result of separating the torque and field components forces the machine to resemble a field-wound dc motor. Once this is accomplished, the rest of the control algorithm remains the same as an ordinary dc motor.

A field oriented ac servo

To explain how field oriented control works, we simplify the description and analysis of a three-phase motor to two phases. This way one can visualize more easily the relation between the components of the stator vector in a vector diagram.

Let the values of the stator current vector $\mathbf{i}_s$ be represented as components i_r^*, i_s^*, and i_t^* of stator windings r, s, and t. Then to transform them to a two phase system that has components i_α and i_β, we use the following transformation matrix (equation 2):

$$\begin{vmatrix} i_\alpha \\ i_\beta \end{vmatrix} = \begin{vmatrix} 1 & 0 & 0 \\ \sqrt{3}/3 & \sqrt{3}/2 & 0 \end{vmatrix} \begin{vmatrix} i_r^* \\ i_s^* \\ i_t^* \end{vmatrix}$$

In the two phase system, i_α and i_β represent the stator current in the stationary reference frame.

But the reference frame we are interested in, for the reason of controlling field and torque independently of each other, is the moving d-q coordinate system (see diagram, next page). The orthogonal d-q (d stands for direct, q for quadrature) coordinates rotate at the synchronous speed ω_m with respect to the stationary reference frame. Projecting the stator current vector $\mathbf{i}_s$ onto d-q yields the components i_{qs} and i_{ds}.

To control torque and field, we must control i_{qs} and i_{ds}. In order to achieve this objective, the stator current vector $\mathbf{i}_s$ must be oriented to d-q. (γ is the orientation angle). That is, since the only controllable variables are those in the stationary frame of reference (i.e. i_α and i_β), the stator windings must be energized by a current vector, that at any point of time is rotated by angle γ. Notice that this is an indirect way of controlling i_{qs} and i_{ds}.

In an induction motor, the rotor speed, ω_r, lags behind the synchronous speed at the rate of slip frequency ω_s. The orientation angle, γ, and synchronous speed ω_m are related via equation (3):

$$\omega_m = d\gamma / dt$$

In order to calculate γ, the slip frequency ω_s must be added to rotor speed ω_r, and the result integrated over time. Then to obtain i_{qs} and i_{ds}, use the transformation matrix represented by equation (4):

$$\begin{vmatrix} i_{ds}^* \\ i_{qs}^* \end{vmatrix} = \begin{vmatrix} -\cos\gamma & \sin\gamma \\ \sin\gamma & \cos\gamma \end{vmatrix} \begin{vmatrix} i_\alpha \\ i_\beta \end{vmatrix}$$

We must remember that the above relations were derived for a two-phase system. To make the findings useful for a conventional three-phase motor, we must transfer equations (2) and (4) back to three phase configuration using inverse transforms. The actual implementation of the inverse of equation (4) in TMS320 code is shown on the next page.

Relation between i_{qs} and i_{ds}

In a vector controlled ac induction motor, the relation between the two components of current in field coordinates simplifies to equation (5):

$$i_{qs}/i_{ds} = \omega_s \cdot T_r$$

Simple inspection of equation (5) reveals that for a fixed field command (i_{ds} = constant), the slip value is

forced to linearly follow the generated torque. Under this control, eq. (1) can be simplified to equation (6):

$$T = \frac{3P}{4}\frac{L_m^2}{L_r} i_{ds} \cdot i_{qs}$$

Therefore the value of the torque constant K_t, for a vector controlled ac induction motor, when compared to an ordinary dc motor can be given as equation (7):

$$K_t = \frac{3P}{4}\frac{L_m^2}{L_r} i_{ds}$$

K_t is a value that represents the field strength. Equation (7) shows that K_t can be controlled by changing i_{ds}. This is especially gratifying in operations that demand a wide range of speed and torque control. The act of decreasing the flux value by means of reducing the field current to expand the speed range is referred to as "field weakening."

The block diagram

The figure at the top of the previous page shows a functional block diagram of the control system for a vector controlled ac induction motor in a velocity loop. In addition to velocity loop, there are loops for i_{qs}, i_{ds}, and the magnetizing current i_{mr}. All of the calculations in the block diagram—the control calculations, matrix multiplications, and so on—can be done by software running on a single-chip digital signal processor.

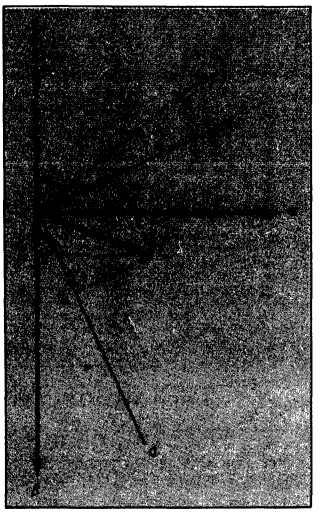
Block number 1, which is the block at the right enclosed with dotted lines and containing four smaller yellow blocks, is the "vector rotator block." The current vector is rotated from the moving reference to the stationary one (top two yellow blocks) and back to the moving reference (bottom two yellow blocks). Block 1 implements equations (2) and (4) in the top two yellow blocks and their inverse transforms in the bottom two yellow blocks.

Dotted-line block 2 shows the control actions working on the velocity and current errors.

Block number 3, also enclosed in dotted lines, illustrates how the slip angular velocity, ω_s, is derived and used in obtaining the magnetizing current angle γ. Note that γ is used as an input to the vector rotator block. Function blocks 1, 2 and 3 are computationally demanding ones; there are, however simpler control blocks that we do not address here.

Starting from block 1, we notice that each vector rotation requires the multiplication of a matrix by a vector. Transformation of 2 to 3, and the inverse transfer of 3 to 2 phases, require several adds and multiplies. The DSP code for one of these matrix multiplications (the inverse of equation 4) is shown in the box at the bottom of this page.

Block 2 involves several computations of angle γ using rotor speed and calculated slip frequency. We can see the need for two multiplies, a divide and precise integration or accumulation of synchronous speed.



Vector rotation diagram to define coordinates and angles in the d-q representation.

In spite of the demanding requirements of this control system, the entire system can be implemented with a single TMS320 DSP device.

DSP systems

DSP systems, like control systems, have special requirements to allow efficient implementation of those algorithms. DSP devices implement new architectures to provide solutions for these requirements. Initial DSP devices were expensive and did not have the functionality available on microcontrollers (a situation similar to early microprocessors). But prices have dropped tremendously, and today DSPs are at par with 16-bit microcontrollers.

Furthermore, DSPs are being introduced that have most of the functionality of microcontrollers. In the future, the functionality of DSP devices may be indistinguishable from that of ordinary microcontrollers.

Microcontrollers were designed to replace hardwired logic; DSPs were designed for signal processing. As system costs of DSPs goes down, they will eventually replace analog systems and microcontrollers in most servo control applications. □

REFERENCES

1. R. Lessmeier, W. Schumacher, and W. Leonhard, "Microprocessor controlled ac servo drives with synchronous or induction motors: which is preferable?" *IEEE-IAS Conference Papers, 1985 pp. 529*
2. S. Meshkat, *Motion Research Servo Design Tutorial*, Motion Research Inc., Plymouth, Minn.
3. S. Meshkat, E. Persson, "Optimum current vendor control of ac amplifiers using microprocessors." *IEEE-IAS Conference Papers, 1984 pp. 451*

TMS320 Code for Vector Rotation

```
VECROT  ZAC              ; clear accumulator
        LT      COSGAMA  ; load first operand
        MPY     IDS      ; IDS x cos gamma
        LTA     SINGAMA  ; load next op, + to accum
        MPY     IQS      ; IQS x sine gamma
        APAC             ; ACC = IQS x sine gamma
        SACH,1  IALPHA   ; stor up, 17 bits w/ shift of 1
        ZAC              ; clear ACC
        LT      SINGAMA  ; load sin gamma into T reg
        MPY     IDS      ; IDS x sine gamma
        LTA     COSGAMA  ; load cos gamma and
                         ; add prev result to ACC
        MPY     IQS      ; IQS x cos gamma
        SPAC             ; ACC = Ids sine gamma-
                         ; IQS cos gamma
        SACH,1  IBETA    ; store result
```

The inverse transform of equation (4) requires 14 cycles to do 4 multiplies, 1 add and 1 subtract. Execution time is 2.4 μsec at 25 MHz.

Microprocessor-Controlled AC-Servo Drives with Synchronous or Induction Motors: Which is Preferable?

R. LESSMEIER, W. SCHUMACHER, AND W. LEONHARD, SENIOR MEMBER, IEEE

Abstract—**With the recent advances of power transistors and microprocessors it has become possible to design high-dynamic-performance ac-servo drives free of moving contacts using synchronous or asynchronous motors. Both schemes have their particular strengths. A general control principle, based on field or rotor orientation, is described which has been realized with a state-of-the-art microcomputer, where all the signal processing, including modulation of the inverter, is performed by software. Extensive tests have been carried out with different motors to compare the characteristics of the various types of drives.**

INTRODUCTION

CONTROLLED electrical drives with high dynamic performance are today almost invariably dc drives fed by power electronic converters. At larger ratings and in stationary applications the converters are of the line-commutated type, presenting an acceptable compromise between dynamic performance, efficiency, and cost. The dc motors, with their transparent control structure, are well suited for high-performance duty: the separately excited field affords flexibility and permits an enlarged speed range at reduced torque, similar to a continuously variable gear.

For servo applications between 1–10 kW, the motors are normally fed from dc-link transistor converters switching at higher frequency (1–5 kHz) to improve the response; the dc link is supplied from a line-side rectifier or a battery. The field winding is usually replaced by permanent magnets, thus excluding the possibility of field-weakening.

To minimize motor inertia, which is important for rapid acceleration, two types of dc-servo motors have evolved: the slim-drum type motor of otherwise conventional design and the disk motor, having an iron-free armature and axial magnetic field. The first is often used for machine tool feed drives, while the second is preferred on robots because of its compact design and short axial length.

Paper IPCSD 86-8, approved by the Industrial Drives Committee of the IEEE Industry Applications Society for presentation at the 1985 Industry Applications Society Annual Meeting, Toronto, ON, Canada, October 6–11. Manuscript released for publication March 19, 1986. This paper is based on work suported by a grant from Deutsche Forschungsgemeinschaft.

R. Lessmeier is with Technische Universitat Braunschweig, Department of Control Engineering, Hans-Sommer-Strasse 65/66, 3300 Braunschweig, West Germany.

W. Schumacher is with Technische Universitat Braunschweig, Institute of Applied Microelectronics, Hans-Sommer-Strasse 65/66, 3300 Braunschweig, West Germany.

W. Leonhard is with Technische Universitat Braunschweig, Institut fur Regelungstechnik, Hans-Sommer-Strasse 65/66, 3300 Braunschweig, West Germany.

IEEE Log Number 8609876.

The motor is normally coupled to the mechanical load through gears because, with an electrical drive, a large power-to-weight ratio calls for high rotational speed. Of course the mechanical commutator sets a limit, often at 3000 min^{-1}; furthermore there are restrictions on temporary torque overload, particularly at very low speed or standstill, which is a frequent mode of operation with position-controlled servo drives.

It is for these reasons that there is intense interest in commutatorless ac-servo drives where these restrictions are lifted; it could eventually open the way to lightweight motors operating at very high speed, for example beyond 10 000 min^{-1}, or to high-torque gearless direct drives.

The obstacles for controlled ac drives for servo applications have in the past been twofold:

- the cost of the power converters and
- the complexity of an ac motor as a nonlinear multivariable control plant.

With the rapid development of semiconductors, solutions for both problems are in sight.

- Fast-switching bipolar and field-effect power transistors requiring minimal snubbing circuits and in cost-saving modular assemblies are becoming available.
- The complex control systems necessary with ac motors can be realized through software on ever more powerful microelectronic components.

This has resulted in accelerated research and has led to the emergence of prototype and early commercial high-performance ac drives. It is the aim of this paper to assess the control aspects of ac-servo drives and to compare the relative merits of different types of ac motors.

AC-Servo Drive with Transistor Inverter and Microcomputer Control

The tasks of electromechanical power and ac/dc conversion, which are jointly carried out in the armature of a dc motor, are separated in the ac drive, resulting in greater flexibility with regard to motor design. The magnetic flux necessary for producing torque is set up either by permanent magnets in the rotor or by magnetizing current in the stator windings. Thus synchronous or induction motors result, both of which can be designed as slim-drum or short-disk type motors.

Synchronous motors with permanent-magnet excitation may be further classified in those having an approximately sinusoi-

dal flux distribution in the airgap and sinusoidal stator currents and the so-called brushless dc motors with built-in position sensor, having a trapezoidal flux distribution and a current-source dc link. Only the synchronous motor with approximately sinusoidal stator currents (below voltage limit) and the induction motor will be discussed here. Both are supplied from a voltage-source transistor inverter with pulsewidth modulation; best dynamic performance is obtained by employing constant link voltage. Both drives permit field-weakening and four-quadrant operation, even though the power generated during dynamic braking is usually absorbed in a ballast resistor to simplify the line-side converter.

A digital pulsewidth modulator may be directly coupled to the microcomputer controlling the drive. If the sampling frequency at which the complete control algorithm is repeated is identical with the pulse frequency of the power inverter, there is the additional benefit that the ripple on the current signals may be greatly reduced without the need for smoothing filters. If the desirable stator frequency is 200 Hz, this calls for a switching frequency of the inverter of about 4 kHz, leaving 250 μs for performing the control algorithm. This is feasible with a high-speed microcomputer containing a signal processor.

Another feature of a purely digital control scheme is that the stator currents can be impressed by current control loops that are closed in transformed coordinates so that the software current controllers are processing dc quantities in steady state. This is explained in more detail later.

To arrive at a valid comparison of the different types of drives, the same inverter and microcomputer hardware will be used for all the tests. The control algorithm is identical, with the exception that the synchronous motor is controlled in rotor coordinates while the induction motor control is performed in field coordinates, based on rotor flux. This calls for a flux model, which is bypassed in the case of the synchronous motor. Obviously, when designing a control scheme for exclusive use with synchronous motors, the program could be simplified or a slower processor would suffice.

For measuring motor speed and position, an optical incremental sensor with 1024 lines/r is attached to the motor shaft. To achieve smooth operation at very low speed and standstill, the analogue sin-, cos-signals are evaluated with A/D converters prior to quantization so that more than a quarter million (2^{18}) increments/r are available for interpolation; outside the low-speed range (± 150 min^{-1}) the additional bits for high resolution become meaningless and are disregarded.

Mathematical Model of Symmetrical AC Motors

All the motors employed for the tests show rotational symmetry including constant effective airgap. In the case of the synchronous motor this is achieved by placing rare-earth magnets directly on the circumference of the rotor. Since these magnets exhibit low permeability to external fields and high resistivity, they may be considered as part of the airgap. It has been shown [18]–[21] that with the usual simplifications, such as symmetrical three-phase windings and sinusoidal MMF distribution (no spatial harmonics), smooth stator and rotor surfaces (no slots), neglecting saturation and iron losses, a two-pole motor with symmetrical stator and rotor windings can be described by a set of four nonlinear differential equations:

$$R_S \underline{i}_S + L_S \frac{d\underline{i}_S}{dt} + L_0 \frac{d}{dt}(\underline{i}_R e^{j\epsilon}) = \underline{u}_S(t), \tag{1}$$

$$R_R \underline{i}_R + L_R \frac{d\underline{i}_R}{dt} + L_0 \frac{d}{dt}(\underline{i}_S e^{-j\epsilon}) = \underline{u}_R(t), \tag{2}$$

$$J \frac{d\omega}{dt} = m_d - m_L = \frac{2}{3} L_0 Im[\underline{i}_S (\underline{i}_R e^{j\epsilon})^*] - m_L, \tag{3}$$

$$\frac{d\epsilon}{dt} = \omega. \tag{4}$$

The instantaneous phase currents and voltages are combined to form complex time-varying vectors in the plane perpendicular to the motor axis:

$$\underline{i}_S(t) = i_{S1}(t) + i_{S2}(t)e^{j\gamma} + i_{S3}(t)e^{j2\gamma}, \quad \gamma = \frac{2\pi}{3} \tag{5}$$

and

$$\underline{u}_S(t) = u_{S1}(t) + u_{S2}(t)e^{j\gamma} + u_{S3}(t)e^{j2\gamma}. \tag{6}$$

Corresponding definitions hold for the rotor currents and voltages. The following symbols are used.

R_S, R_R	Winding resistances per phase.
L_S, L_R, L_0	Stator, rotor, and mutual inductances per phase, assuming same number of turns in stator and rotor.
J	Inertia of motor.
m_d, m_L	Electrical driving torque and load torque at motor coupling.
ω	Angular velocity.
ϵ	Angle of rotation.
$\underline{i}_R^*$	Conjugate complex vector of $\underline{i}_R$.

The model equations are valid for any waveform of currents and voltages as long as the condition for isolated neutral, e.g.,

$$i_{S1} + i_{S2} + i_{S3} = 0, \tag{7}$$

is maintained.

This unified model may be adapted to the constraints of an induction motor with cage rotor by introducing the short-circuit condition at the rotor terminals $\underline{u}_R = 0$. The model of a synchronous motor with permanent-magnet excitation is obtained when the fictitious rotor windings are supplied from assumed dc sources, $i_R = 2/3\ I_F$, rendering the rotor voltage equation superfluous. Further simplifications result when the stator currents are impressed by current sources, even though the current control loops may in fact be closed in a transformed coordinate system as is shown in the next paragraph. This finally leaves (2)–(4) as the dynamical models for controlling the induction motor and (3), (4) for controlling the synchronous motor.

Control of AC Motors in Moving Coordinates

The principle of field orientation as formulated by Blaschke [2] has emerged as a very effective method for controlling ac

machines with high dynamic performance; it may be applied to asynchronous as well as synchronous motors. With a stator-fed machine, it calls for transformation of the stator current vector into a moving frame of reference given by the rotor current vector (or rotor flux vector—whichever is more convenient to access). By splitting the transformed stator current vector into direct and quadrature components i_{Sd}, i_{Sq}, respectively, inputs for decoupled control of flux and torque are obtained as is the case with a dc machine.

Control of Synchronous Motor

With a permanently excited synchronous motor this principle is easy to apply, because the fictitious rotor current vector is fixed to the rotor position,

$$\underline{i}_R(t) = (3/2) I_F e^{j\epsilon}. \tag{8}$$

Hence the mechanical equation (3) becomes

$$m_d = L_0 I_F Im[\underline{i}_S e^{-j\epsilon}] = \Phi_F i_S \underbrace{\sin(\xi - \epsilon)}_{\delta} \tag{9}$$

$$= \Phi_F i_{Sq}$$

where $\delta = \xi - \epsilon$ is the load angle and

$$i_{Sq} = i_S \sin \delta \tag{10}$$

the quadrature current in a rotor-based coordinate system (Fig. 1). Maximum torque for a given stator current is obtained for $\delta = \pm \pi/2$, i.e., purely quadrature current. In the base speed range this is the optimal mode of operation but at higher speed; where the maximum inverter voltage would be exceeded a negative direct component $i_{Sd} < 0$ may be introduced for limiting the terminal voltage of the motor.

Control of Induction Motor

Field-oriented control of an induction motor is much more difficult because the rotor flux moves across the rotor at slip speed and the rotor currents cannot be measured directly. The rotor flux may be characterized by an equivalent stator-based magnetizing current containing a component for magnetic leakage

$$\underline{i}_{mR}(t) = \underline{i}_S + (1 + \sigma_R)\underline{i}_R e^{j\epsilon} = i_{mR} e^{j\rho}, \tag{11}$$

where σ_R is the coefficient representing rotor leakage. The angle ρ may be used as a frame of reference for field orientation. Instead of direct flux-sensing schemes (Hall sensors, search coils, etc.) it was found to be more effective to compute the magnetizing current in a dynamic model on the basis of terminal quantities and speed [11], [15], [21]; no modifications of the motor are then required. The model remains operative even at standstill, which is an important condition for servo drives. Combining (2) and (11) results in

$$T_R \frac{di_{mR}}{dt} + i_{mR} = Re[\underline{i}_S e^{-i\rho}] = i_{Sd}, \tag{12}$$

$$\frac{d\rho}{dt} = \omega_{mR} = \omega + \frac{1}{T_R}\frac{1}{T_R i_{mR}} Im[\underline{i}_S e^{-i\rho}] = \omega + \frac{i_{Sq}}{T_R i_{mR}}, \tag{13}$$

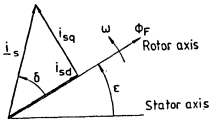
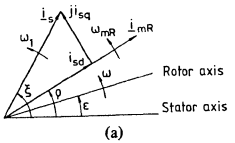


Fig. 1. Coordinates for controlling synchronous motor.



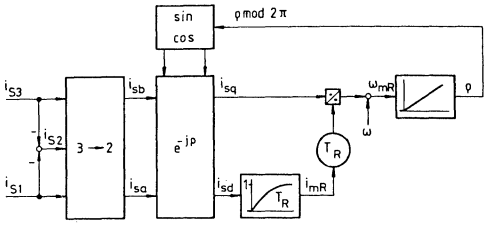


Fig. 2. Induction motor. (a) Coordinates for motor control. (b) Flux model for motor control.

which represent a flux model with the stator currents and speed serving as input signals (Fig. 2a, 2b). By solving these equations with the microcomputer in real time and with adequate precision, estimates for the modulus and angle of the magnetizing current, as well as the components of the transformed stator current vector and the electrical torque, are obtained.

The only uncertain parameter is the rotor time constant T_R, which depends on rotor temperature and saturation. Temperature effects are quite slow, but the degree of saturation changes rapidly when the motor accelerates into the field-weakening range. It has been shown [12], [16], [31], [35] that an estimate of T_R may be obtained by utilizing terminal voltages except at very low speed, where all voltage measurements become uncertain due to the large resistive component. The T_R adaptation is performed by comparing the vector of the measured stator voltages with that derived from the flux model; if differences are detected, the model parameter T_R is changed accordingly.

Measuring the fundamental components of the terminal voltages proves difficult with a PWM inverter because of the highly distorted waveforms. However, considerable simplifications result when the current control loops are closed within the microcomputer, because the voltage reference signals are then available in the computer program and there is no need for voltage sensors.

Realization of the Control Scheme with a Microcomputer

An overview of the control scheme is shown in Fig. 3, containing the hardware and software structure in the usual

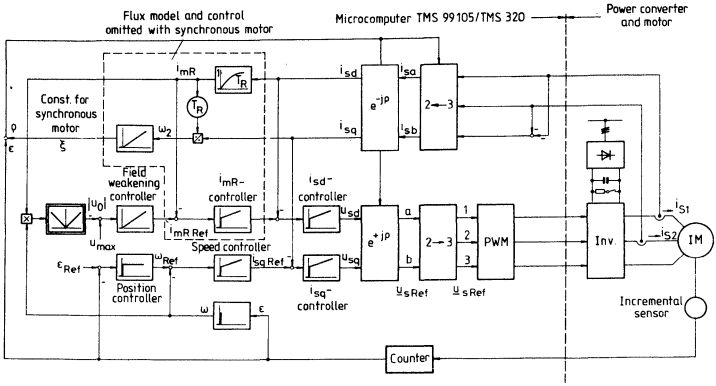


Fig. 3. Block diagram for ac motor control in moving frame of coordinates.

form of a block diagram. Compared with an earlier realization [32] the following features have been implemented.

- Totally digital control by employing a single signal processor.
- Sampling period of 256 μs for all control loops.
- Improved resolution of current and voltage signals.
- T_R adaptation in case of the induction motor.
- Minimum control delay between the sampling instant of the currents and the center of the subsequent stator voltage pulse.

The Microcomputer

Both the complex control algorithms already discussed and the short sampling period necessary for achieving a high dynamic performance drive call for considerable computing power (see Fig. 4). In particular, the two transformations from stator to field coordinates and the reverse (each requiring four multiplications with sin ρ, cos ρ, as well as the flux model for the induction motor, are computationally demanding, but a cost-effective microelectronic solution has become possible with signal processors [30]. This method has been further advanced by performing all arithmetic operations necessary for controlling the motors in a single-signal processor: TMS 32010. This arithmetic unit is tied by two-port RAM to a control processor, TMS 99105, which handles the I/O interface and the communication with an operator's console. The double processor card (233 × 160 mm) is suitable for connection to a multiprocessor bus to accommodate multi-axis drives such as found on machine tools and robots. A local bus connects each processor card to an interface card containing the A/D converters and modulator.

The Interface Card

As explained earlier, control of a high-performance ac-servo drive requires the measurement of instantaneous stator currents, rotor velocity, and rotor position. Both mechanical signals are derived from an optical incremental sensor with 1024 lines/r, which has been augmented by analogue techniques to the very high resolution necessary for smooth operation at very low speed and standstill. With the 256 μs sampling period, the speed increment is 0.2 min^{-1} [33].

Two of the stator currents are measured by commercial magnetic sensors employing Hall devices. They provide electrical insulation and approximately 50-kHz bandwidth. The output signals of the sensors are sampled by two 12-bit A/D converters with 12-μs settling time.

The output signals of the controller, representing reference values for the three-phase voltages, are transferred to a digital pulsewidth modulator, which is designed with transistor–transistor logic (TTL) providing pulsewidth increments of 1/16 μs, which corresponds to 12-bit resolution of the 256-μs sampling period.

All these I/O channels are placed on the interface card.

The Transistor Inverter

The voltage-source three-phase transistor inverter is supplied with a constant dc link voltage U_D = 300 V, generating a maximum sinusoidal line-to-line voltage U_{rms} = 220 V. Several bipolar transistors are parallel-connected in each leg of the inverter bridge circuit, allowing a maximum sinusoidal output current of I_{rms} = 40 A. Of course the maximum output power of 15 kVA is not always used during the tests where the current limit is set to a value compatible with each machine. The switching frequency of the inverter can be raised to 20 kHz, i.e., beyond the audible range; but to achieve synchronous operation with the control unit, the frequency was set at 4 kHz.

List of Motors

To demonstrate the flexibility of the microprocessor control unit, six different ac motors were used during the tests (corresponding to rotors shown in Fig. 5).

1) A specially designed low-inertia synchronous motor with SmCo magnets attached to a slim rotor [23]. The four-pole machine has a continuous rating of about 1.2 kW at 2000 min^{-1}
2) A specially designed low-inertia induction motor employing the same stator as for 1).

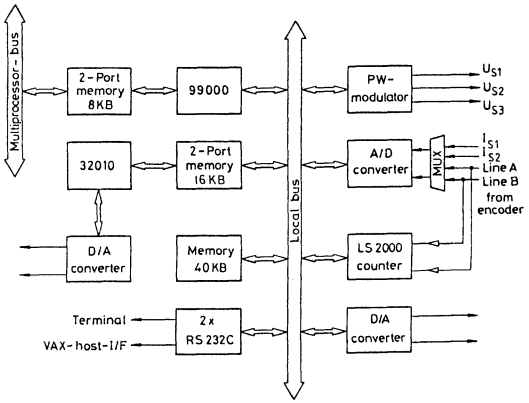


Fig. 4. Microcomputer for ac motor control.

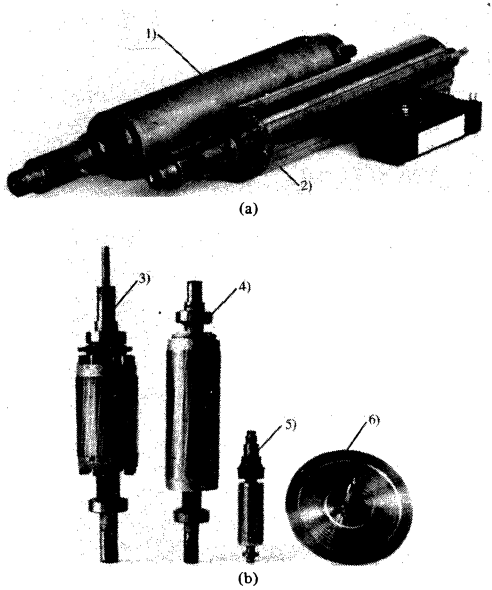


Fig. 5. Rotors of motors used during tests. (a) Synchronous and induction motor rotors (shown as 1) and 2), respectively). (b) Induction motor rotors.

3) A standard induction machine (4 poles, 1.5 kW at 1420 min^{-1}).
4) A special induction machine (4 poles, 1.5 kW at 1420 min^{-1}), which had roughly the shape of two axially joined standard motors.
5) An industrial high-speed induction motor (2 poles, 1 kW at 12 000 min^{-1}) designed for intermediate-frequency power tools.
6) A disk-type induction motor (4 poles) consisting of a commercial stator for a synchronous disk motor (1.5 kW at 6000 min^{-1}) and an aluminum rotor disk [30].

Test Results

To simplify the comparison, only the results with motors 1) and 2), which are identical in shape and employ the same stator with a two-layer three-phase winding skewed by one slot (to avoid lock-in effects), will be discussed. In view of the higher flux density possible with induction motor 2), the dc link voltage was raised to 450 V.

Figs. 6 and 7 show the results of tests in steady state, where the rms stator currents are plotted against no-load speed and torque at standstill. The synchronous motor with permanent magnets has a linear current–torque characteristic and a very low no-load current below rated speed; this is due to the absence of rotor losses and magnetizing current. However, when the motor is operated into the field-weakening region, which may be desirable for rapid traversing, the motor is overexcited and draws huge reactive current from the inverter, causing high stator losses. The reason is the relatively wide airgap of the synchronous motor. This large reactive current is in contrast to the magnetizing current of the induction motor, which is reduced above rated speed.

The test indicates that the absence of a magnetizing current in case of the synchronous motor below rated speed should not be overemphasized, because with a servo motor this is only a small fraction of the maximum current which may be required during temporary overload conditions; also, when the motors are loaded, reactive power is caused not only by magnetizing current but also by armature reaction and magnetic leakage, which is present with both motors. None of the motors can operate with unity power factor under load.

The overall steady-state performance of the two motors at rated torque is described by Fig. 8, which shows the efficiency between the dc link and the mechanical output measured by a dynamometer; hence the losses of the inverter are included. Clearly the synchronous motor is superior, which is due mainly to the rotor losses of the induction motor. At higher speed, where the stator losses of the synchronous motor rise, the differences tend to become smaller. The rotor losses are a definite disadvantage of the induction motor for servo applica-

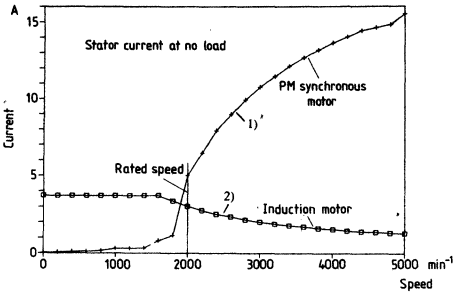


Fig. 6. Stator currents versus no-load speed for drives 1) and 2).

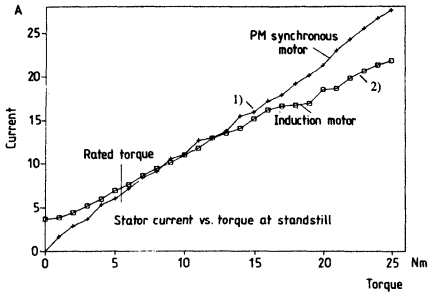


Fig. 7. Stator currents versus torque at standstill for drives 1) and 2).

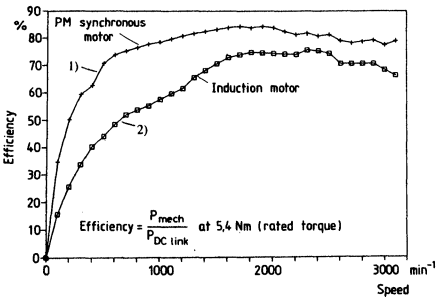


Fig. 8. Efficiency of drives 1) and 2) at rated torque.

tions because they may necessitate forced cooling while in the other case natural cooling might suffice.

The dynamic performance of the two drives with speed control at no-load is exemplified by the two frequency response curves shown in Fig. 9. The measurements were taken at small amplitude to avoid nonlinearities. The curves show flat response up to about 100 Hz. The difference between the two motors is not characteristic for the synchronous and induction motor drive, but depends also on slightly different controller settings.

Finally, large signal transients are depicted in Figs. 10 and 11, again for the motors 1) and 2). They show speed-reversing transients of the speed-controlled drives at no-load, where the maximum current was limited to 25 A, corresponding to approximately 5 times rated current. The curves show very rapid response; at 1000 min^{-1} the braking distance is about 20°, roughly one slot division. The difference between the motors is minimal, even though the inertia of the induction motor is larger due to the copper cage winding. At higher speed the response of the induction motor is somewhat delayed because of the field lag, which becomes effective when the field is weakened.

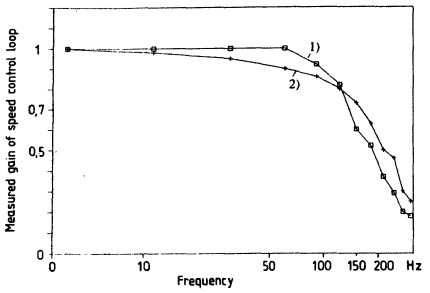


Fig. 9. Frequency response curves of speed-controlled drives 1) and 2) at no-load.

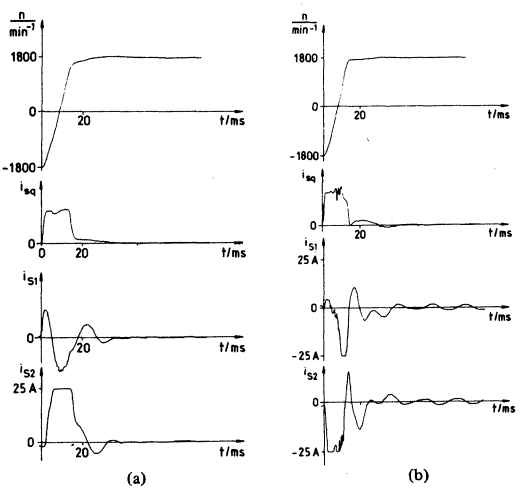


Fig. 10. No-load reversing transients for speed-controlled drives. (a) Drive 1). (b) Drive 2).

Finally in Fig. 12 large signal step responses are shown, with the position control loops closed; the position controllers used for these tests were of the nonlinear, time-optimal response type. The differences between the two drives are again negligible. The performance of these drives is quite remarkable; following a change in position reference, the torque responds within one sampling period of the controller, i.e., after 250 μs.

As a result of these tests it can be stated that very high-performance ac-servo drives can be designed with pulsewidth-modulated transistor inverters and microcomputer control. Synchronous motors with permanent-magnet excitation and

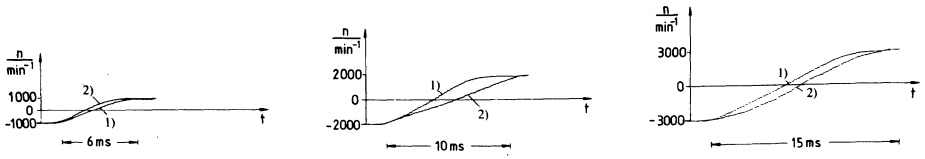


Fig. 11. No-load reversing transients for speed-controlled drives 1) and 2) below and above rated speed.

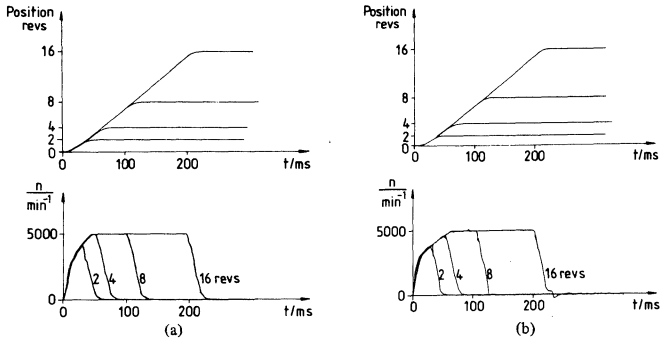


Fig. 12. Step response of position-controlled drives employing time-optimal control. (a) Drive 1). (b) Drive 2).

induction motors are both suitable. In our view, the choice is still open because it depends on a number of factors that can only be determined with fully optimized designs and more industrial experience. The main points to be taken into consideration are as follows.

- With regard to the rating of the inverter, the synchronous motor has a slight advantage as long as field-weakening is avoided. This is mainly due to the rotor losses in the induction motor; the fact that the synchronous motor needs no magnetizing current seems less important in view of the fact that servo drives are normally rated for intermittent duty and high short-time overload.
- There will be borderline cases where the synchronous motor can be operated without forced cooling while the induction motor would normally need forced cooling.
- The induction motor permits easy field-weakening over a wide speed range with constant power; this makes this motor particularly attractive for spindle drives, but it is also applicable for position-controlled feed drives.
- The induction motor, even when specially designed for low inertia and leakage, is likely to be less expensive than the synchronous motor with rare-earth permanent magnets. The design would take advantage of the fact that full line voltage starts do not occur with inverter-fed motors. Also, the induction motor can be designed for higher flux density than the synchronous motor with permanent magnets.
- The microcomputer can be simpler for the synchronous motor because no signal processor is required. In principle it could be designed without a microprocessor at all, given the capabilities of VLSI custom design, but the additional flexibility in the speed and position-control function are a definite asset of control by software.

Conclusions

Extensive laboratory tests have been conducted to explore the design of microcomputer-controlled ac-servo motors with synchronous and induction motors fed by a pulsewidth modulated transistor inverter. The control is all digital, employing 4-kHz sampling frequency and synchronous switching of the inverter. The results obtained thus far show that both types of motor are applicable even though there are special strengths and weaknesses with each of them. One criterion could be the condition of a wide field-weakening speed range, which would rule out the synchronous motor; on the other hand the induction motor has rotor losses and is more difficult to control.

Only time will tell which design will eventually prove superior in the majority of applications; our belief is that there will be a bright future for both of them.

References

[1] A. Abbondanti, "Method of flux control in induction motors driven by variable frequency, variable voltage supplies," in *Proc. IEEE/IAS Int. Semicond. Power Conv.*, 1977.

[2] F. Blaschke, "The principle of field orientation as applied to the new TRANSVECTOR closed-loop control system for rotating field machines," *Siemens Rev.*, p. 217, 1972.

[3] F. Blaschke and K. Böhm, "*Verfahren der Flußerfassung bei der Regelung stromrichtergespeister Asynchronmaschinen*," presented at the IFAC Symp., Control in Power Electronics and Electrical Drives, Düsseldorf, VDI/VDE Ges. Mess-und Regelungstechnik, p. 635, 1974.

[4] B. K. Bose, "Adjustable speed AC drives—A technology status review," in *Proc. IEEE*, p. 116, 1982.

[5] A. Brickwedde, D. Head, and H. Graham, "Microprocessor controlled

50 kVA PWM inverter motor drive," in *Conf. Rec. 1981 Ann. Meeting IEEE Ind. Appl. Soc.*, p. 660.
[6] G. Chollet, "*Commande par microprocesseur d'un commutateur de courant pour moteur asynchrone de traction*," in *Proc. Conumel 83*, Toulouse, pp. IV-1.
[7] C. Conrath, "*Commande a microprocesseur d'une machine asynchrone autopilotée. Application a un systeme de levage*," in *Proc. Conumel 83*, Toulouse, pp. IV-7.
[8] R. Gabriel, "*Antriebsregelung einer thyristorgespeisten Asynchronmaschine durch einen Mikrorechner*," diploma thesis, Control Eng. Dept., Technical University of Braunschweig, 1978.
[9] R. Gabriel, W. Leonhard, and C. Nordby, "*Regelung der stromrichtergespeisten Drehstrom-Asynchronmaschine mit einem Mikrorechner*," *Regelungstechnik* 27, p. 379, 1979.
[10] R. Gabriel, W. Leonhard, and C. Nordby, "Field-oriented control of a standard AC-motor using microprocessors," *IEEE Trans. Ind. Appl.*, IA-16, no. 2, p. 186, 1980.
[11] R. Gabriel and W. Leonhard, "Microprocessor control of induction motor," in *Proc. 1982 IEEE Int. Semicond. Power Conv. Conf.*, p. 385.
[12] L. J. Garces, "Parameter adaption for the speed controlled static ac-drive with squirrel cage induction motor," in the *Conf. Rec. of the 1979 IEEE Ind. Appl. Soc. Ann. Meet.*, p. 843.
[13] M. Grotstollen and G. Pfaff, "*Bürstenloser Drehstrom-Servoantrieb mit Erregung durch Dauermagnete*," *Elektrotech. Bd.*, 100 (1979), p. 1382.
[14] K. Hasse, "*Zur Dynamik drehzahlgeregelter Antriebe mit stromrichtergespeisten Asynchronkurzschlußläufermaschinen*," Ph.D. thesis, Technical University of Darmstadt, 1969.
[15] R. Jötten and G. Mäder, "Control methods for good dynamic performance induction motor drives based on current and voltage as measured quantities," presented at the 1982 IEEE Int. Semicond. Conv. Conf., p. 397.
[16] T. Irisa, S. Takata, R. Ueda, T. Sonoda, and T. Mochizuki, " A novel approach on parameter self-tuning method in AC-servo systems," presented at the 1983 IFAC Symp. Control in Power Electron. and Electric. Drives, Lausanne, p. 41.
[17] I. Iwakane, H. Inokuchi, T. Kai, and J. Hirai, "AC servo motor drive for precise positioning control," in *1983 Proc. Int. Power Electr. Conf.*, Tokyo, p. 1453.
[18] K. P. Kovacz, and J. Racz, *Transiente Vorgänge in Wechselstrommaschinen*. Budapest: Hungarian Acad. of Science, 1959.
[19] W. Leonhard, *Regelung in der elektrischen Antriebstechnik*. Stuttgart: B. G. Teubner, 1974.
[20] W. Leonhard, "Control of ac machines with the help of microelectronics: A survey" presented at the Control in Power Electron. and Electric. Drives IFAC Symp., Lausanne, 1983. Oxford: Pergamon Press, 1984. Published in *Automatica* 1986, p. 1.
[21] W. Leonhard, *Control of Electrical Drives*. Berlin, Heidelberg, New York, Tokyo: Springer, 1985.
[22] H. H. Letas and W. Leonhard, "Dual axis servo drive in cylindrical coordinates using permanent magnet synchronous motors with microprocessor control," in *Proc. Conumel 83*, Toulouse, p. 11-23.
[23] H. H. Letas, "Mikrorechner-geregelter Synchron-Stellantrieb," Ph.D. thesis, Technical University of Braunschweig, 1985.
[24] B. Mokrytzki, "Pulse width modulated inverters for ac motor drives," *IEEE Trans. Ind. Appl.*, p. 312, 1968.
[25] J. M. D. Murphy, L. S. Howard, and R. G. Hoft, "Microprocessor control of a PWM-inverter induction motor drive," in *Conf. Rec. 1979 Power Electr. Spec. Conf.*, p. 344.
[26] A. Nabae, K. Otsuka, H. Uchino, and R. Kurosowa, "An approach to flux control of induction motors operated with variable-frequency power supply," *IEEE Trans. Ind. Appl.*, p. 342, 1980.
[27] D. Pauly, G. Pfaff, and A. Weschta, "Brushless servo drives with permanent magnet motors or squirrel cage induction motors—A comparison," presented at the 1984 IEEE/IAS Ann. Conf., p. 503.
[28] G. Pfaff, A. Weschta, and A. Wick, "Design and experimental results of a brushless AC servo drive," in the *Conf. Rec. of the 1982 IEEE Ind. Appl. Ann. Meeting*, p. 692.
[29] A. B. Plunkett, "Direct flux and torque regulation in a PWM-inverter induction motor drive," *IEEE Trans. Ind. Appl.*, IA-13, no. 2, 1977.
[30] W. Schumacher, "Microprocessor controlled AC servo drive," in *Proc. Microelectronics in Power Electron. and Electric. Drives*, Darmstadt, 1982, p. 311.
[31] W. Schumacher and W. Leonhard, "Transistor-fed AC servo drive with microprocessor control," in *Proc. 1983 Int. Power Electr. Conf.*, Tokyo, p. 1465.
[32] W. Schumacher, H. H. Letas, and W. Leonhard, "Microprocessor controlled ac-servo drive," in *Proc. 1984 IEE Conf. on Power Electron. and Variable Speed Drives*, London, p. 233.
[33] W. Schumacher, P. Rojek, and H. H. Letas, "*Hochauflösende Lage- und Drehzahlerfassung optischer Geber für schnelle Stellantriebe*," *Elektronik*. p. 65, 1985.
[34] W. Schumacher, "*Mikrorechner-geregelter Asynchron-Stellantrieb*," Ph.D. thesis, Technical University of Braunschweig, 1985.
[35] Y. Yoshida, R. Ueda, and T. Sonoda, "A new inverter-fed induction motor drive with a function of correcting rotor circuit time constant," in *Proc. 1983 Int. Conf. on Power Electr.*, Tokyo, 1983, p. 672.

Rainer Lessmeier was born in 1956 at Oerlinghausen, West Germany. From 1975 to 1984 he studied electrical engineering at the Technical University of Braunschweig, Braunschweig, West Germany, where he received his Dipl. Ing. degree in electrical engineering in 1984.

Since then he has been a Research Associate in the Department of Control Engineering, Technical University of Braunschweig, working for the Dr. Ing. degree.

Walter Schumacher was born in 1952 at Hamburg, West Germany. From 1973 to 1979 he studied electrical engineering at the Technical University of Braunschweig, Braunschweig, West Germany, where he received the Dipl. Ing. degree in electrical engineering in 1979.

From 1979 to 1984 he was with the Department of Control Engineering, Technical University of Braunschweig, where he received the Dr. Ing. degree in 1985. Since 1984 he has been with the Institute of Applied Microelectronics in Braunschweig, West Germany.

Werner Leonhard (M'56–SM'80) was born in 1926 at Weiden, Germany. From 1946 to 1951 he studied electrical engineering at the Technical University of Stuttgart, West Germany, where he received the Dipl. Ing. degree in 1951 and the Dr. Ing. degree in 1954.

He was with Westinghouse Electric Corporation, Pittsburgh, PA, from 1954 to 1958 and with Siemens-Schuckert-Werke AG, Erlangen, West Germany, from 1959 to 1963. Since then he has been a Professor of control engineering at the Technical University of Braunschweig, Germany, where he has also served as Head of the Electrical Engineering Department and Dean of the Mechanical and Electrical Engineering Faculty.

Dr. Leonhard is the author seven textbooks on various aspects of control, and was Chairman of the International Federation for Automatic Control (IFAC) Symposia on Control in Power Electronics and Electrical Drives in 1974 and 1977, respectively, in Düsseldorf, West Germany, as well as Chairman of the *Verband Deutscher Elecktrotechniker* (VDE) Conference on Microelectronics in Power Electronics and Electrical Drives, in 1982 in Darmstadt, West Germany. He is a co-founder of the European Power Electronics Conference, first held in 1985 in Brussels, Belgium.

A Microcomputer-Based Control and Simulation of an Advanced IPM Synchronous Machine Drive System for Electric Vehicle Propulsion

BIMAL K. BOSE, SENIOR MEMBER, IEEE, AND PAUL M. SZCZESNY

Abstract—**Advanced digital control and computer-aided control system design techniques are playing key roles in the complex drive system design and control implementation. The paper describes a high-performance microcomputer-based control and digital simulation of an inverter-fed interior permanent magnet (IPM) synchronous machine that uses a Neodymium-Iron-Boron magnet. The fully operational four-quadrant drive system includes a constant-torque region with zero speed operation and a high-speed field-weakening constant-power region. The control uses the vector or field-oriented technique in constant-torque region with the direct axis aligned to the stator flux, whereas the constant-power region control is based on torque angle orientation of the impressed square-wave voltage. All the key feedback signals for the control are estimated with precision. The drive system is basically designed with an outer torque control loop for electric vehicle application, but speed and position control loops can be added for other industrial applications. The distributed microcomputer-based control system is based on Intel-8096 microcontroller and Texas Instruments TMS32010 type digital signal processor. The complete drive system has been simulated using the VAX-based simulation language SIMNON[1] to verify the feasability of the control laws and to study the performances of the drive system. The simulation results are found to have excellent correlation with the laboratory breadboard tests.**

I. Introduction

INTERIOR OR buried type permanent magnet synchronous machines are showing increasing promise for industrial drive applications, and recently a considerable amount of research and development effort is being made in that direction. Because of buried magnet installation, IPM machines are robust and thus permit higher operating speed. The effective airgap in this class of machines is low and therefore armature reaction effect is very dominant. This permits control of the machine in constant-torque region as well as in field-weakening constant-power region up to a high speed such that the machine can be used for traction type applications. Again, the saliency ($X_{qs} > X_{ds}$) in this type of machine permits economical machine design because torque is contributed by the magnet field as well as by the reluctance effect. In the past, ferrite and Cobalt–Samarium magnets have generally been used in PM machines. Recently, a Neodymium–Iron–Boron (NeFeB) magnet has been introduced, which shows considerable promise. The NeFeB magnet has much higher energy density at reasonably "low cost" and therefore permits economical machine design. However, one characteristic of the magnet is that its field strength weakens as the temperature increases. Considering the recent research and development trends, it is expected that the price of the NeFeB magnet will fall considerably and its characteristics will improve, thus promoting extensive applications in the future.

In the past, induction motors have generally been considered as viable ac machines for electric vehicle drive applications. Although induction machines are simple, economical, and satisfy all the performance needs of electric vehicle drives, this type of machine has some additional loss penalties compared to the PM synchronous machine because of rotor copper loss. Besides conservation of energy, which is of paramount importance in the EV drive system, extensive analysis indicates that the life cycle cost of the IPM machine drive system is generally lower than that with an induction motor, not only for EV but for general industrial applications also. An IPM machine can be operated near unity power factor (unlike an induction machine), except at high speed and low torque where the power factor becomes low (leading) because of excessive counter emf.

The high performance requirements of the IPM machine drive system for EV application demands a considerable amount of control complexity and this is the subject of discussion in this paper. Fortunately, microcomputer technology and computer-aided control system design techniques have advanced tremendously in recent years and advanced control laws are being implemented easily in real time that could not be done before. The paper will first review the control principles of the IPM machine, which include constant-torque and constant-power regions. Then, after reviewing the salient features of the simulation language SIMNON, the drive system simulation is described. The hardware and software design features of the distributed microcomputer-based control are then described. Finally, laboratory tests that verify the simulation results are discussed.

II. Description of Control System

The complete drive control system of the IPM synchronous machine is described in [1]. It will be briefly reviewed here for completeness of the paper. Fig. 1 shows the simplified schematic of the drive system power circuit. The traction battery (204-V nominal) shown on the left is the lead-acid type

Manuscript received June 25, 1987; revised June 15, 1988. This work was supported by the Department of Energy under cost shared contract (Contract DE-AC07-85NV10418) with Ford Motor Company.

B. K. Bose was with the General Electric Research and Development Center, Schenectady. He is now with the Department of Electrical Engineering, The University of Tennessee, Knoxville, TN 37996-2100, and also with the Power Electronics Applications Center (PEAC), Knoxville, TN.

IEEE Log Number 8823552.

[1] Simnon was developed by Lund Institute of Technology, Sweden.

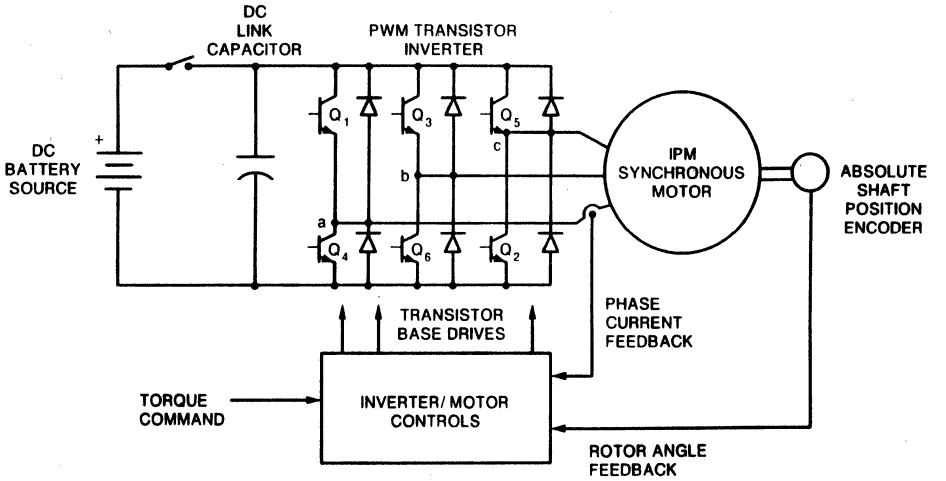


Fig. 1. Simplified schematic of the drive system power circuit.

and supplies power to the PWM transistor inverter. The inverter generates variable-frequency variable-voltage (or current) power supply for the IPM machine. The machine shaft power is transmitted to the drive axle through a two-speed transmission. The drive system operates in all four quadrants, and regenerative braking energy is easily absorbed by the battery. The machine shaft has an analog resolver type absolute position encoder that permits the drive system to be controlled in the "brushless dc machine" mode. The drive system has essentially two different modes of operation. In the constant-torque region the inverter is current-controlled in the PWM mode so that the desired flux-torque relationship can be maintained. In an IPM machine, the flux can be controlled by stator injected reactive current, which can be lagging (magnetizing) or leading (demagnetizing). As the inverter saturates at higher speed, the current control is lost and then the drive system enters into the field-weakening constant-power region. With this condition, the inverter generates a six-step square-wave voltage, which is phase-shifted to control the developed torque. As the machine speed increases in the constant-power mode, the induced voltage increases proportionally with speed, thus demanding more leading reactive current to balance with the constant stator voltage. The invertor/motor controls are shown as a block in Fig. 1 where transistor base drive signals are generated from the operator torque command and feedback signals.

A simplified control block diagram of the drive system in the constant-torque region is shown in Fig. 2. The core drive system in this region is current-controlled by using the hysteresis-band (bang-bang) PWM principle. The vector or field-oriented control principle is used to enhance the system transient performance. The IPM synchronous machine can be considered as somewhat analogous to a wound-field synchronous machine where the "field current" is controlled from the stator side. Therefore in vector control the direct axis has been aligned to the stator flux [2], [3], [16] instead of magnet flux. In such a control mode, the in-phase or active component of the stator current can be controlled to control the developed torque, whereas the quadrature or reactive component of the current can be controlled to control the stator flux. In Fig. 2, the operator commanded torque is controlled by the close loop and the torque component of current (I_T^*) is generated by the torque loop. The drive system incorporates a flux control loop to prevent flux drift due to parameter variation. The command flux (Ψ_s^*) is programmed with torque (T_e^*) to optimize the core loss so that the overall drive efficiency is improved. The flux is essentially controlled in the feed-forward manner with the help of the current program as shown, except the incremental ΔI_M^* from the flux loop supplements the current program output. The current signals I_T^* and I_M^* are processed through the overlay current control loops (Fig. 3), and the output current signals in the synchronously rotating reference frame are then vector rotated to transform into stationary frame phase current commands for the inverter current-controller.

All the essential feedback signals for the control system as shown in the feedback signal processing block are estimated with precision. These signals include torque (T_e), stator flux (Ψ_s), torque angle (cos δ, sin δ), rotor position (cos θ_e, sin θ_e), and rotor temperature for magnet flux compensation. The detailed description of feedback signal processing can be found in [1]. Basically, the d^e and q^e components of stator flux are described respectively as a function of magnet flux with the stator d^e-q^e currents and the stator d^e-q^e currents. The relations are derived by extensive modeling and laboratory calibration where parameter saturation and cross-coupling effects have been taken into consideration. The torque and torque angle are estimated by the equations

$$T_e = K_t[\psi'_{ds} i_{qs} - \psi_{qs} i_{ds}] \tag{1}$$

$$\sin \delta = \frac{\psi_{qs}}{\sqrt{\psi'^2_{ds} + \psi^2_{qs}}} \tag{2}$$

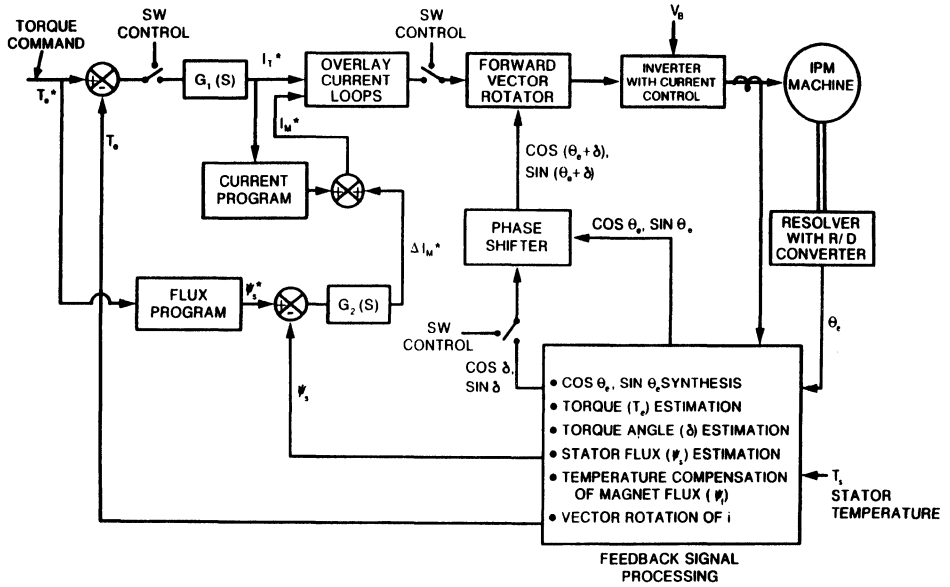


Fig. 2. Simplified control block diagram of the drive system in constant torque region (PWM mode).

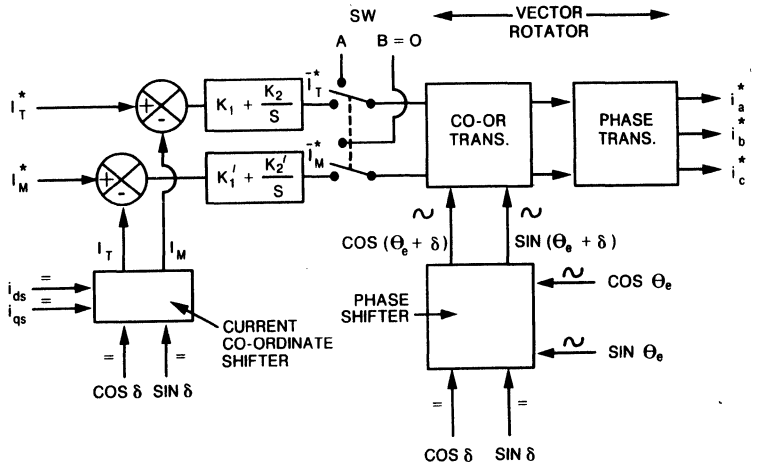


Fig. 3. Overlay active and reactive current control loops with forward vector rotation.

$$\cos\delta = \frac{\Psi'_{ds}}{\sqrt{\Psi'^2_{ds} + \Psi^2_{qs}}} \tag{3}$$

where

Ψ'_{ds}, Ψ_{qs} are the stator d^e-q^e flux components (rotating frame) and i_{ds}, i_{qs} stator d^e-q^e current components (rotating frame).

It was mentioned before that the NeFeB magnet flux has some negative temperature sensitivity that should be corrected with the help of magnet temperature information. A simplified single time constant dynamic thermal model is solved to compute the magnet temperature approximately from the stator temperature.

Fig. 3 shows the overlay active and reactive current control loops with forward vector rotation. These loops permit vector control to be effective in partial saturation of the current-controller (quasi-PWM) and help smooth transition between the PWM and square-wave modes. The operation of the loops

can be considered as redundant in normal PWM operation. A small amount of coupling is introduced [5] in vector control by the loops that tends to slow down the response, but this can be ignored because of high loop gains. The current coordinate shifter converts the d^e-q^e current components to I_M and I_T by the relations

$$I_M = i_{qs} \cos \delta - i_{ds} \sin \delta \tag{4}$$

$$I_T = i_{qs} \sin \delta + i_{ds} \cos \delta. \tag{5}$$

The loops compare the respective command and feedback currents and generate outputs through the PI compensators. The PI control assures matching of command and feedback currents as long as current control remains effective. The loop outputs $\bar{I}_T^*$ and $\bar{I}_M^*$ are vector rotated by the unit vector signals cos $(\theta_e + \delta)$ and sin $(\theta_e + \delta)$ such that $\bar{I}_T^*$ and $\bar{I}_M^*$ are aligned to phase voltage V_s and stator flux Ψ_s, respectively. In normal PWM mode, $\bar{I}_T^*$ signals remain identical to the respective command signals. But, as speed increases in the constant-torque region, the current-controller enters into the quasi-PWM mode due to increasing counter emf. With this condition, the loop outputs become higher than the respective command inputs while assuring matching between the command and feedback signals. As speed increases, the number of chops in the current-controller decreases and eventually at square-wave output voltage, the loop outputs $\bar{I}_T^*$ and $\bar{I}_M^*$ saturate to the clamped values A and B, respectively. Then, the control of the overlay loops is completely lost and the switch is thrown to the "SW" position, as shown. The drive system then enters into the constant-power region with square-wave impressed voltage and the control block diagram shown in Fig. 4 becomes valid. It should be mentioned here that efficiency consideration dictated that the drive system should operate in square-wave mode in the constant power region; otherwise, vector control, which gives better transient response (but demands PWM operation mode), could have been implemented. The structure change in Fig. 4 for the PWM control mode is shown by the two switches. The torque loop error generates the sin δ^* command through a PI compensator, which is then converted into the torque angle command δ^* through a look-up table. The δ^* angle is then added with the rotor position angle θ_e to generate the unit vector signals cos $(\theta_e + \delta)$ and sin $(\theta_e + \delta)$. These signals permit phase shift angle (δ) control of the machine input voltage by the same vector rotator and current-controller as described before. In fact, the control principle is essentially the same as shown in Fig. 3 with the switch in "SW" position and considering cos δ and sin δ as the command signals. Since the magnitude of A is very high and $B = 0$, the vector rotated signals can be expressed as

$$i_a^* = V_{a0}^* = A \cos (\theta_e + \delta^*) \tag{6}$$

$$i_b^* = V_{b0}^* = A \cos (\theta_e + \delta^* - 120°) \tag{7}$$

$$i_c^* = V_{c0}^* = A \cos (\theta_e + \delta^* + 120°) \tag{8}$$

where V_{a0}^*, V_{b0}^*, and V_{c0}^* are the respective phase voltage commands with respect to the hypothetical battery center point.

Note that the steep sides of the current commands will force the current-controller to switch only at the edges of the half-cycle, thus generating square voltage waves. The above equations indicate that the applied phase voltage will be aligned at an angle δ with the respective induced voltage. Fig. 4 indicates an alternate δ^* angle control loop where the three-phase square command current waves are fabricated directly from the $\theta_e + \delta^*$ angle. This control is implemented at higher speed (forward direction only) where small computation sampling time is needed for the desired torque resolution. Although flux control loop is inactive in the square-wave mode, the loop error, as shown, helps in the transition to the PWM mode, which is explained later.

The transition between PWM and square-wave modes is required to be fast and smooth under all conditions of operation of the drive system. The transition performance is especially demanding if it overlaps with gear shifting. An UP or DOWN shifting request placed independently by the higher level vehicle control computer will cause a fast speed change in the machine and therefore the control response should be fast compared to the rate of speed change. The transition is designed such that if gear shifting is requested during transition, it will be inhibited until transition is completed successfully. However, transition should be successful if initiated during gear shifting. Fig. 5 shows the sequence diagram for the transition, which also indicates the criteria for transitions and the corresponding actions. The transition from the PWM to square-wave mode is initiated when the current-controller is near saturation that is indicated by the transistor base drive pulse transistion counts in two successive fundamental frequency cycles. As this condition is detected, sin δ^* control is activated with the initial value updated by computation and then the switch in Fig. 3 is transferred to the "SW" position. For successful operation, the control requires that the polarity of A is to be sensitive to the direction of machine rotation ($+A$ for forward rotation). Once the system is transitioned to the square-wave mode, a delay time is added to settle the transients and then the look-up table control method is activated (in forward direction only). The criterion for the square-wave to PWM mode transition is determined by the flux loop error as indicated in Fig. 4. As the error decreases and eventually becomes negative, the PWM mode is activated by enabling the overlay currents and flux control loops. Note that a transition may occur at constant torque due to speed variation, at constant speed due to torque variation, or due to battery voltage variation at the same operating point on torque-speed plane.

III. Drive System Simulation

Computer-aided control system design tools are playing increasingly important roles in the design of power electronic and drive systems. These tools are becoming simple, economical to use, and more user-friendly day by day. A complex newly developed control system can be conveniently designed and simulated on a computer to verify the feasibility of the control laws. The control system design parameters can be

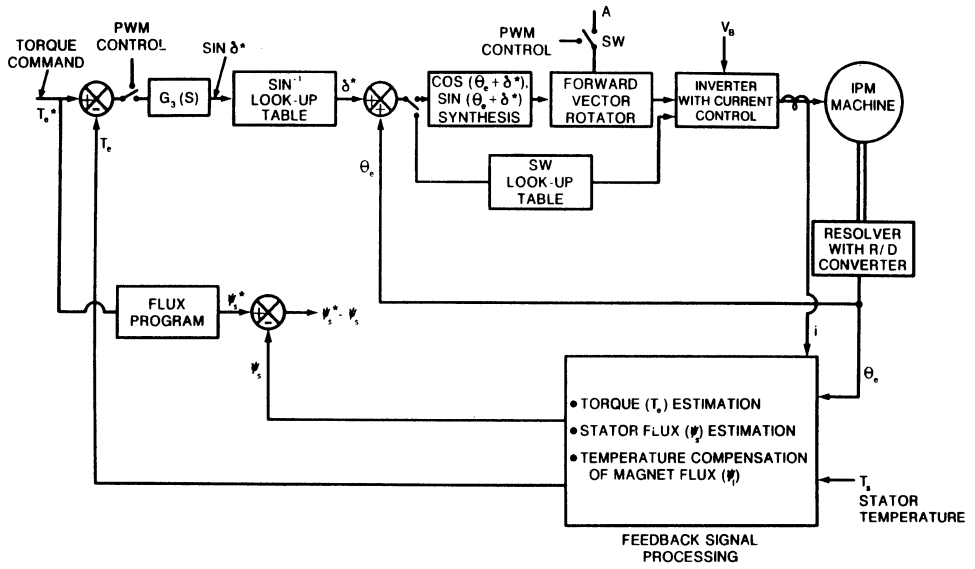


Fig. 4. Simplified control block diagram of the drive system in constant power region (square-wave mode).

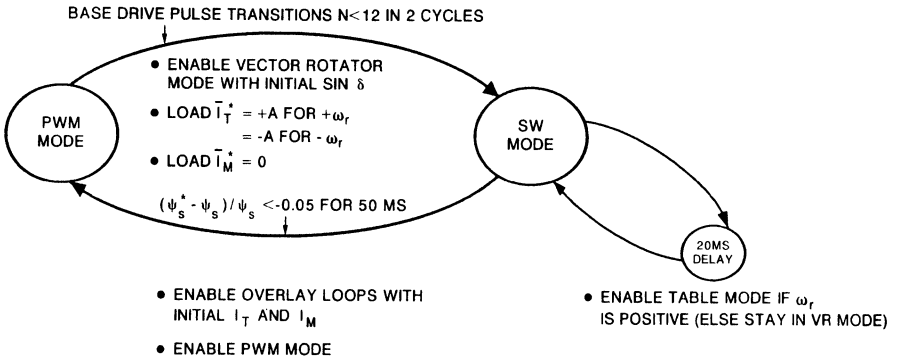


Fig. 5. Sequence diagram for PWM-square wave transitions.

iterated on simulation until the static and dynamic performances become optimal. Besides, the harmonics and the fault performance of the system can be studied in considerable detail. The simulation approach is often time-saving and economical and has less risk of damage than the trial-and-error method of breadboard design. However, it should be noted that simulation performance of a system can be only as good as its model description, and therefore, this approach should be considered for preliminary study of a system. An approximate model simulation with a breadboard test is usually the desirable approach because an accurate model description of a physical system is often very involved.

A. Review of the Simulation Language SIMNON

SIMNON is a popular simulation language among a number of computer-aided design tools that have been available recently [17], [19]. This language has been used in the present drive system simulation, and therefore its salient features will be briefly reviewed. SIMNON is a command driven interactive program for simulation of dynamical systems that can be described by linear/nonlinear ordinary differential and difference equations. The commands, for example, can change parameters of the model, perform simulation, graphically plot results on a terminal, and modify the model. With the macro

facility, the user can construct a command string. This compiler is included in the program and works in parallel with an editor. This enables the user to correct the erroneous lines of the program immediately.

For SIMNON simulation, a large system is normally resolved into a number of subprograms. These subprograms are then interconnected by input and output signals through a connecting routine. The SIMNON programs can be connected with specially formatted FORTRAN files. SIMNON offers a special advantage for a microcomputer-controlled system. Here, the physical process, which is normally a continuous system, can be modeled by differential equations, whereas the controller, which is a discrete time system, can be described by difference equations. All the system descriptions are in state space form. Table I shows the general structure of a SIMNON program for a continuous system. The structure of the discrete time system follows a similar pattern, and is illustrated later. The program starts with a heading that defines the type of system and gives a filename. The body of the program consists of three sections: declarations, initial section, and assignments, and then terminates with an END statement. The sequence of program statements is arbitrary and SIMNON automatically sorts them into proper order. The INPUT and OUTPUT statements indicate the signals that link with other programs. The TIME declaration is necessary if a time related statement appears in the program. The STATE and DER statements relate to state variables and their derivatives, respectively, of the state space equations, and must be declared in the same order. The SORT statement is required only if an INITIAL statement has been included and it acts as the terminator for the section. The assignment statements are FORTRAN-like and these include parameter and state initial values. This section may include standard functions, such as SIN(X), SQRT(X), ABS(X), etc. When multiple programs are interconnected by INPUT and OUTPUT signals, a connecting system of the following structure should be used:

CONNECTING SYSTEM <name>
Declarations
Connect section
END

For integration of state space equations in a continuous system, one of the following algorithms can be selected:

HAMPC	Hamming predictor corrector (default)
RK	Runge–Kutta variable step size
RKFIX	Runge–Kutta fixed step size
DAS	Integration routine for stiff systems

Once the SIMNON program for the entire system is written, a typical string of commands as follows can be exercised:

>SYST X Y Z	Compiles the system containing X, Y, Z files
>EDIT X	Changes the program
>STORE A B C	Stores the variables
>ALGOR <name>	Selects the algorithm
>SIMU O T	Simulates the system for interval T
>ASHOW A	Plots the stored variable A with automatic scaling

TABLE I
GENERAL STRUCTURE OF SIMNON PROGRAM

```
{CONTINUOUS SYSTEM      <system identifier>}
[INPUT  <simple variable>
[OUTPUT <simple variable>
[TIME   <simple variable>
[STATE  <simple variable>
[DER    <simple variable>

[INITIAL
Computation of initial values for state variables
Computation of parameters
SORT

[Computation of auxiliary variables]
[Computation of output variables]
[Computation of derivatives]
[Parameter assignments]
[Initial value assignments]
{END}
```

B. Drive System Simulation in SIMNON

The complete drive system including the inverter and the machine was simulated in the computer using the VAX-based SIMNON program. The purpose of simulation is to verify the complex control algorithms, design the controller parameters, and study the static and dynamic performances of the system before building the laboratory breadboard. In fact, once the initial simulation phase was completed, the iteration of simulation and laboratory tests went hand-in-hand whenever the test results were not up to expectation. It may be of interest to mention here that the simulation also included the study of dc link harmonics and fault performance of the inverter-machine system, but these aspects will not be described here.

Fig. 6 shows the simulation block diagram of the drive system where each functional block can be identified from Figs. 2 and 4. A SIMNON program is written for each functional block with the program name as indicated, and then all the blocks are interconnected with the I/O signals using the connecting system CON. The nature of the system (continuous or discrete time) is indicated in each block. The discrete time systems use the actual sampling times that are used for microcomputer implementation. Thus, the design of sampling times in multitasking microcomputer control could be verified by simulation. The PWM and square-wave control modes were simulated independently using the common program modules as indicated, i.e., the simulation does not incorporate the sequence diagram of Fig. 5. SIMNON has some limitations in looping and sequencing operations and therefore further study is needed to simulate the sequencing control. In Fig. 6, the basic simulation functions are

1) Controller transfer functions—converted to difference equations in state space form
2) Flux and current programs—described by segmented straight lines
3) Algebraic relations
4) Standard functions
5) Inverter—described by ideal on-off switches
6) Machine—described by differential equations in state space form

Table II illustrates the simulation program for the machine (the machine rotor has negligible damping and therefore the rotor equivalent circuits are considered open). It is developed in the format described in Table I. The comments in each

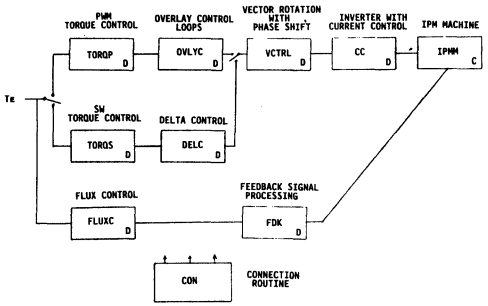


Fig. 6. Simulation block diagram of the drive system.

TABLE II
SIMULATION PROGRAM FOR THE MACHINE

```
CONTINUOUS SYSTEM IPMM
"IPM MACHINE MODEL IN SYNC. REF. FRAME(INERTIA LOAD)
INPUT VQSE VDSE    "MODULE INPUT SIGNALS
OUTPUT IA IB IC TE TEM WE X3 X4 IQS1 IDS1
TIME T                "IN SECONDS
STATE IQS IDS W TH
DER DIQS DIDS DW DTH    "DERIVATIVE OF STATES
"
DIQS =(WB/XQS)*(VQSE-RS*IQS-WE*XDS*IDS/WB-WE*EFF/WB)
DIDS = (WB/XDS)*(VDSE+WE*XQS*IQS/WB-RS*IDS)
FDSEP = IDS*XDS    "D-AXIS ARMATURE REACTION
FQSE = IQS*XQS
TE = (3/WB)*((FDSEP+EFF)*IQS-FQSE*IDS) "IN Nm.
DW = (2/J)*(TE-TL)   "SPEED EQUATION
DTH = W
WE = W
THE = MOD(TH,6.2831)   "ROTOR ANGLE, 0-360 DEG.
X3 = COS(THE)
X4 = SIN(THE)
IQSS = IQS*X3+IDS*X4   "STA. FRAME Q-CURRENT
IDSS = -IQS*X4+IDS*X3
IA = IQSS     "PHASE A CURRENT
IB = -(IDSS*SQRT(3)+IQSS)/2
IC = -IA-IB
TEM = TE*.738    "TORQUE IN LB. FT.
IQS1 = IQS    "ROTATING FRAME Q-CURRENT
IDS1 = IDS
WB:710.48     "BASE FREQUENCY (RAD./S.)
XQS:0.16      "MACHINE PARAMETERS AT WB
RS:0.00443
XDS:0.103
EFF:57.4      "MAG. FLUX AT WB (VOLTS)
J:1.2         " INERTIA
W:100         " INITIAL SPEED
TH:0          " INITIAL ANGLE
END
```

statement make the program self-explanatory. The program inputs the voltages (from the program CC—see Table III) to the synchronously rotating frame equivalent circuits and solves the stator currents using the following sets of equations:

Machine equations

$$v_{qs}=R_s i_{qs}+\left(\frac{\omega_e}{\omega_b}\right)X_{ds}i_{ds}+\left(\frac{\omega_e}{\omega_b}\right)V_{f0}+\frac{X_{qs}}{\omega_b}\frac{di_{qs}}{dt} \quad (9)$$

$$v_{ds}=R_s i_{ds}+\left(\frac{\omega_e}{\omega_b}\right)X_{qs}i_{qs}+\frac{X_{ds}}{\omega_b}\frac{di_{ds}}{dt} \quad (10)$$

$$T_e=\frac{3}{\omega_b}\left[(i_{ds}X_{ds}+V_{f0})i_{qs}-i_{qs}X_{qs}i_{ds}\right] \quad (11)$$

$$T_e-T_L=J\left(\frac{2}{P}\right)\frac{d\omega_e}{dt} \quad (12)$$

$$\frac{d\omega_e}{dt}=\theta_e. \quad (13)$$

Vector rotation equations

$$i_{qs}^s=i_{qs}\cos\theta_e+i_{ds}\sin\theta_e \quad (14)$$

$$i_{ds}^s=-i_{qs}\sin\theta_e+i_{ds}\cos\theta_e \quad (15)$$

$$i_a=i_{qs}^s \quad (16)$$

$$i_b=-\frac{\sqrt{3}}{2}i_{ds}^s+\frac{1}{2}i_{qs}^s \quad (17)$$

$$i_c=-i_a-i_b \quad (18)$$

where

V_{f0} is the machine induced voltage at base speed ω_b (in radians per second)

P is the number of poles

All other quantities are given in standard notation [3]. The machine parameters are given in the lower part of the table.

Table III illustrates the simulation program for the current controller (CC), which is described as a discrete time system with a sampling time of 0.1 ms. In laboratory breadboard, the hysteresis-band current-controller has been designed by using dedicated hardware. The format of a discrete time system is similar to that of a continuous system except that the statements STATE, NEW, TSAMP and TS characterize the description of difference equations. In the program, the command currents IAC, IBC, and ICC are compared with the feedback currents IA, IB, and IC, respectively to generate the current loop errors as shown. The state of the inverter switches is generated by comparing the current error with the hysteresis band HB. The inverter output voltages in the rotating frame are then generated by the following equations [3]:

$$v_{as}=V_B\cdot \mathrm{NA} \quad (19)$$

$$v_{bs}=V_B\cdot \mathrm{NB} \quad (20)$$

$$v_{cs}=V_B\cdot \mathrm{NC} \quad (21)$$

$$v_{qs}^s=\frac{2}{3}v_{as}-\frac{1}{3}v_{bs}-\frac{1}{3}v_{cs} \quad (22)$$

$$v_{ds}^s=-\frac{1}{\sqrt{3}}v_{bs}+\frac{1}{\sqrt{3}}v_{cs} \quad (23)$$

$$v_{qs}=v_{qs}^s\cos\theta_e-v_{ds}^s\sin\theta_e \quad (24)$$

$$v_{ds}=v_{qs}^s\sin\theta_e+v_{ds}^s\cos\theta_e \quad (25)$$

where

V_B is the Battery voltage

NA, NB, NC are the new states of the inverter phase legs and all other variables are in standard symbols. The inverter starts with the initial state shown in the table.

The simulation program of the whole drive system was built

TABLE III
SIMULATION PROGRAM FOR THE CURRENT CONTROLLER

```
DISCRETE SYSTEM CC
"HYSTERESIS-BAND CURRENT-CONTROLLED PWM INVERTER
INPUT VB IAC IBC ICC IA IB IC X3 X4
OUTPUT VQSE VDSE
TIME T          "IN SECONDS
STATE A B C     "STATE OF A PHASE LEG: 1 OR 0
NEW NA NB NC    "NEW STATE
TSAMP TS        "SAMPLING INSTANT
"
TS = T+0.1E-3
IAE = IAC-IA    "CURRENT LOOP ERROR
IBE = IBC-IB
ICE = ICC-IC
NA = IF IAE>HB THEN 1 ELSE IF IAE<-HB THEN 0 ELSE A
NB = IF IBE>HB THEN 1 ELSE IF IBE<-HB THEN 0 ELSE B
NC = IF ICE>HB THEN 1 ELSE IF ICE<-HB THEN  0 ELSE C
VQSS = VB*(NA*2-NB-NC)/3  "STA. FRAME Q-VOLTAGE
VDSS = VB*(NC-NB)/SQRT(3)
VQSE = VQSS*X3-VDSS*X4   "ROTATING FRAME Q-VOLTAGE
VDSE = VQSS*X4+VDSS*X3
HB:30                    "HYSTERESIS BAND
A:1   "INITIAL STATE DEFINATION
B:1
C:0
END
```

up step by step starting with the inner core drive elements. Fig. 7 shows the typical simulation command and feedback current waves in the PWM mode. The large sampling time of the simulation often causes the current to exceed the 20-A band of the command wave as evident in the figure. Fig. 8 shows the typical close loop torque response in PWM mode with a 25 lb ft step command. The ripple in the estimated torque was found to be higher than that of the shaft output.

IV. MICROCOMPUTER CONTROL

Since the advent of microcomputers in the early 1970's, the technology has gone through a dynamic evolution in the last one and a half decades. Microcomputers are available today with large word-size, high computation speed, and large functional integration, and this trend will continue in the future. Super microcomputers, based on the same principle as the super computer (such as CRAY 2) where parallel processors add to the processing speed, look very promising and will add tremendous capability for real time control of systems in the future. The control system under consideration uses state-of-the-art microcomputers and their hardware and software design features are described as follows:

A. Hardware Design

The microcomputer-based control hardware uses two Texas Instruments TMS32010 digital signal processors (DSP) and one Intel-8097 (generic name 8096) microcontroller. The key features of these devices are given in Tables IV and V, respectively. Both are 16-bit high-performance microcomputers and are ideally suitable for real time control applications. The 16 × 16-bit dedicated parallel multiplier on the DSP chip that multiples in 200 ns permits very time-critical I/O signal processing (including vector rotation) in the drive control system. The TMS32010 DSP chip was selected over the alternate DSP chips based on performance benchmarks, military spec. availability, and excellent hardware/software development support. Although the DSP chips are extremely fast and allow software implementation for high-speed control functions, they do not provide general purpose hardware interfaces that allow simple connections to standard I/O

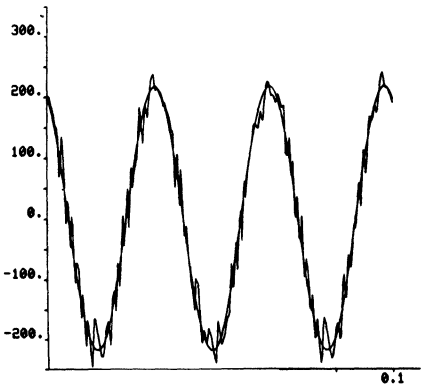


Fig. 7. Typical command and feedback current waves.

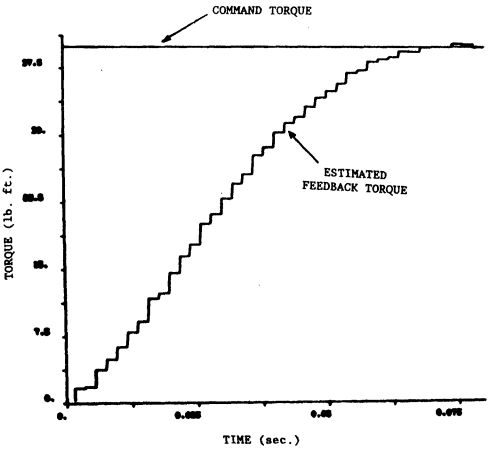


Fig. 8. Close loop torque response in PWM mode.

TABLE IV
KEY FEATURES OF THE DIGITAL SIGNAL PROCESSOR TMS32010

- 160-ns instruction cycle
- 144-word on-chip data RAM
- ROMless version—TMS32010
- 1.5K-word on-chip program ROM-TMS320M10
- External memory expansion to a total of 4K words at full speed
- 16-bit instruction/data word
- 32-bit ALU/accumulator
- 16 × 16-bit multiply in 160-ns
- 0 to 15-bit barrel shifter
- Eight input and eight output channels
- 16-bit bidirectional data bus with 50-megabits-per-second transfer rate
- Interrupt with full context save
- Signed two's complement fixed-point arithmetic
- NMOS technology
- Single 5-V supply
- Two versions available
  - TMS32010-20 . . . 20.5 MHz Clock
  - TMS32010-25 . . . 25.0 MHz Clock

TABLE V
KEY FEATURES OF THE INTEL 8096 MICROCOMPUTER

- 8K-byte on-chip ROM
- 232-byte register space (RAM)
- 10-bit, eight-channel A/D converter
- Five 8-bit I/O ports
- Full-duplex serial port
- High-speed pulse I/O
- Pulse-width-modulated output
- Eight interrupt sources
- Four 16-bit software timers and two 16-bit hardware timers
- Watchdog timer
- Hardware signed and unsigned multiply/divide

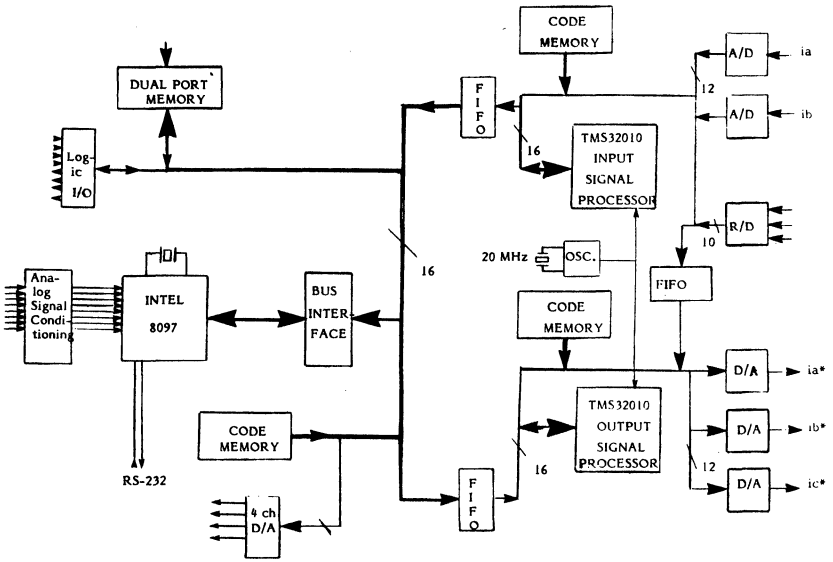


Fig. 9. Simplified block diagram of controller hardware.

devices. Furthermore, implementation of functions that do not require high-speed processing becomes cumbersome because of small stack size and limited program and data memory spaces. The Intel-8097 microcontroller, which incorporates the bulk of control functions, overcomes the above problems. Besides an expansive instruction set, it has a high level of functional integration.

Fig. 9 shows the simplified controller hardware architecture. The two signal processors have the same core hardware design and each is tailored to its specific tasks via the respective I/O devices. The input signal processor (ISP) is interfaced to A/D converters for acquiring the machine current signals, whereas the output signal processor (OSP) has D/A converters to supply reference current waves to the current-controller. The resolver-to-digital (R/D) converter provides 10-bit (0.352° resolution) shaft angle (θ_e) information up to the maximum tracking rate of 20 400 rpm. All interprocessor communications are accomplished with 16-bit wide 16-location FIFO (first-in-first-out) registers. A key goal in the DSP-based I/O hardware design is to use the full potential of the processors by minimizing the software overhead required to perform I/O.

The Intel-8097 consists of a powerful CPU tightly coupled with program and data memory along with several I/O features all integrated into a single chip. The 8097 chip incorporates a 10-bit unipolar (0–5 V) A/D converter and an 8-channel analog multiplexer on the same chip. This converter is used for acquisition of signals required for drive system sequencing and in-line monitoring functions.

B. Software Design

The distribution of control functions among the three microcomputers and their processing rates were determined by system analysis. The processing rate, i.e., the sampling time interval of each task, was verified by SIMNON simulation. Obviously, the control functions that require high sampling rates (5–30 KHz) are executed by the signal processors whereas the less time-critical functions are executed in the

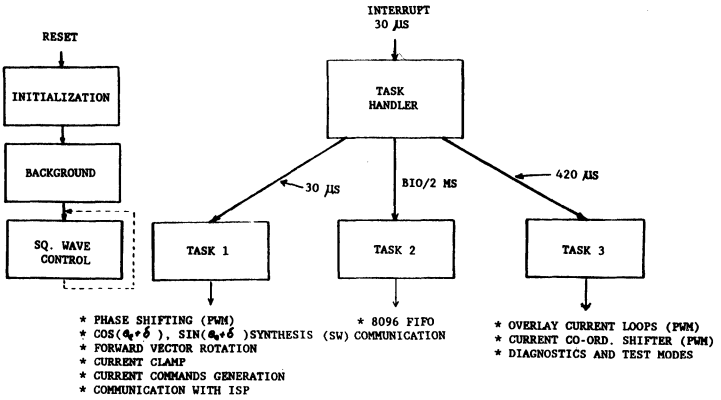


Fig. 10. Simplified structure of TMS32010 software (output DSP).

8097 microcomputer. The high throughput capability of the DSP's is utilized for performing transformations from rotating to stationary reference frame and vice-versa, regulation of the active and reactive currents in the PWM mode, and generation of the transistor switching commands in the square-wave mode. The table look-up capability of the TMS320 permits easy synthesis of sin θ_e and cos θ_e functions from the input θ_e signal. The ability to make decisions in software to reconfigure the control schemes in real time provides a great advantage over a dedicated hardware approach. Additionally, diagnostic functions are incorporated to ease the development process.

Software for all the three microcomputers is written in assembly language (ASM-96, XASM) using scaled integer arithmetic. An Intel Series IV development system and an Intel SBE-96 board were used for the development of the 8096 software system. The software for the TMS32010 signal processing systems was developed with the VAX-based cross-assembler and bench tested with the TI TMS32010 simulator. Real time testing of the DSP software was performed on the TI EVM-32010 evaluation module boards.

A simplified structure chart of the output DSP is shown in Fig. 10 that also indicates the key functions under each task and the task processing intervals. The interrupt input to the signal processor is connected to a 30-μs pulse train that serves to set the basic sampling rate (33 KHz). Upon receipt of the interrupt, the return address of the interrupted program is saved on the processor stack, interrupts are disabled, and control is passed to the task handler. Since the TMS32010 stack is only four words deep, another logic stack in data RAM is utilized to save the status of key registers. The TASK 1 (30 μs) functions are executed and a counter to detect if TASK 3 is ready is decremented. The state of the BIO input is polled to determine if the 8096 processor is requesting an interaction. If so the interrupt is enabled and TASK 2 is started. TASK 2 either loads (ISP) or unloads (OSP) the FIFO registers that are interfaced to the 8096 computer system and serves to synchronize the inter-processor communications. If no interaction is requested or when TASK 2 is complete, the TASK 3 counter is tested to determine if TASK 3 should be executed. Finally, whether or not TASK 3 is run, the status of the registers is restored and execution of the interrupted program is resumed. Task priorities are established by the sequence in which the task handler schedules them (TASK 1 highest --> TASK 3 lowest). A fourth lowest priority task called BACKGROUND task that never finishes (loops) serves to occupy the CPU when all the essential tasks are completed.

The ISP software structure is similar to Fig. 10 and both signal processors share the same operating system design, except that the OSP is structured differently depending on the mode of control. In other words, the ISP executes the same set of software routines from power-up to power-down, whereas the OSP software is configured by the 8096 to allow operation in and transitions between the PWM and square-wave modes of control. The two DSP's also share a common diagnostic I/O routine and data RAM initialization scheme.

The 8096 computer system is primarily responsible for estimating and regulating the torque and flux of the IPM machine. Inputs to the estimators are obtained from the input signal processor and outputs from the regulators are transformed into three-phase current references by the output signal processor. Additional 8096 microcomputer system functions include: vehicle control microcomputer interface, start-up/shut-down sequencing, in-line monitoring functions, PWM <---> square-wave mode transitions, and diagnostics.

An operating system similar to the one used for the DSP systems serves to schedule the tasks at fixed sampling rates. The 8096 software timer interrupt is programmed to generate the basic 2-ms clock ticks and software counters are maintained in RAM to generate the additional sampling intervals. The 8096 architecture differs from most computers in that there are no general purpose registers; any internal RAM location can serve as the operand in instructions. In order to maintain compatibility with the PL/M-96 language and provide a more conventional environment, four 16-bit RAM locations are defined as working registers. The statuses of

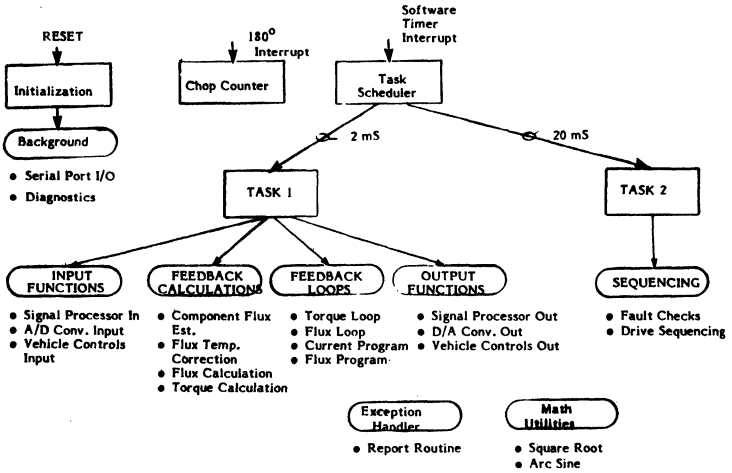


Fig. 11. Simplified structure of Intel-8096 software.

these registers are preserved and restored every time a task is performed. Once again task busy flags are used to prevent stack overrun.

Fig. 11 shows the simplified structure chart of 8097 software, which also indicates the functions under various tasks and task sampling intervals. Every 2 ms a new set of inputs is obtained (input functions) and used for the computation of the machine fluxes and torque. These estimations are regulated to match the commanded values and the data are output (output functions). Sequencing logic serves to select the appropriate control algorithm for either the PWM or square-wave mode and open loop modes are incorporated for debugging. The output routines, feedback calculations, and output functions remain the same in all modes of control. The software also permits various feedback loop configurations so that the system can be debugged systematically starting with the core drive elements. The configurations can be summarized as follows:

Mode 0 Open all the loops with $\delta = 0$
Mode 1 Open all the loops and release δ
Mode 2 Close overlay current loops and initialize torque loop integrator
Mode 3 Close torque loop and use current program
Mode 4 Close torque loop and get loop gains from A/D channels
Mode 5 Initialize flux loop integrator
Mode 6 Close all PWM loops
Mode 7 Open vector rotator square-wave mode loop
Mode 8 Open table look-up square-wave mode loop
Mode 9 Close vector rotator square-wave mode
Mode 10 Close table look-up square-wave mode loop
Mode 11 Close all PWM modes and evaluate transition to square-wave
Mode 12 PWM --> square-wave mode transition
Mode 13 Square-wave --> PWM transition

V. Drive System Tests

The complete drive system with the microcomputer controller was thoroughly tested in laboratory on a dynamometer and performances were found to be excellent. The test also showed general correlation with the simulation results. The 70-hp 4-pole star-connected IPM machine under test was custom designed using an NeFeB (Crumax 30A) magnet in segments. The key machine parameters are included in the simulation program shown in Table II. The machine has a base or corner-point speed of 3394 rpm, crossover speed (the speed at which the machine counter emf balances the fundamental frequency square-wave voltage) of 5044 rpm, and a maximum speed of 13 750 rpm. The battery voltage varied from 135 to 265 V corresponding to worst case motoring and regeneration, respectively. The inverter consists of three phase-leg modules where each Darlington transistor was rated for 500 A, 500 V. A Darlington transistor again consists of three component matched transistors in parallel of 200-A rating. The dynamometer used for the tests could be operated in constant (but programmable) speed or inertia mode. The test set-up includes a computer-based data acquisition and analysis system [18], where steady state waveforms can be captured and drive performances, such as efficiencies, power factor, various losses, etc., can be calculated and displayed on a video terminal.

Once the drive system was simulated successfully and the controller hardware and software were debugged, the system was ready for extensive laboratory tests on the dynamometer. A careful test procedure was formulated so that the task becomes smooth and time efficient. The microcomputer-controller permitted various test modes where, starting with the inner core drive system, the outer loops could be added in steps and thoroughly tested. Initially, all the tests were performed on the dynamometer in constant speed mode, then

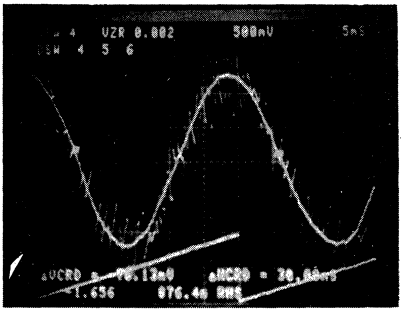


Fig. 12. Waves at PWM forward motoring mode (Te = 25 lb ft, 1000 rpm). Top: Command and feedback currents (50 A/d). Bottom: Rotor position (180° A/d).

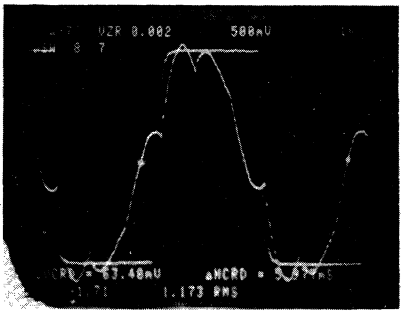


Fig. 13. Phase voltage and current waves at square wave forward motoring mode (Te = 25 lb ft, 5004 rpm, 50 A/d, V = 166 V).

the inertia mode was exercised, and finally the transitions as shown in Fig. 5 were tested.

Fig. 12 shows the typical command and feedback phase current waves (top) in the PWM mode when the dynamometer was operating at constant speed. The rotor position (θ_e) obtained from the R/D converter is also shown at the bottom. The $\theta_e = 0$ corresponds to alignment of the magnet north pole with the stator phase a-axis. The figure indicates that the current phasor leads the magnet flux by an obtuse angle. Fig. 13 shows the typical phase voltage (with respect to the battery center-tap) and phase current waves in square-wave mode. The current slightly leads the voltage wave, and the inverter switching at each edge of the square-wave is evident. As the speed increases at constant torque, the phase lead increases because of increasing machine counter emf. Fig. 14 shows the four-quadrant operation of the drive system with the dynamometer in inertia mode. The system starts at zero speed in the forward direction with a constant motoring torque as shown. As the speed increases beyond a critical value, transition occurs smoothly from PWM to square-wave mode. As the torque command is reversed, the drive system enters into regeneration with immediate transition to PWM mode because of increase of the battery voltage. The system then goes through zero speed and eventually speed builds up in the reverse direction. The performance in the reverse direction is essentially symmetrical to that in the forward direction.

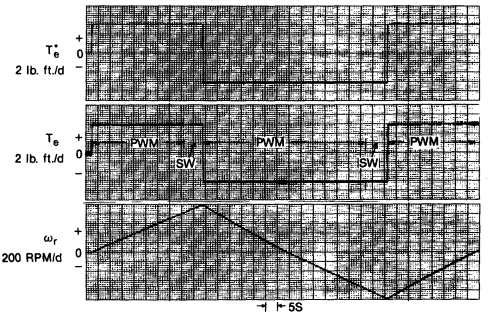


Fig. 14. Four-quadrant operation of the drive system.

VI. Conclusion

An advanced digital control of a drive system that uses an interior magnet synchronous machine with an Neodymium–Iron–Boron permanent magnet has been described. The drive system operated with full performance in the constant-torque region as well as in the field-weakening constant-power region. The drive system has been designed with close loop torque control for electric vehicle application, but the control can be easily extended for other industrial applications also.

The drive system has been extensively simulated using the VAX-based simulation language SIMNON. The salient features of SIMNON have been reviewed, and then the drive simulation has been described. A simulation study of the complex drive system was found to be extremely valuable to verify feasibility of the control laws and to design the controller parameters.

The drive system uses a distributed microcomputer-based control system where state-of-the-art Intel-8096 microcontroller and Texas Instruments TMS32010 digital signal processors are used. The 8096 is essentially responsible for feedback control and signals estimation functions, whereas 32010 processors perform the time-critical I/O signal processing functions. The hardware and software design features of the controller have been discussed.

A 70-hp inverter-fed drive system has been extensively tested in the laboratory with the help of a dynamometer, and experimental results show good correlation with the simulation results. The test results, including four-quadrant operation on a dynamometer that was programmed in the inertia mode, have been discussed. The performance in transition between the PWM and square-wave modes with and without gear shifting was found to be excellent. The results of this study will help to promote IPM synchronous machine drives for various industrial applications in the future.

Acknowledgment

The authors are grateful to R. D. King for his help during the experimentation phase of the project.

References

[1] B. K. Bose, "A high performance inverter-fed drive system of an

interior permanent magnet synchronous machine," in *Conf. Rec. IEEE/IAS Annu. Meeting*, pp. 269–276, 1987.
[2] T. Nakano, H. Ohsawa, and K. Endoh, "A high performance cycloconverter fed synchronous machine drive system," in *Conf. Rec. IEEE/IAS Int. Sem. Power Converter Conf.*, pp. 334–341, 1982.
[3] B. K. Bose, *Power Electronics and AC Drives.* Prentice-Hall, Englewood Cliffs, N.J. 1986.
[4] B. K. Bose, Ed., *Microcomputer Control of Power Electronics and Drives.* New York: IEEE Press, 1987.
[5] K. Hasse, "Control of cycloconverters for feeding asynchronous machines," in *Proc. IFAC Symp. Control Power Electronics and Elec. Drives,* pp. 537–545, 1977.
[6] K. J. Åström, *A SIMNON Tutorial*, Dept. Automatic Control, Lund Institute of Technology, 1982.
[7] K. J. Åström and B. Wittenmark, *Computer Controlled Systems.* Englewood Cliffs, NJ: Prentice-Hall, 1984.
[8] H. A. Spang, "The federated computer-aided control design system," *Proc. IEEE*, vol. 72, pp. 1724–1731, Dec. 1984.
[9] H. Elmqvist, "SIMNON: an interactive simultation program for nonlinear systems," *Proc. Simulation*, 1977.
[10] D. K. Frederick, *SIMNON Reference Manual*, General Electric Co., Schenectady, NY, 1982.
[11] B. K. Bose, "Sliding mode control of induction motors," in *IEEE/IAS Annu. Meeting Conf. Rec.*, pp. 479–486, 1985.
[12] P. Katz, *Digital Control Using Microprocessor.* Englewood Cliffs, NJ: Prentice-Hall, 1981.
[13] *Intel Microcontroller Handbook*, 1984.
[14] *TMS32010 User's Guide*, Texas Instruments, 1983.
[15] B. K. Bose, "Technology trends in microcomputer control of electrical machines," *IEEE Trans. Ind. Electron.*, vol. 35, pp. 160–177, Feb. 1988.
[16] K. H. Bayer, H. Waldmann, and M. Weizbelzahl, "Field-oriented close-loop control of a synchronous machine with the new transvector control system," *Siemens Review*, vol. 39, pp. 220–223, 1972.
[17] D. K. Frederick, "Computer packages for the simulation and design of control systems," Arab School on Science and Technology, 4th Summer Session, Blouden, Syria, Sept. 1981.
[18] A. B. Plunkett, G. B. Kliman, and M. J. Boyle, "Digital techniques in the evaluation of high-efficiency induction motors for inverter drives," *IEEE Trans. Ind. Appl.*, vol. IA-21, pp. 456–463, Mar./Apr. 1985.
[19] D. K. Frederick and C. J. Herget, "The extended list of control software," *ELCS*, U.S. Edition, no. 2, June 1986.

DSP-BASED ADAPTIVE CONTROL OF A BRUSHLESS MOTOR

Nobuyuki Matsui and Hironori Ohashi

Department of Electrical and Computer Engineering
Nagoya Institute of Technology
Gokiso, Showa, Nagoya 466, JAPAN.

ABSTRACT

The paper presents the software control of the brushless DC motor with the parameter identification. Not only speed and current controls but a real time identification of the motor parameter can be implemented by software of the digital signal processor, TMS320C25.

The unique current control is performed according to an instantaneous voltage equation of the d-q model of the motor. In the system, the control accuracy depends on the motor parameters so the parameter identification in regard to armature inductance and emf constant is necessary. The identification algorithm has been verified by both simulations and experiments. The control program, including an parameter identification is of 2.5K words and the processing time is 99 μs.

INTRODUCTION

The inverter drive of AC motors has many advantages over the conventional DC motors and high performance drives have increased in popularity as AC servo motors. The required control characteristics are becoming higher so the introduction of modern control theories and high performance processors is positively tried to meet the requirements. Especially, by using the high performance processor, it is possible not only to implement the feedback or feedforward control but to realize the various compensating capabilities.

It is well known that the precise current control is the key technology to realize the high performance AC drives such as brushless motors and vector controlled induction motors. Consequently, the problem of obtaining precise current control has received much attention. It is requested that the motor current is always coincident with the sinusoidal current command under the steady state and transient conditions. In the existing current control using the voltage-fed inverter, the current hysteresis controlled PWM and the sub-harmonic PWM shown in Fig.1 have been widely used[1][3]. In the current hysteresis controlled PWM, the sinusoidal current is maintained within the hysteresis band but a voltage waveform is not necessarily desirable and the switching frequency of the inverter changes according to the operating condition of the motor. On the other hand, the sub-harmonic PWM has no problem associated with the voltage waveform and the switching frequency but the steady state phase lag is the problem to the high frequency operation.

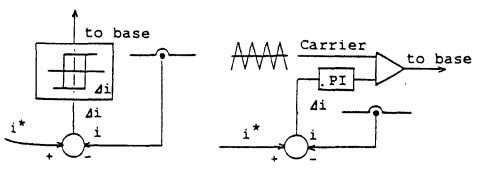


Fig.1. Conventional current controls. (a)current hysteresis controlled PWM, (b)sub-harmonic PWM.

In this paper, a new current control for brushless motors using the digital signal processor, TMS320C25, is presented. In the system, DSP performs not only the current control but the necessary control processing such as the rotor position sensing, the speed calculation and the calculation of the torque command through the speed control loop. The current control is performed by selecting PWM patterns for the inverter according to the on line calculation of the ideal voltage. The calculation is based on the d-q axis voltage equation of the brushless motor. Two PWM strategies are explained and compared. This control leads to good coincidence of the motor current with the current command under steady state and transient conditions.

As the control presented in this paper is based on the voltage equation of the motor, the control error depends on the parameters used in the controller. The dependency of the current control error on the parameters is investigated and the identification using the reference model is explained. The simulations and experiments show the effectiveness of the proposed identification.

CURRENT CONTROL ALGORITHM[1]

Fig.2 shows the well known equivalent d-q axis model of the brushless motor. The voltage equation is obtained as follows;

$$\mathbb{V} = (R + pL) \mathbb{I} + \mathbb{E} \quad \text{----------(1)}$$

In equation (1), $\mathbb{V}$, $\mathbb{I}$ and $\mathbb{E}$ represent, respectively, the applied voltage, current and the induced emf vectors which are defined by the following relations.

$$\mathbb{V} = [\ vd,\ vq\ ]^T,\quad \mathbb{I} = [\ id,\ iq\ ]^T \quad \text{----(2)}$$

$$\mathbb{E} = \dot{\theta}\ [\ L\ iq,\ Ke - L\ id\ ]^T \quad \text{----------(3)}$$

In equation (1), R and L are the armature resistance and inductance and Ke(=Mif) is the emf constant, respectively. These parameters

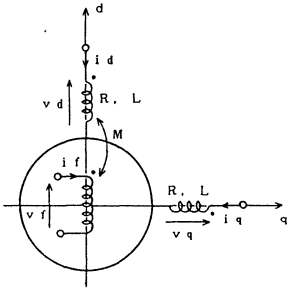


Fig.2. Equivalent d-q model of brushless motor

are reduced to the equivalent d-q axis model. To perform the software control, the difference equation corresponding to equation (1) is necessary. The instantaneous applied voltage, current and induced emf are approximated to the corresponding average values during the sampling interval, that is;

$$\mathbb{V} = \bar{\mathbb{V}}(n), \quad \mathbb{I} = \bar{\mathbb{I}}(n), \quad \mathbb{E} = \bar{\mathbb{E}}(n) \quad \text{-----(4)}$$

and the derivative term in equation (1) is approximated to the term given below:

$$p\mathbb{I} = [\, \mathbb{I}(n+1) - \mathbb{I}(n) \,] / T \quad \text{-------(5)}$$

where T is the sampling period. Using these approximations, the difference equations below are obtained.

$$\bar{\mathbb{V}}(n) = R\,\bar{\mathbb{I}}(n) + (L/T)\,[\, \mathbb{I}(n+1) - \mathbb{I}(n) \,] + \bar{\mathbb{E}}(n) \quad \text{----------(6)}$$

$$\bar{\mathbb{E}}(n) = \dot{\theta}(n)\,[\, L\,\bar{i}q(n),\ Ke - L\,\bar{i}d(n) \,]^T \quad \text{----------(7)}$$

In Fig.2, it is noted that the torque of the brushless motor is proportional to the q-axis current and the d-axis current does not contribute to the production of torque. Therefore, the d-axis current is controlled to be zero. Now, the ideal voltage $\bar{\mathbb{V}}^*(n)$ is defined. This voltage is applied between the n-th and n+1-th samplings to makes the motor current $\mathbb{I}(n)$ equal to the current command $\mathbb{I}^*(n+1)$ at the next sampling. Replacing $\mathbb{I}(n+1)$ in equation (6) with the current command $\mathbb{I}^*(n+1)$, the equation for the ideal voltage is obtained below.

$$\bar{\mathbb{V}}^*(n) = R\,\bar{\mathbb{I}}(n) + (L/T)\,[\, \mathbb{I}^*(n+1) - \mathbb{I}(n) \,] + \bar{\mathbb{E}}(n) \quad \text{----------(8)}$$

$$\bar{\mathbb{E}}(n) = \dot{\theta}(n)\,[\, L\,\bar{i}q(n),\ Ke \,]^T \quad \text{---------(9)}$$

The current at the n-th sampling in equation (8) can be predicted by voltage and current prior to the n-th sampling using equations (6) and (7). Inserting this prediction into equations (8) and (9), together with the approximate relations below;

$$R\,\bar{\mathbb{I}}(n) \fallingdotseq R\,\bar{\mathbb{I}}(n-1) \fallingdotseq R\,\mathbb{I}(n-1) \quad \text{-------(10)}$$

$$(1/2)\,[\, \bar{\mathbb{E}}(n) + \bar{\mathbb{E}}(n-1) \,] \fallingdotseq \mathbb{E}(n) \quad \text{----(11)}$$

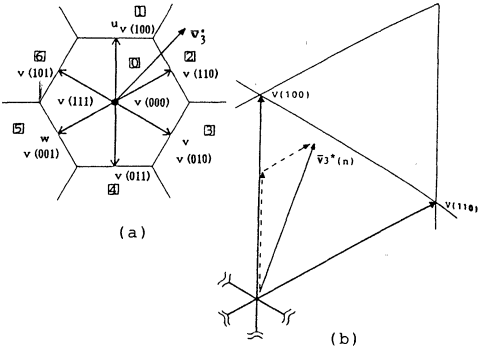


Fig.3. Two PWM strategies. (a)vector selection PWM, (b)average voltage PWM.

the ideal applied voltage is, therefore, given by the following equations.

$$\bar{\mathbb{V}}^*(n) = 2\,R\,\mathbb{I}(n-1) - \bar{\mathbb{V}}(n-1) + 2\,\mathbb{E}(n) + (L/T)\,[\, \mathbb{I}^*(n+1) - \mathbb{I}(n-1) \,] \quad \text{(12)}$$

$$\mathbb{E}(n) = \dot{\theta}(n-1)\,[\, L\,iq(n-1) + T\,[\, \bar{v}q(n-1) - Ke\,\dot{\theta}(n-1) - R\,iq(n-1)\,],\ Ke \,]^T \quad \text{--(13)}$$

These equations show that the ideal applied voltage between the n-th and n+1-th samplings can be calculated using the voltage and current prior to the n-th sampling. This means that the ideal applied voltage can be obtained by on line calculations before the n-th sampling and, therefore, it can be applied by the PWM control of the inverter without sampling delay.

PWM STRATEGY

The ideal voltage given by equations (12) and (13) is the space vector in the d-q model. Therefore, it should be transformed to the three phase voltage to drive the inverter. The transformed voltage vector $\bar{\mathbb{V}}3^*(n)$ and the eight possible voltage vectors of the three phase voltage-fed inverter are shown in Fig.3. It is noted that there are six voltage vectors with amplitude of (2/3)Vdc and two (called as zero vectors, hereafter.) without amplitude. Two PWM methods, vector selection PWM and average voltage PWM, are proposed in order to realize the $\bar{\mathbb{V}}3^*(n)$ with the inverter.

(1)Vector Selection PWM In the vector selection PWM, one of the eight vectors is selected during the sampling period. For selecting the vectors, the space is divided into eight regions [0]-[6] as shown in Fig.3 (a) and the vector may be selected depending on the position of $\bar{\mathbb{V}}3^*(n)$; for example, V(110) may be chosen when $\bar{\mathbb{V}}3^*(n)$ exists in region [2] and zero vector may be chosen for $\bar{\mathbb{V}}3^*(n)$ in region [0]. As a result, the selected voltage vector differs from the calculated ideal voltage vector both in amplitude and phase.

(2) Average Voltage PWM In this PWM method, $\overline{V}3^*(n)$ is synthesized by two adjacent vectors and zero vectors.[2] For example, $\overline{V}3^*(n)$ in Fig. 3(b) can be synthesized by the combination of V(100), V(110) and zero vector as shown in the figure. The time interval for each vector is easily calculated and is controlled by the interrupt from the internal timer of the DSP. The method is similar to the conventional sub-harmonic PWM but the switching frequency would be reduced to 2/3 when the ideal voltage is within a hexagon shown in Fig.3(b).

CURRENT CONTROL CHARACTERISTICS

Here, the experimental comparison of two PWM strategies are briefly explained. Fig.4 shows the steady state voltage and current waveforms for the 1.5 kW, 4-pole brushless motor under the rated current load. Fig.4(a) was obtained when the inverter was operated by the vector selection PWM, where the sampling period of the current control loop was 100 μs. On the other hand, the waveforms in (b) were obtained for the average voltage PWM, where the sampling period was 100 μs. It is apparent from the figures that the current waveform for the average voltage PWM is better than that of

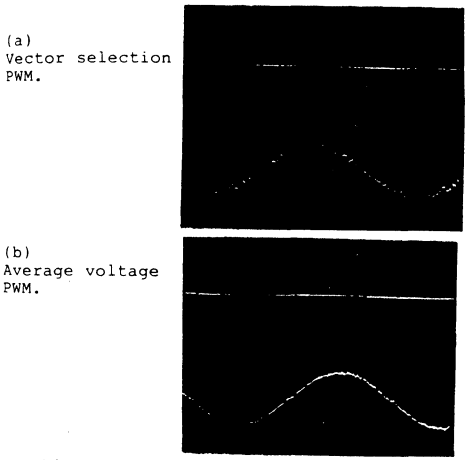
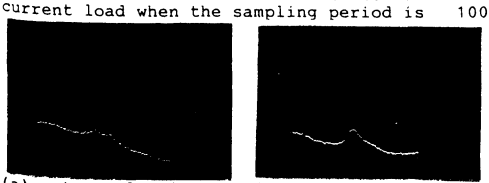


Fig.4. Steady state voltage and current waveforms.(T=100 μs, N=750 rpm)

the vector selection PWM. However, it may be concluded from the experiments that the vector selection PWM can reduce the acoustic noise compared with the average voltage PWM. Fig.5 shows the comparisons of the harmonic analysis between the two PWM methods under 25 Hz rated current load when the sampling period is 100



(a)vector selection PWM (b)average voltage PWM
Fig.5. Harmonic analysis of line current.

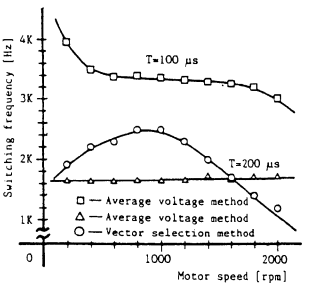
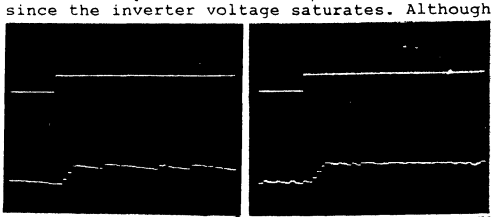


Fig.6. Switching frequency characteristics.

μs. Compared with the vector selection PWM, the average voltage PWM has a reduced harmonic contents below 5 KHz, thus improving the current waveforms.

In Fig.6 are shown the characteristics of the switching frequency versus motor speed. As apparent from the figure, switching frequency of the vector selection PWM varies according to the motor speed and has the maximum near 1000 rpm. When the operating frequency is low in the lower speed range, the required applied voltage is also low and the zero vectors are frequently selected in sequence. On the other hand, the required voltage is high in the high speed range and the voltage vectors with an amplitude are frequently selected in sequence to produce the applied voltage. It is noted that no switching occurs when the same vector is selected in succession at every sampling. However, in the intermediate speed range, the zero vectors and the voltage vectors with an amplitude are often selected alternately to produce an intermediate applied voltage near 1000 rpm. This is the reason why the switching frequency has the maximum near 1000 rpm. However, the switching frequency for the average voltage PWM are substantially constant and nearly equal to (2/3)*(1/2T), where T is a sampling period.

Fig.7 shows the step response of the q-axis current for two types of PWM methods when the stepwise change in the current command is applied. These figures show that there is no appreciable difference between the two PWM methods in regard to the transient response. For a small change in the current command, the motor current settles in one sampling because the output voltage of the inverter does not saturates. However, for the large change in the current command, for example, from 2 A to 10 A as shown in the figure, the current can not settle in one sampling and 3 - 4 samplings (this corresponds to 300-400 μs) are required since the inverter voltage saturates. Although



(a)vector selection PWM (b)average voltage PWM
Fig.7. Step response of q-axis current.

the output voltage of the inverter saturates, the settling time is as short as 300 or 400 μs, thus realizing the high speed transient response of the motor current.

EFFECTS OF PARAMETER VARIATION

As stated, the proposed current control is implemented by calculating the ideal voltage based on equation (12) and the control error would increase when the parameters used in calculation differ from those of the motor. The experiments were done to investigate the variation of the motor parameters due to the magnetic saturation and temperature rise. The results showed that there was no appreciable variation in armature inductance even when the motor current was increased up to five times the rated current, whereas the armature resistance increased by about 50 percent. On the other hand, the emf constant increased by about 16 percent due to the temperature rise.

The effect of parameter variations on the accuracy of current control characteristics has been investigated by the simulation. In simulating the system, the inverter has been treated to perform the PWM strategy explained in the preceding chapter but the dead time has not been considered. The results obtained for two kinds of PWMs have shown no appreciable difference and, therefore, the results for the average voltage PWM with sampling time of 100 μs are shown hereafter.

Steady State Characteristics Fig.8 shows the current control error for the same current command when parameters R, L and Ke of the motor varies while parameters used in the DSP-based controller are constant. Fig.8 (a) gives the result for low speed operation and (b) for the rated speed. The variation coefficient k is defined for each parameter as follows;

$$k = \frac{\text{R/L/Ke of motor}}{\text{R/L/Ke of controller}} \qquad \text{-------(14)}$$

and the control error is defined as follows;

$$e = \frac{Iqx - Iqo}{Iqo} \qquad \text{----------(15)}$$

where Iqo means the average q-axis current when the motor parameters are coincident with the controller parameters whereas Iqx is the average q-axis current when there is the parameter disagreement in motor and controller. The conclusion is summarized as follows.

(a)Armature Resistance: The control error is not hardly affected by the variation of the armature resistance, regardless of the motor speed.

(b)Armature Inductance: The effect of the armature inductance variation on the control error is somewhat different depending on the motor speed, but not serious in the range of k larger than 0.5, as shown in the figure. Below k=0.5, the fluctuation of the motor current increases because the inductance of the motor is small compared with that used in the controller, so approximations used in the development of the current control algorithm are assumed to be ineffective.

(c)EMF Constant(=torque constant): The effect of the variation of the emf constant is also somewhat different depending on the motor speed and the error increases with an increase of the motor speed. Due to the limitation of the inverter voltage, the error increases with the variation coefficient in the range of k larger than 1.5 when the motor speed is high.

Transient Characteristics The q-axis current

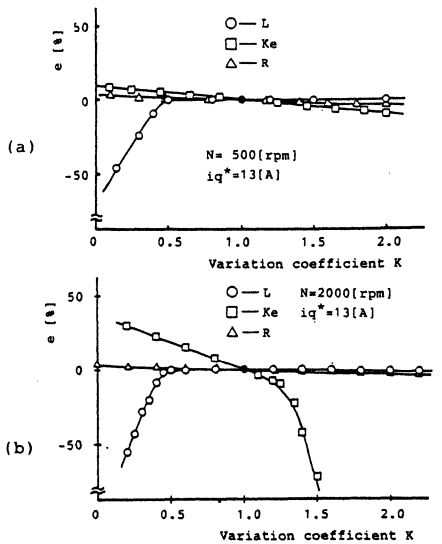


Fig.8. Control error for parameter variation. (a)500 rpm, (b)2000 rpm.

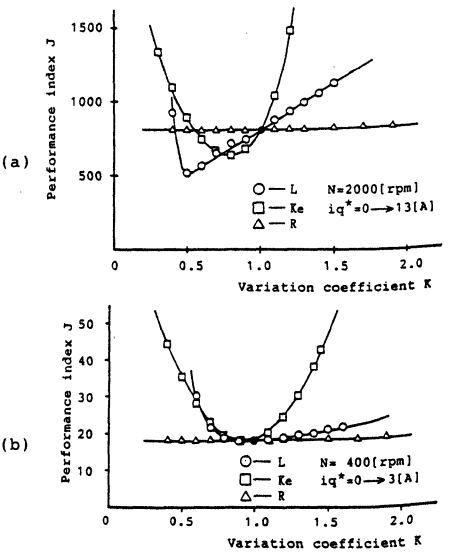


Fig.9. Performance index. (a)saturated voltage command, (b)un-saturated voltage command.

is used to estimate the effects of parameter variation on the transient characteristics. The performance index given by equation (16) is introduced for a stepwise change of the current command;

$$J = \sum_{i} [\, iq^*(i) - iq(i) \,]^2 \quad \text{---------(16)}$$

For calculating equation (16), summation was made from i=0 to i=99, because the oscillatory response might be obtained in some cases where there were the disagreements in parameters but even in that case, an oscillation would settle within 100 samplings. Fig.9 shows the performance index, where in (a) the inverter voltage is saturated for the large change of current command and in (b) the change of the current command is small so that the inverter voltage does not saturate. It is concluded from these estimations that the variation of armature resistance has no appreciable effect on the transient characteristics whereas variations of armature inductance and emf constant have significant effects on the transient characteristics.

IDENTIFICATION ALGORITHM

Parameter identification in regard to the armature inductance and the emf constant is performed by using the reference model of the brushless motor. The mathematical reference model is obtained by replacing n with n-2 in equations (6) and (7) and solving for $\mathbb{I}(n-1)$ and;

$$\mathbb{I}(n-1) = (T/L)[\bar{\mathbb{V}}(n-2) - \mathbb{E}(n-2) - R\mathbb{I}(n-2)] + \mathbb{I}(n-2) \quad \text{----------(17)}$$

$$\mathbb{E}(n-2) = \dot{\theta}(n-2) \begin{bmatrix} L\, iq(n-2) \\ Ke - L\, id(n-2) \end{bmatrix} \quad \text{--(18)}$$

where approximate relation $\bar{\mathbb{E}}(n-2) \fallingdotseq \mathbb{E}(n-2)$ was used. Taking $\mathbb{I}(n-1)$ in equation above as the reference output $\hat{\mathbb{I}}(n-1)$ and replacing L and Ke with corresponding parameters to be identified $\hat{L}(n-2)$ and $\hat{Ke}(n-2)$, the reference model is finally given by the following relation.

$$\hat{\mathbb{I}}(n-1) = [T/\hat{L}(n-2)][\, \bar{\mathbb{V}}(n-2) - R\,\mathbb{I}(n-2)] + \dot{\theta}(n-2)\, T \begin{bmatrix} -iq(n-2) \\ id(n-2) - \hat{Ke}(n-2)/\hat{L}(n-2) \end{bmatrix} + \mathbb{I}(n-2) \quad \text{----------(19)}$$

Then, the difference $\Delta\mathbb{I}(n-1)$ is obtained as follows by using equations (17) and (19).

$$\Delta\mathbb{I}(n-1) = \begin{bmatrix} 0 \\ Ke/L - \hat{Ke}(n-2)/\hat{L}(n-2) \end{bmatrix} \times \dot{\theta}(n-2)\, T + T \left[1/\hat{L}(n-2) - 1/L \right] \times \left[\bar{\mathbb{V}}(n-2) - R\,\mathbb{I}(n-2) \right] \quad \text{-----(20)}$$

Equation (20) means that $\Delta\mathbb{I}(n-1)$ would be zero when the identified parameters should be coincident with the motor parameters. When the parameters should not be identified correctly, $\Delta\mathbb{I}(n-1) \neq 0$ would result.

According to the previous explanation, the parameters could be identified by processing the current error $\Delta\mathbb{I}$ given by equation (20) through the PI controller and by taking the d-axis component as $\hat{L}(n-1)$ and q-axis component as $\hat{Ke}(n-1)$. Here, the d-axis component of equation (20) is;

$$\Delta id(n-1) = Ad(n-2) \left[1/\hat{L}(n-2) - 1/L \right] \quad \text{----------(21)}$$

$$Ad(n-2) = T \left[\bar{V}d(n-2) - R\, id(n-2) \right] \quad \text{----------(22)}$$

It is apparent from equations above that $\hat{L}$ would converge to the motor inductance L when the terms $[\, 1/\hat{L}(n-2) - 1/L \,]$ and $\Delta id(n-1)$ would have the same sign. For the emf constant, on the other hand, the relation for identification would be obtained from the q-axis component of equation (20) under the assumption that the armature inductance should have been identified as $L = \hat{L}(n-1)$ using the equations (21) and (22) and is given below.

$$\Delta iq(n-1) = Aq(n-2)\, [T/\hat{L}(n-1)] \times [\, Ke - \hat{Ke}(n-2) \,] \quad \text{----(23)}$$

$$Aq(n-2) = \dot{\theta}(n-2) \quad \text{----------(24)}$$

$\hat{Ke}$ would converge when $[\, Ke - \hat{Ke}(n-2) \,]$ and $\Delta iq(n-1)$ would have the same sign. From the discussion above, the identification algorithm is, therefore, given as in equation (25).

$$\begin{bmatrix} \hat{L}(n-1) \\ \hat{Ke}(n-1) \end{bmatrix} = \mathbb{K}p\, Sgn[\, \mathit{\Delta}(n-2) \,]\, \Delta\mathbb{I}(n-1) + \mathbb{K}i \sum_{k}^{n-1} [\, Sgn[\, \mathit{\Delta}(k-1) \,]\, \Delta\mathbb{I}(k) \quad \text{----(25)}$$

where

$$Sgn(x) = \begin{cases} +1 & (x > 0) \\ 0 & (x = 0) \\ -1 & (x < 0) \end{cases} \quad \text{--------(26)}$$

$$\mathit{\Delta}(k) = \begin{bmatrix} Ad(k), & 0 \\ 0, & Aq(k) \end{bmatrix} \quad \text{----------(27)}$$

and $\mathbb{K}p$ and $\mathbb{K}i$ are the gain matrix of the proportional integral type compensator and are given as follows.

$$\mathbb{K}p = \begin{bmatrix} Kpd, & 0 \\ 0, & Kpq \end{bmatrix}, \quad \mathbb{K}i = \begin{bmatrix} Kid, & 0 \\ 0, & Kiq \end{bmatrix} \quad \text{--(28)}$$

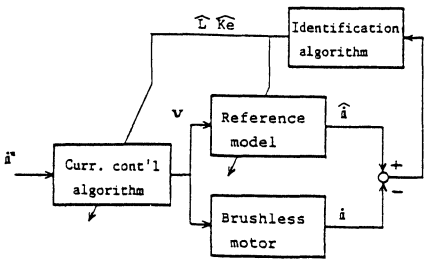


Fig.10. Adaptive current control system.

Fig.10 shows the block diagram of the adaptive current control system including the parameter identification. In the figure, the current control algorithm, the reference model and the identification algorithm correspond, respectively, to equations (12) and (13), equation (19) and equations (25) through (28). To verify the identification algorithm, simulations were made. The results of simulations have proved that the current control error can be greatly reduced by adding the ability of parameters identification to the preceding current control algorithm.

EXPERIMENTAL RESULTS

Fig.11 is the control system configuration of the prototype of the 1.5 kW brushless motor with 10X resolver. The configuration of the controller is quite simple because TMS320C25 can perform all necessary controls such as the position and speed calculations, identification and current control by software. The external electronics are necessary only for the resolver, the current detection and the

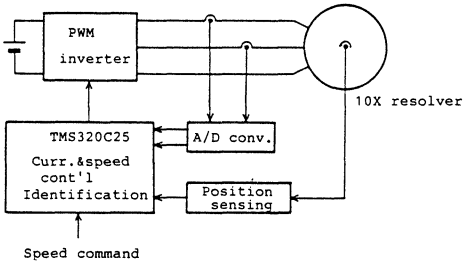


Fig.11. Control system configuration.

base amplifier. The position information is received from the 10X resolver every 800 μs and is interpolated every 100 μs by using the speed information to obtain the intermediate position information for the current control. The 16-bit speed information is obtained using the difference of position divided by sampling period. The motor current is detected every 100 μs by a Hall-CT and transformed through a 12-bit A/D converter. The control program was of 2.5K words and the required processing time

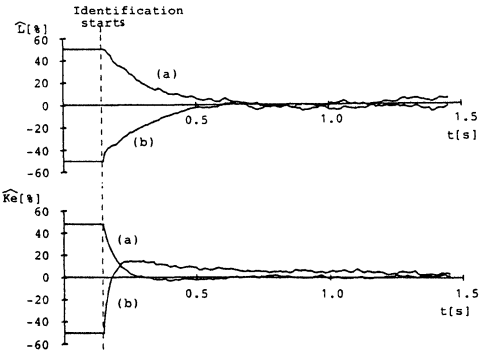


Fig.12. Convergence characteristics.

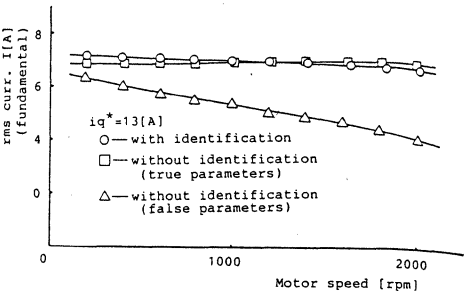


Fig.13. Current control characteristics.

was as short as 99 μs.

Fig.12 is the convergence characteristics of the parameters identification where in (a) the armature inductance and emf constant in the DSP-based controller were set 1.5 times as large as those of the motor, while in (b) the parameters used in the controller were set 0.5 times the motor parameters.

The current control characteristics with and without identification are shown in Fig.13. In the figure, □ corresponds to the case where the parameters in a controller are coincident with those of the motor itself, △ corresponds to the case where inductance of the controller is 70 percent of that of the motor and the emf constant is 30 percent of that of the motor.

CONCLUSIONS

In this paper, the new current control scheme for brushless DC motors using the high performance digital signal processors has been described. The system has a feature that the motor parameters used in the controller to determine the voltage command are identified at every sampling and, therefore, the current control can be always attained with high accuracy, regardless of the operating conditions such as the temperature rise. The algorithm has been verified by simulations and experiments.

REFERENCES

(1)T. Takeshita, K. Kameda, H. Ohashi and N. Matsui,"Digital Signal Processor-Based High Speed Current Control of Brushless Motor" Trans. Inst. Elect. Eng. Japan, vol-106B, No. 9, Sep. 1986.
(2)A. J. Pollmann,"Software Pulsewidth Modulation for μP Control of AC Drives" Trans. on Ind. App. vol-IA-22, No.4, July/August, 1986.
(3)P.Enjeti, P. D. Ziogas, J. F. Lindsay, and M. H. Rashid. " A New Current Control Scheme for AC Motor Drives" I.E.E.E. 1987 IAS Annual Meeting Conf. Record, pp202-208.

HIGH PRECISION TORQUE CONTROL OF RELUCTANCE MOTORS

Nobuyuki Matsui, Norihiko Akao and Tomoo Wakino

Department of Electrical and Computer Engineering
Nagoya Institute of Technology
Gokiso, Showa, Nagoya 466, Japan.

ABSTRACT

This paper presents the high precision torque control of the reluctance motor for servo applications. The prototype is the 3-phase, 8-pole reluctance motor driven by the MOSFET inverter. The current control as well as the speed control is performed by software of the digital signal processor, TMS 32010.

The motor is supplied by the sinusoidal current and two current control methods are proposed. One is based on the vector control principle to achieve the linearity between current and torque and another is developed to obtain the maximum torque/current ratio.

Due to the saliency, the instantaneous torque contains a large amount of ripple component. In case of the test motor, the ripple torque was as much as 26% of the rated torque under the sinusoidal current drive. The experiment showed the ripple component could be reduced to 6 % by superimposing the compensation current component to the current reference.

INTRODUCTION

Recently, the research on variable speed control of the reluctance motor has been done as "the switched reluctance motor" all over the world. The reasons for this tendency is that the motor is simple in construction and economical compared to the synchronous motor and the induction motor. In addition to that, the unipolar drive of the reluctance motor is possible and, therefore, the converter to drive the motor requires fewer switching devices compared to the inverter. From these reasons, the drive system can be more simple, economical and reliable.

Many papers have been reported on the switched reluctance motor in the past, but their main interests have been focused to the analysis and design of the motor or the drive circuit configurations. There are few papers which discuss the control aspects of the reluctance motor. In most of the control discussed in the literature, the winding current has the constant amplitude and is supplied to the motor in accordance with the rotor position.

This paper presents the digital signal processor-based control of the reluctance motor which is capable of operation as the servo motor. The controller functions include the computations of the rotor position and the feedback speed, current control and the compensation of the torque ripple. The current control, wholly implemented by software of the digital signal processor(DSP), is performed to obtain the linear relationship between current and torque similar to the concept of the vector controlled induction motor. In addition to that, another current control is proposed to obtain the maximum torque for the given winding current. In any case, the winding current is sinusoidal. However, due to the saliency, motor produces the torque ripple under the sinusoidal current excitation. The current control can also perform the compensation of the torque ripple by superimposing the compensation component to the current command. The amplitude and frequency of the compensation component can be determined by the information of the winding inductance.

The complete control system has been constructed and tested and the test results have been found excellent as a servo motor.

BASIC ANALYTICAL MODEL

Fig.1 is the configuration of the test motor whose construction is the same as the 3-phase variable reluctance type stepping motor. As the first approximation, the flux distribution along the air gap is assumed to be sinusoidal, then, the analytical model for one pole pair of the motor is obtained as in Fig.2. The inductance varies with the rotor position and, therefore, the self inductance is assumed as;

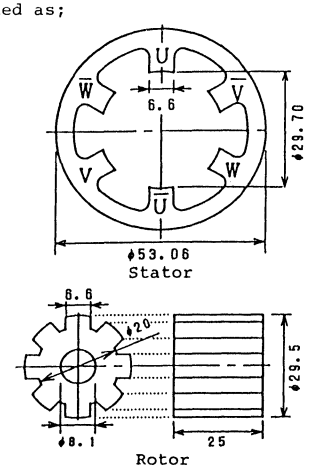


Fig.1. Configuration of test motor. (All dimensions are in mm.)

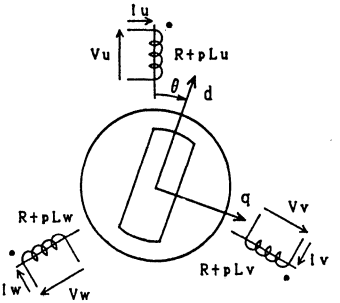


Fig.2. Analytical model for one pole pair of test motor.

$$Lu = Lg0 + Lg2 \cos 2\theta$$

$$Lv = Lg0 + Lg2 \cos (2\theta + 2\pi/3) \quad ---(1)$$

$$Lw = Lg0 + Lg2 \cos (2\theta - 2\pi/3)$$

where,

$$Lg0 = (Lmax + Lmin)/2 \quad ----------(2)$$

$$Lg2 = (Lmax - Lmin)/2$$

The mutual inductance between the stator windings is also assumed as;

$$Muv = Mg0 + Mg2 \cos (2\theta - 2\pi/3)$$

$$Mvw = Mg0 + Mg2 \cos 2\theta \quad --(3)$$

$$Mwu = Mg0 + Mg2 \cos (2\theta + 2\pi/3)$$

where,

$$Mg0 = (Mmax + Mmin)/2 = - Lg0/2$$

$$Mg2 = (Mmax - Mmin)/2 = Lg2$$

Using these definitions, the voltage equation of the motor is obtained as follows:

$$\begin{bmatrix} Vu \\ Vv \\ Vw \end{bmatrix} = \begin{bmatrix} R + pLu, & pMuv, & pMwu \\ pMuv, & R + pLv, & pMvw \\ pMwu, & pMvw, & R + pLw \end{bmatrix} \begin{bmatrix} iu \\ iv \\ iw \end{bmatrix} \quad ----------(4)$$

Using the well known d-q axis defined in Fig.2, the voltage equation (4) can be transformed into;

$$\begin{bmatrix} vd \\ vq \end{bmatrix} = \begin{bmatrix} R + pLd, & -\dot{\theta} Lq \\ \dot{\theta} Ld, & R + pLq \end{bmatrix} \begin{bmatrix} id \\ iq \end{bmatrix} \quad (5)$$

where,

$$Ld = 3(Lg0 + Lg2)/2 \quad ----------(6)$$

$$Lq = 3(Lg0 - Lg2)/2$$

and, from this equation, the analytical model of the reluctance motor is obtained as shown in Fig.3, assuming that the flux distribution is sinusoidal. The torque equation can be

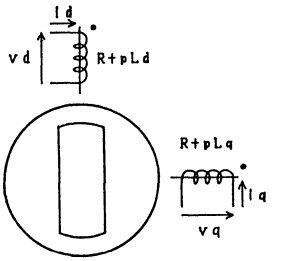


Fig.3. Equivalent d-q axis model of test motor.

obtained from Fig.3 and;

$$T = (Ld - Lq) id\ iq \quad ----------(7)$$

As a result, the two torque control methods can be proposed as follows;

(1) Vector Control Based on the model, the winding current can be controlled in the same way as that of the vector controlled induction motor, that is, the d-axis current is controlled as the exciting component and q-axis current as the torque component. In this case, the q-axis inductance is generally smaller than the d-axis inductance and, therefore, q-axis current is chosen as the torque component to achieve the fast response of the torque. As a result, the motor torque can be controlled to be proportional to the q-axis current as follows;

$$T = K\ iq, \quad K = (Ld - Lq)\ id \quad ------(8)$$

(2)Maximum Torque Control For the given winding current iw, the ratio id/iq can be controlled. In this case, the linearity between current and torque can not be achieved and the torque is given by the following relation;

$$T = (3/2)\ iw^2 (Ld - Lq)\ \sin 2D \quad ---(9)$$

where,

$$\tan D = iq/id, \quad iw = \sqrt{(id^2+iq^2)/3} \quad --(10)$$

Neglecting the magnetic saturation of the motor, the maximum torque is obtained for D = 45(deg.)

CONTROL SYSTEM

Fig.4 shows the control system configuration of the reluctance motor. Unlike the many drives of the reluctance motor in the literatures, the FET inverter supplies the sinusoidal current to the motor. The simple unipolar drive circuit for the sinusoidal current drive is now under consideration. The current hysteresis controlled PWM implemented by software was used to supply the sinusoidal current, where current control program was of 1.4 k words and the processing time was as short as 34 usec. The rotor position is obtained by the incremental type encoder(1000 ppr, Nikon RX1000-22-1). The output of the encoder is multiplied by four and is trans-

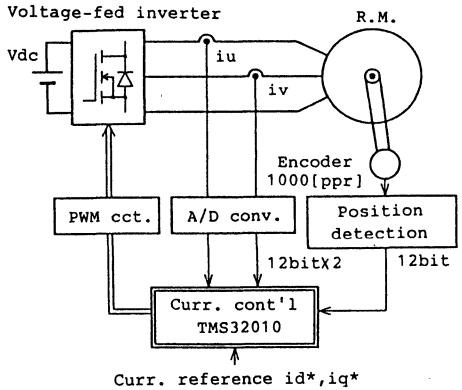


Fig.4. Control system.

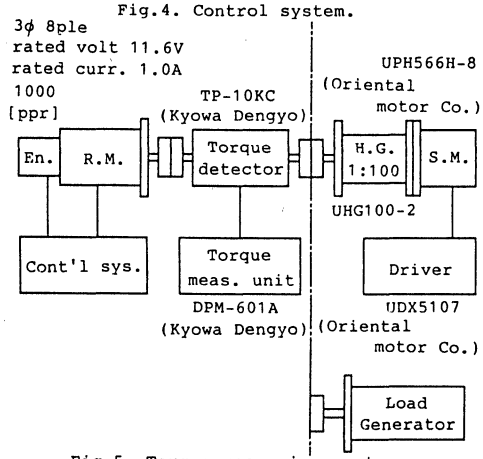


Fig.5. Torque measuring system.

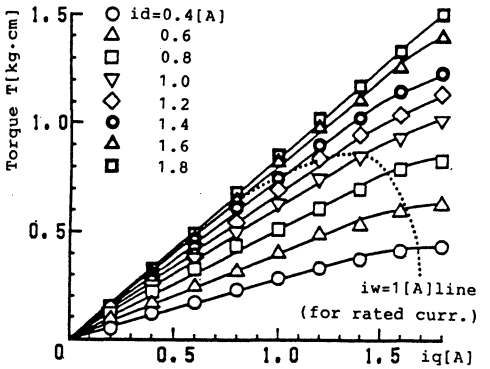


Fig.6. Torque-current characteristics for vector control.

formed into 12-bit digital quantity in the position detecting circuit. The winding current is detected by the Hall CT(NANA Electronics, 20CA-W) and is also transformed into 12-bit digital quantity.

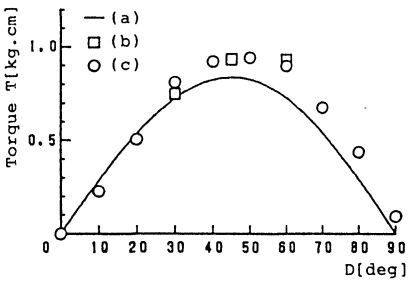
The estimation of the motor torque was performed by using the measuring system shown in Fig.5. This system was used to estimate both the steady state torque characteristics and the instantaneous torque characteristics. The steady state torque-speed characteristics can be obtained when the load DC generator is coupled with the axis. On the other hand, the torque ripple can be measured by connecting the stepping motor and the harmonic gear (1:100) to the shaft in place of the load generator. As the step angle of the stepping motor is 0.36 deg/step, the resultant resolution is 0.0036 deg/step and, therefore, it is possible to measure the instantaneous torque with respect to every rotor position by rotating the reluctance motor at very low speed(1.9 rpm).

TORQUE CONTROL CHARACTERISTICS

The steady state torque-current curves are shown in Fig.6 when the vector control is performed. In the figure, the d-axis current (exciting component) is the parameter and the dashed line corresponds to the rated current. Within the rated current region, the torque can be controlled to be proportional to the torque current.

Fig.7 shows the relation between torque and the current ratio angle D for the rated winding current. Two calculated curves obtained from equation (9) are shown in the figure, one is based on the motor inductances measured by impedance method and another by torque method. The details of the measurement will be explained later. Fig.8 shows torque-winding current characteristics when the maximum torque control is performed.

In Fig.4, the speed control loop can be added by modifying the control software. Fig. 9 shows the step response of the motor speed when the speed command is changed from -900 rpm to 900 rpm. In the figure, the current limit of the winding current is 1(A) and (a) was obtained according to the vector control for id=1(A) and (b) was obtained according to the maximum torque control.



(a)Impedance method
(b)Torque method
(c)Experimented

Fig.7. Torque and current ratio angle for the rated winding current.

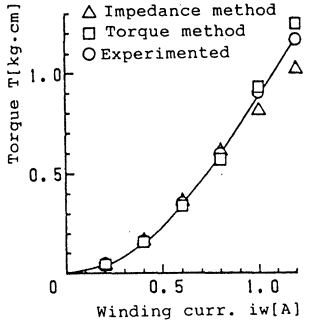


Fig.8. Torque-winding current characteristics for the maximum torque control.

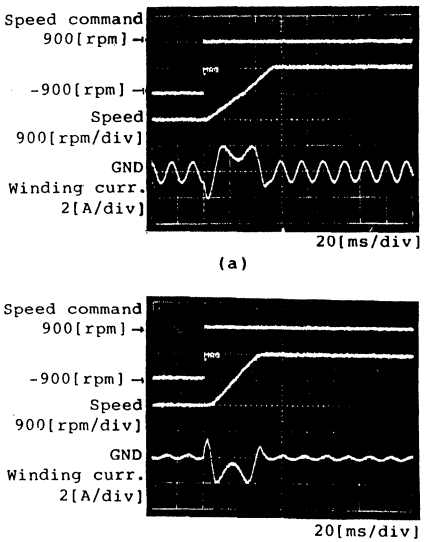
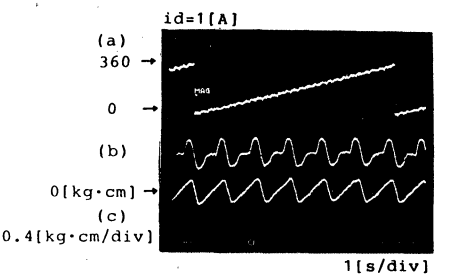


Fig.9. Step response of the motor speed. (For speed command change from -900 to 900 rpm) (a) vector control for id=1A (b) maximum torque control

Fig.10 is the instantaneous output torque characteristics versus rotor position when the vector control is performed for id=1(A). As expected, the torque ripple is notable. It is observed from this figure that the shape of instantaneous torque curves differs for different torque current and that the torque ripple exists even when the torque current is zero(This corresponds to the detent torque of the conventional stepping motor.).

MEASUREMENT OF INDUCTANCE

(1)Impedance Method In this method, the winding impedance is measured at every rotor position using the voltmeter, ammeter and



(a)Rotor position elec. deg.[deg]

(b)Torque for iq=1[A]

(c)Torque for iq=0[A]

Fig.10. Instantaneous torque current versus rotor position.

wattmeter. The measurement was done at every five mechanical degrees under the 60 Hz commercial supply. It should be noted that the current is distorted at a certain rotor position, which may produce the measurement error. The result is given in Fig.11.

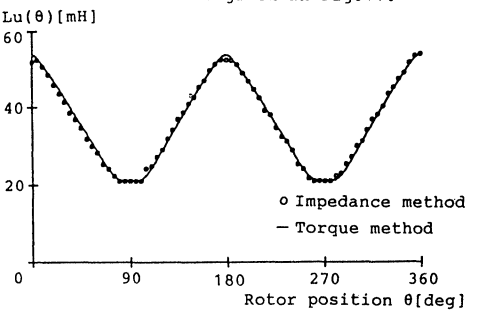


Fig.11. Winding inductance versus rotor position.

(2)Torque Method As is well known, the developed torque is given by equation (11) when, for example, only the U phase winding is excited by the dc current Iu. When the inductance Lu can be expressed as a sinusoidal function of θ, the developed torque is also a sinusoidal function of θ. However, when the inductance Lu

Table 1. Results of harmonic analysis of inductance.

(a)	2	3	4	5	6	7	8	9	10
(b)	3.99	5.34	2.96	0.11	1.64	0.15	0.60	0.04	0.20
(c)	3.35	5.03	2.78	0.12	1.51	0.23	0.64	0.06	0.21
(d)	3.93	5.57	2.11	0.15	0.65	—	—	—	—

Fundamental amplitude is 100%.

(a)Harmonic order

(b)Iu=0.5[A](Torque method)

(c)Iu=1.0[A](Torque method)

(d)Iu=0.5[A](impedance method)

$$Tu = (Iu^2/2) \partial Lu/ \partial\theta \quad \text{----------(11)}$$

contains the harmonic components, equation (12) should be used Lu in (11)

$$Lu(\theta) = Lg0 + Lg2 \sum_{n=1}^{\infty} hn \cos 2n\theta \quad \text{-----(12)}$$

and, therefore,

$$Tu = -Iu^2 Lg2 \sum_{n=1}^{\infty} hn\, n \sin 2n\theta \quad \text{---------(13)}$$

is obtained. Equations (12) and (13) mean that the frequency analysis of torque gives the inductance as a function of the rotor position. The result is also shown in Fig.11 by a straight line Table-1 shows the result of the harmonics analysis.

ESTIMATION AND COMPENSATION OF TORQUE RIPPLE

The mutual inductance of the reluctance motor is relatively small compared to the self inductance(in the test motor, it was 1-2 % of the self inductance) and, therefore, it can be neglected for the estimation of the developed torque. As a result, the torque equation is given as follows.

$$T = \sum_{k=UVW} (ik^2/2) \partial Lk/ \partial\theta \quad \text{-----------(14)}$$

Considering the harmonic component of inductance, equation (14) can be arranged as follows by substituting equation (12) into equation (14)

$$T = - Lg2 \left[\sum_{n=1}^{\infty} hn\, n \{ iu^2 \sin 2n\theta + iv^2 \sin n(2\theta+2\pi/3) + iw^2 \sin n(2\theta-2\pi/3) \} \right]$$

$$\text{----------(15)}$$

Here, the winding current is approximately related to the d-q axis current as follows.

$$\begin{bmatrix} iu \\ iv \\ iw \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} \cos\theta & -\sin\theta \\ \cos(\theta-2\pi/3) & -\sin(\theta-2\pi/3) \\ \cos(\theta+2\pi/3) & -\sin(\theta+2\pi/3) \end{bmatrix} \begin{bmatrix} id \\ iq \end{bmatrix}$$

$$\text{----------(16)}$$

Fig.12 shows the results of calculation for iq=0(A) and iq=1(A) under the same excitation id=1(A). It is noted that the higher harmonic

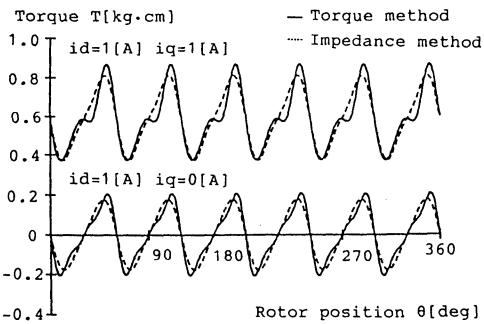


Fig.12. Calculated instantaneous torque.

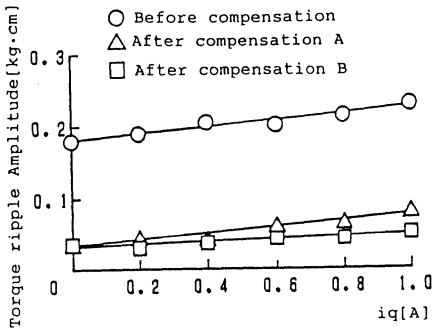


Fig.13. Amplitude of torque ripple versus torque current before and after compensation.

components of torque obtained by impedance method are omitted due to the inaccuracy of inductance. Fig.10 is the corresponding experimental result, which shows the calculated and the experimental values are well in accord. The amplitude of the torque ripple was measured for the constant excitation(id=1 A) and the result is shown in Fig.13. From this result, it is confirmed that the amplitude of the torque ripple is nearly proportional to the torque current. Therefore, the torque ripple can be expressed by equation (17), where the first term

$$\Delta T = K0\ F0(\theta) + K1\ iq\ F1(\theta) \quad \text{-------(17)}$$

represents the detent torque and the second term is associated with the torque current. In equation (17), K0 and K1 are the constants and F0(θ) and F1(θ) are the torque ripple functions which can be determined by equation (15) or from Fig.13.

There are two stages to compensate the torque ripple, compensation A and B.
(Compensation A) To compensate the detent torque, the compensation current iq0 defined by the following relation should be supplied to the motor.

$$T = K\ iq0 + K0\ F0(\theta) = 0 \quad \text{----------(18)}$$

(Compensation B) Once the detent torque has been compensated, the torque can be given by

$$T = K\ iq0 + K1\ F1(\theta)\ iq0$$

$$= K\ iq0 [\ 1 + K1\ F1(\theta)/K\ ] \quad \text{-------(19)}$$

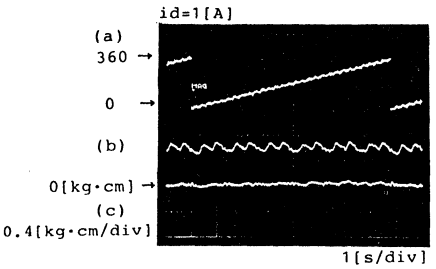
equation (19) and, therefore, the compensation current iq1 in equation (20) should be

$$iq1 = \frac{iq0}{1 + K1\ F1(\theta)/K}$$

$$\fallingdotseq iq0 (1 - K1\ F1(\theta)/K) \quad \text{-------(20)}$$

supplied in place of iq for developing the constant torque independently of the rotor position.

Fig.14 shows the result of compensation A. From this figure, it is observed that the

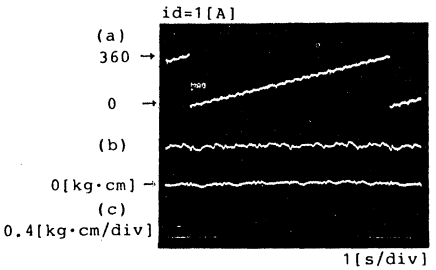


(a)Rotor position elec. deg.[deg]

(b)Torque for iq=1[A]

(c)Torque for iq=0[A]

Fig.14. Result of torque ripple compensation. (Compensation A)



(a)Rotor position elec. deg.[deg]

(b)Torque for iq=1[A]

(c)Torque for iq=0[A]

Fig.15. Result of torque ripple compensation. (Compensation B)

detent torque is compensated but the torque ripple due to the torque current is still remained. The amplitude of the torque ripple is also shown in Fig.13. Fig.15 is the result of compensation B. The amplitude of the torque ripple is measured and plotted in Fig.13.

The effect of compensation is largely affected by the estimation of the torque. As explained, the torque is calculated by equation (15) and the accuracy of inductance is important. The amplitude of the torque ripple has been reduced from 26 % to 6 % of the rated torque when the inductance obtained by the torque method has been used, whereas it has been only reduced to 10 % when the inductance by the impedance method has been used.

CONCLUSIONS

This paper describes the digital signal processor-based high precision torque control of the reluctance motor with the sinusoidal current excitation. Based on the analytical model, two types of the torque control are proposed, one is the vector control and another is the maximum torque control. In the vector control, the developed torque is proportional to the torque current as in the conventional vector controlled induction motor. In the maximum torque control, the linearity between torque and current is not achieved but the maximum torque is obtained for the given winding current.

It is well known that the reluctance motor produces the large amount of torque ripple. In the test motor, the amplitude of the torque ripple was as much as 26 % of the rated torque. For the estimation of the torque ripple, the accuracy of the winding inductance measurement is very important and, therefore, the measurement is discussed in the paper. Using the results, the torque ripple is estimated and compared to the experimental values. In addition, the compensation of the torque ripple by the current control is proposed. The prototype was tested and the performances were to be excellent.

REFERENCES

(1)P. J. Lawrenson : Variable-Speed Switched Reluctance Motors ; IEE Proc. vol.127, Pt.B, No.4, July, 1980.

(2)J. R French : Switched Reluctance Motor Drives for Rail Traction:Relative Assessment; IEE Proc. vol.131, Pt.B, No.5, Sep. 1984.

(3)A. Chiba, T. Fukao : A Control Method of Super High Speed Reluctance Motor for Quick Torque Response(in Japanese);Trans.IEE Japan vol.107-D, No.10, Oct. 1987.

(4)B. K. Bose, T. J. Miller, P. M. Szczesny, W. H. Bicknell : Microcomputer Control of Switched Reluctance Motor; IEEE Trans. on Ind. App., vol. IA-22, No.4, July/August, 1986.

High resolution position control under 1 sec. of an induction motor with full digitized methods

Isao Takahashi

Department of Electrical and Electronic System Engineering
Nagaoka University of Technology
1603-1 Kamitomioka Nagaoka Japan 940-21

Makoto Iwata

Power Supply Division
Sanken Electric Co., Ltd
677 Ohnohara Simoakasaka Kawagoe Japan 356

Abstract

The paper proposes a method of high resolution position control using an induction motor drive system. To get high resolution position control, it is combined two control methods.

One is ultra-low speed control based on principles of impulsive torque drives by using a high frequency dither signal which can compensate standstill frictions at low speed.

The other is linear control along an optimum sliding line which is decided by free-run characteristics of the mechanical load. The sliding line enables the improvement of a response and robustness of the system, and linear control area situated along this line improves the accuracy and stability.

The control circuit is composed of a high resolution position sensor (1296Kpulses/rev.), a controller using by a Digital Signal Processor(DSP) and a PWM inverter having optimized PWM switching patterns. The PWM pattern memorized in a ROM is made to generate the impulsive torque. The DSP makes simple circuit configurations, short calculation times and a speed sensorless system. Moreover it is made to have flexibility and intelligent ability such as auto tuning control for a parameter variations of the load.

The accuracy of the position control obtained in the experiment is 1/1296000 (rev.) which corresponds to one second of the mechanical angle.

1. Introduction

Recently, factory automation systems such as industrial robots and numerically controlled machines became highly advanced. Owing to maintenance-free, the use of an ac servo in the system would be most desirable in todays industry servo applications. But its complexity and expensiveness of the control circuit disturb its popularization, therefore a dc servo is still now widely applied for mechanical actuators.

Because of having stronger structure and better overload endurance, an induction motor is more suitable to overworked servo drive systems than dc or ac motors using permanent magnets.

The requirements of high accuracy, quick response and high stiffness characteristics are indispensable to highly advanced servo mechanism. In higher resolution position control systems, direct drives servo systems become applied in exchange for servo systems with reduction gears. But most of all these motors are reluctance machines with a large number of poles to get high position resolution. Therefore, the small size and light weight of the motor cannot expected and smaller air gap construction of the machine is also necessary to get the larger torque.

In spite of above merits, the induction motor has not applied in these systems because of difficulty to get the control accuracy. In the position control system, the more high resolution is needed the more affects stand-still frictions at low speed on the resolution.

This paper proposes a method of high resolution position control strategies using the induction motor for improving the above problem. To reduce effects of the stand-still friction at low speed control, the pulsation torque generated by the PWM inverter is employed for the torque dither signal. By using principles of its impulsive torque drive, ultra-low speed control of the induction motor under 1 rpd(day) has been experimentally realized. [1]

It combines the above ultra-low speed control with an optimum sliding line which has linear control regions near along the sliding line. The control can make not only robust systems but also stabilized and high resolution systems.

By the recent advancement of high speed and low cost micro-processors , it becomes possible to replace a conventional analog control circuit for a dizital control one. The use of micro-processors makes the circuit simpler as well as gives more sophisticated functions such a intelligent control as auto tuning adaptive control. The auto tuning control by the optimum sliding line for a parameter variations of the load is also proposed in this paper.

In the experiments, the accuracy of the position control (1/1296000 (rev.)), i.e., one second is achieved, which has never been realized by the usual induction motor drive technique.

2. Principles of the high resolution position control

Because of having robustness, a sliding mode has become increasing. But, from the point of view of high resolution control, the sliding mode control method would not always be suitable because of its large torque ripples or acoustic noises.

The control presented in this paper is somewhat different from the usual switching type sliding mode control as follows;

(1) A optimum sliding line

Figure 1 shows an optimum sliding line on a phase plane. To achieve the mentioned characteristics, the line is decided as close as to coincides with the free-run decelerate characteristic curve at low speed condition. And at the other speed region, to minimize the setting time, the line has to be set up as maximum deceleration curve as the drive system can generate.

Accordingly, because of the small torque ripple on the optimum sliding line near the target position, it is not only suitable a high resolution position control but also torque ripples or acoustic noises to minimum.

(2) Impulsive torque drive

In the linear control region, the impulsive torque is generated by using a PWM torque modulator. In this regions, the torque is proportional to the status error S decided from the speed ω and the position error θe as

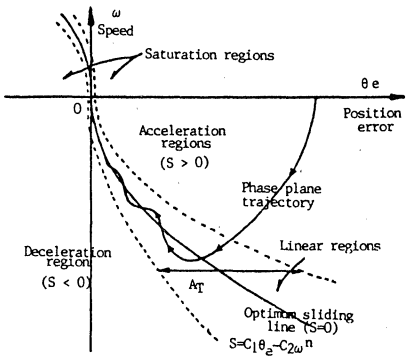
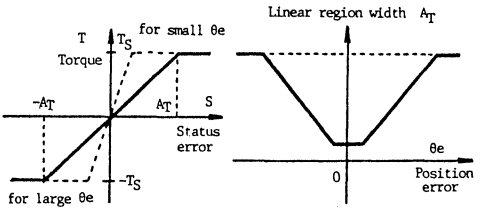


Figure 1 Optimum sliding control on a phase plane



(a) Relation of status error and torque

(b) Relation of position error and linear region width

Figure 2 Control methods of the linear control region

shown in figure 2(a). The status error S will be discussed fully in next section. The saturation level of the motor torque Ts varies with θe as shown in figure 2(b). At low speed, the system could have a fairly large gain under stable states. Therefore, for getting the good accuracy, it is better to use the high gain at the low speed and small error position states.

It is known that a high frequency dither signal makes compensate a non-linearity of the control system such as a static friction.[2] It can realize by superimposing the high frequency torque generated by inverter switchings to the mechanical load. As the stand-still friction must be canceled in the high resolution position servo mechanism, the high frequency impulsive torque drive would be superior to the linearly controlled one.

Figure 3 shows the schematic diagram of the ultra-low speed control system by the impulsive torque drive. Applying the high frequency and small amplitude impulsive torque slightly larger than the static friction, ultra-low and smooth speed control can be achieved. As shown in this figure, the motor speed ωm is measured by a dc tacho-generator and directly feedbacked. Comparing the reference ωm* with the speed ωm, the speed error ωm*-ωm is controlled to minimize by a high-gain PI circuit. Superimposing the impulsive triangular carrier, the non-linear load is linearlized and the system becomes more stable.

Figure 4 shows the inverter control circuit which

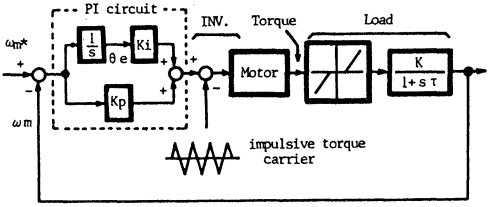


Figure 3 Schematic diagram of an ultra-low speed control

gives the impulsive torque to the induction motor. The absolute value of the output of the PI circuit is pulse width modulated with a impulsive triangular carrier wave and makes run/stop(R/S) signal of the motor. The carrier frequency used in the experiment is 2.0 KHz.

The output of comparator C1 which detects the polarity of PI output, gives forward/backward(F/B) signal of the motor and the output of C2 is R/S signal which specifies the time ratio of the non zero and zero voltage vector of the inverter. R/S signal gates 30 KHz clock signal which drives a 9 bits up/down counter and adjusts the inverter frequency. F/B signal is connected to the up/down control terminal of the counter and controls the direction of the phase rotation of the inverter. The output of the counter is connected to the address lines A0 A8 of a ROM.

Figure 5 shows the relation of waveforms of the impulsive carrier, the PI output and the motor control signals, F/B and R/S signals.

The ROM is programmed to get V/f constant control and least torque ripple. The impulsive torque frequency

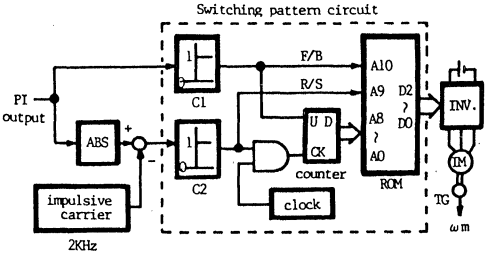


Figure 4 System configuration of an inverter controller

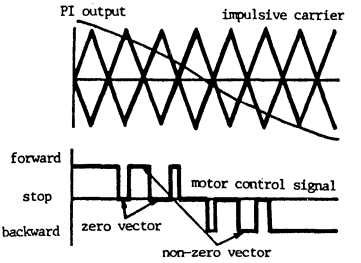


Figure 5 Relation of the PI output, impulsive carriers and control signals

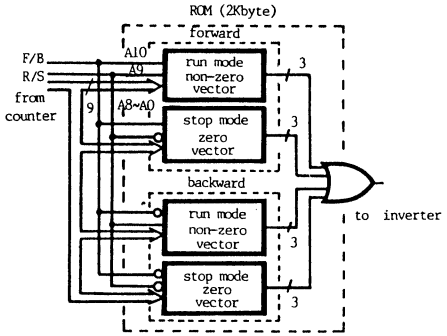


Figure 6 Schematic arrangement of PWM switching patterns of the ROM

controlled by R/S signal specifies the amplitude of the torque ripple. If the frequency is too low, the large torque ripple causes the position error. But too high, the system approaches to linear control and becomes unstable.

Figure 6 shows a schematic diagram of contents of the ROM composed of four kinds of the optimum switching patterns. The patterns are set for getting the minimum harmonic current at steady states. It has four switching patterns; run and stop modes for forward and backward modes, respectively. The run mode patterns generate the vectors to follow the circular locus of the primary flux linkage $\psi 1$ as close as possible with smallest number of switching. The patterns can read only by accessing the address A9 to the high level.

The zero voltage vectors patterns are used to decrease the voltage and frequency of the output. When the patterns are accessed, the flux is stopped its rotation and the motor decreases its torque. Accessing the signal of A9 to the low level, the patterns corresponding address of the above switching pattern can be read and simultaneously the counter is stopped by closing the gate.

Figure 7 shows an experimental results of ultra-low speed control characteristics of a conventional 0.75 KW induction motor. The speed control from 1 rpd (day) to 1500 rpm at no load condition is experimentally obtained. The speed ripple will be under + 0.2 rpd. For forward, locked, and backward control states, the speed drift and unstable states are not observed even in the loaded state.

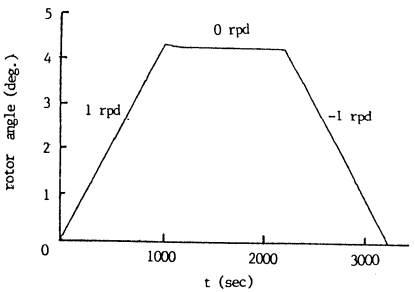


Figure 7 Experimental result of ultra-low speed motor control

3. Optimum sliding control line

The principles of ultra-low speed control can be applied to linear region control in the optimum sliding control. But a design of a PI circuit in figure 3 is very difficult to get a high stiffness and stability in all the area of the phase plane.

The simple PI circuit in figure 3 is only composed of the integrator with the constant gain of Ki and the proportional component with the constant gain of Kp. Since the output of the integrator 1/s corresponds to a position error θe and the output of the proportional component is the speed of the motor, those trajectory can be expressed by the straight line on the phase plane.

If the control is perfectly performed, it moves along the switching line as follows;

$$Kp\omega + Ki\theta e = 0 \qquad (1)$$

If the trajectory moves along the line, no output voltage is appeared in the inverter terminal because of no PI output voltage. But, in free-run condition, the trajectory doesn't always draw a straight line as equation (1).

Assuming that the load torque TL composed of the constant stationary torque TL0, the damper component $D\omega$, and the moment of inertia $J\dot{\omega}$, the state equation of the motion is

$$\begin{bmatrix} \dot{\theta}e \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -D/J \end{bmatrix} \begin{bmatrix} \theta e \\ \omega \end{bmatrix} + \begin{bmatrix} 0 \\ -1/J \end{bmatrix} (TL0+Tm) \qquad (2)$$

where,
Tm ; motor torque

In the case of the low speed operation, the value $D\omega$ is small in comparison with TL0 so that equation (2) is rewritten as follows;

$$\begin{aligned} &\theta e + (1/2)(TL0/J)\omega^2 = \theta e0 \qquad (\omega>0) \\ &\theta e - (1/2)(TL0/J)\omega^2 = \theta e0 \qquad (\omega<0) \end{aligned} \qquad (3)$$

where,
$\theta e0$; offset of the position

When the torque component of $D\omega$ is fairly large in comparison with TL0 such as in the high speed case, the trajectory is expressed by a straight line as follows

$$\theta e + (J/D)\omega = \theta e0 \qquad (4)$$

Accordingly, assuming the offset of the position is zero, it may be considered that a optimum sliding line which equals to the free-run trajectory of the system at low speed as equation (3) is set to the curve S=0 in equations (5).

$$\begin{aligned} &S = C1\theta e - C2(K\omega)^n \qquad (\omega<0) \\ &S = C1\theta e + C2(K\omega)^n \qquad (\omega>0) \end{aligned} \qquad (5)$$

where,
n = 2, C1 = 1, C2 = (1/2)(TL0/J) ; at low speed
n = 1, C1 = 1, C2 = J/D ; at high speed

The value of S is the status error. On the optimum sliding line, there is no switching and torque ripple-less operation is obtained.
But in the high speed region, the motor speed must be operated at the maximum speed, and must generate maximum braking torque if a minimum setting time is desired. In linear region, according to the value of S in equation (5), the torque is pulse-width modulated with the impulsive triangular carrier. It makes not

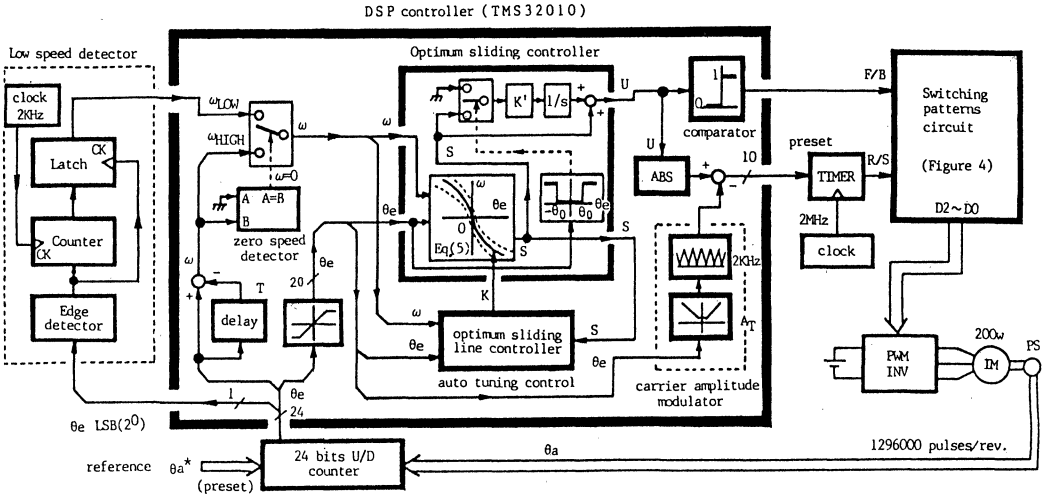


Figure 8 Schematic diagram of the DSP controller

only impulsive torque but also the linear control along the optimum sliding line. The carrier frequency used in the experiment is 2.0 KHz.

To compensate the error by the variation of the stand still load torque, another PI control must be also applied. The reference of the modulator corresponding to the PI output is switched to the value expressed by the following equation.

$$U = S + K'(\theta e)\int Sdt \qquad (6)$$

where,

K' ; variable function of θe

S ; status error in equation (5)

The second term can be used for compensation of a small disturbance torque and it acts within only a small θe region as $|\theta e| < 32$ (sec.). In the other region, the integration is stopped and saturated to reduce the extra transient phenomena. A time constant K'(θe) of the integrator is a function of θe and the value becomes larger as near θe=0. For a disturbance torque, the more works the integrator with high gain, the more maximum position error becomes small and gets higher response. The first term in equation (5) is a proportional component to improve stability.[3]

4. System configuration and Software

Figure 8 shows a configuration of the proposed DSP controller. As shown in this figure, the position θa of the induction motor is measured by a optical position sensor (81000 pulses per revolution), and one pulse is electrically divided into 16 to obtain the pulse train of 1296000 pulses per revolution which corresponds to one second of the mechanical angle.

Comparing θa with the digital reference θa* by a 24 bits up/down counter, the 24 bits position error e is applied to the DSP (TMS32010) controller. Inside of the controller, the data calculated using upper 16bits, but limited in 20 bits to simplify the calculation.

In this controller, calculating the status error U in equation (6), F/B signal and R/S time ratio data is decided just as the same way as shown in figure 4. The R/S time ratio data is transformed R/S signal by a presetable timer. But the amplitude of the carrier is modulated by AT shown in figure 2(b) and the PI gain is changed according to the optimum sliding line. F/B signal and R/S signal are applied to the switching pattern control circuit as shown in figure 4, and drives the PWM inverter by the optimum switching pattern.

Figure 9 shows the flowchart of the proposed linear sliding control algorithm of the DSP controller.

The motor speed ω at (k+1)T is estimated in high speed conditions by the following way.

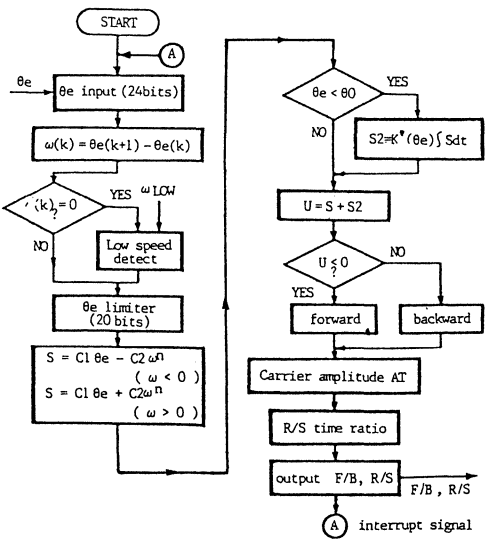


Figure 9 Flowchart of the DSP software

$$\omega((k+1)T) = \theta e((k+1)T) - \theta e(kT)/T \qquad (7)$$

where,
θe ; position error
T ; sampling period

It is only calculated by the pulse number of the position sensor during sampling period T. Sampling period T in the experiment is 500 μsec. (2.0KHz) which is equal to the period of the impulsive triangular carrier wave. Assuming that the output pulse of the position sensor is 2.0 KHz, the minimum detectable speed of the motor is 0.0926 (rpm).

When the DSP estimates the speed ω as zero, another measuring scheme must be applied under 0.0926 rpm. It is realized by measuring the pulse duration of the position sensor as shown in figure 8. When the value of ω((K+1)T) becomes zero, the low speed detector works by switching SW to ωLOW side. And when the counter data of the low speed detector is saturated, the speed ω is regarded as zero. θ_e is limited 20 bits (+524288 ~ -524288 pulses) in this controller.

To get the status error S, θe and ω are substituted in equation (5). When $|\theta e| < \theta 0$(32sec.), another PI control expressed in equation (6) is applied. The linear region width AT is decided by the position error. It corresponds to amplitude of the impulsive triangular carrier wave.

The status error S calculated from equation (5) is compared with zero and give F/B signal. The absolute value of S is pulse width modulated with the impulsive triangular carrier to get R/S time ratio, and gives R/S time ratio data.

The optimum sliding line with auto tuning controlled for a parameter variations will be discussed in next section. The calculation time is accomplished within 60 μsec. It is so small compared to T, but, considering from the stability problem, it is better to set the value as small as possible.

5. Auto tuning control

The recent development of a micro-processor enables digital controllers with a high intelligent abilities. Increasing the demands of complex servo mechanisms, it becomes very difficult to adjust the gains of controllers.

Accordingly, an auto tuning control is now a very promising method to the motion control system. (5) In this system, a simple auto tuning control is tried by changing the slope of the optimum sliding line with a parameter variations. The optimum sliding line is varied instantly by observing the relation of the phase plane trajectory and the sliding line.

The phase plane trajectory usually varies along to specified the sliding line as shown in figure 10(a). But as shown in the trajectory (b), when the slope of the sliding line is larger than that of the trajectory, the trajectory has some ripples. On the other hand, as trajectory (c), when the slope of the sliding line is too smaller, the overshoot and the limit cycle is observed. Accordingly, the optimum sliding line would be able to specified by observing the motor and load characteristics variation.

Figure 11 shows a control result using the tuned optimum sliding line. As shown in this figure, two auto tuning lines SL=0 and SH=0 are considered in both sides of the optimum sliding line. When the trajectory collides with the lower auto tuning line SL=0, it is better to use the larger slope optimum sliding line to get more stable response. On the other hand, the trajectory doesn't reach the higher side of the auto tuning line SH=0, the slope of the optimum sliding line must be increased.

The auto tuning lines SL and SH are specified in this paper as follows;

$$SL = C1\theta e - C2(KL\omega)^n \qquad (\omega<0)$$
$$SH = C1\theta e - C2(KH\omega)^n \qquad (\omega>0) \qquad (8)$$

where,
KL ; K - ΔK , KH ; K + ΔK
K ; the gain of equation (5)

In this region, only the gain of K in the optimum sliding line is adjusted, whether the trajectory collides with the optimum sliding line or not.

Figure 11(a) shows the trajectory with no auto tuning where the ripple is observed in the phase plane trajectory. Figure (b) is auto tuning where the ripple is compensated. These real time control is easily executed by the DSP controller.

Figure 12 shows a schematic diagram of the DSP software of proposed auto tuning control. S in equation (5) is calculated from θe and ω by the sliding line controller and SL and SH are by the auto tuning lines controller. Comparing the value of S with the value of SL and SH, the output of 3-state comparator is

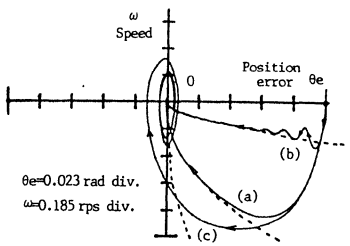
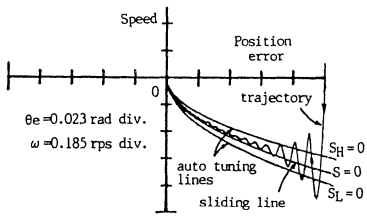
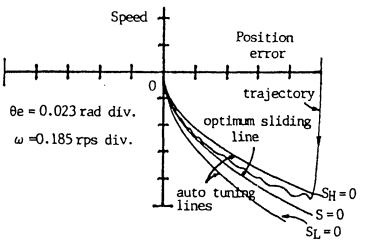


Figure 10 Phase plane trajectories for various sliding lines (simulation results)



(a) no-auto tuning control



(b) auto tuning control

Figure 11 Control results using the tuned optimum sliding line (simulation results)

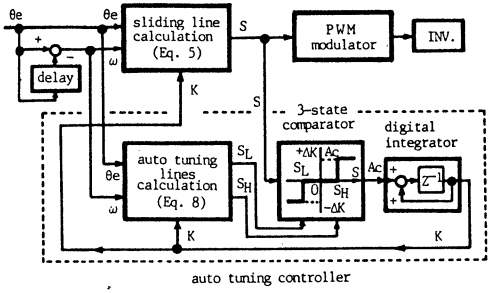


Figure 12 Schematic diagram of auto tuning control

digitally integrated for calculating the optimum gain factor K. At every sampling period, K is decided by adding the adjusting coefficient Ac of K which corresponds to the output of the 3-state comparator, ΔK, 0 or -ΔK. Thus the optimum sliding line can be adjusted if auto tuning lines are decided.

Figure 13 shows a flowchart of the auto tuning control. Only the auto tuning control loop is expressed by solid line. To get the values of S, SL and SH, θe and ω are substituted into equations (5) and (8). Comparing the results S with SL and SH, the adjusting coefficient Ac of the gain K is decided using above method. When S is smaller than SL, the adjusting coefficient Ac is -ΔK. When S is larger than SH, Ac is +ΔK.

Accordingly, when S is situated between SL and SH, the sliding line is recognized as an optimum sliding line, and Ac is zero. Otherwise, K is modified by Ac at next time. To decide the gain K of the optimum sliding line, Ac is integrated at sampling period. The adjusting coefficient ΔK is set to 0.2K in this software. The calculation for the auto tuning is accomplished within 10 μsec..

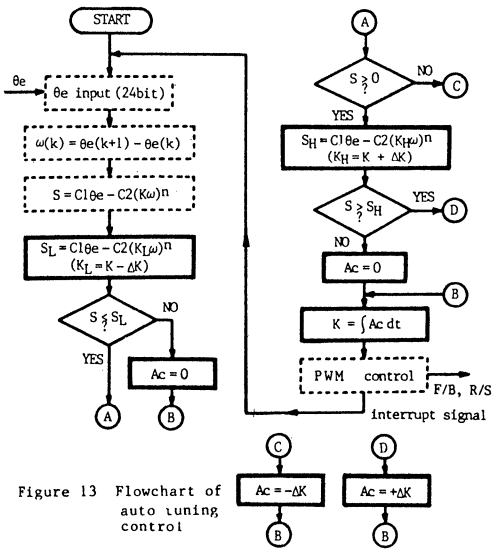


Figure 13 Flowchart of auto tuning control

6. Experimental results

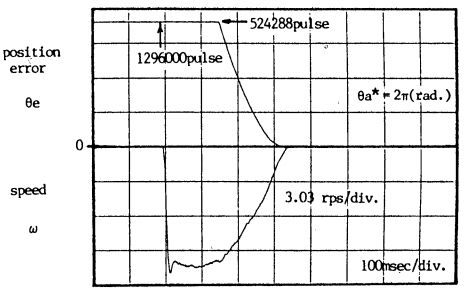
Figure 14 (a) shows the step response at no load condition and figure 14 (b) shows its phase plane trajectory under the condition. The reference θa* is 1296000 pulses = 2π (rad.). In this figure (a), because of the position error limiter(20bits), it is saturated from 1296000 pulses to 524288 pulses.

Figure 15 shows the transient response near the target position. In this figure, the minimum 1 step of the position error corresponds to 1/ 1296000 (rev.) = 1 (sec.). The accuracy of the position control obtained in the system is 1/1296000 (rev.) = 1 (sec.)

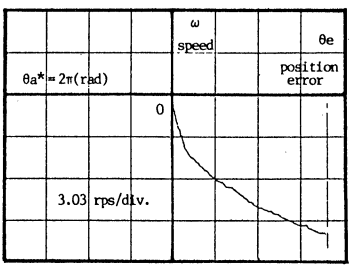
Figure 16 shows the distribution of the position error of one hundred times tests. Considering the allowable error of ±1 pulse, 90 % of the test results are satisfied the error limit.

Figure 17 shows the response of the stepwise disturbance torque input. As shown in this figure, the position error is observed during 20 msec but it is canceled by a disturbance compensator shown by equation (6).

Figure 18 shows the trajectories of the untuned and tuned cases. If the sliding line is not optimum, a trajectory ripple are observed as shown in figure (a). It gives not only the torque vibration to the mechanical load but also a bad response and accuracy. Comparing with these figures, it is shown that the trajectory ripple and the response are fairly improved by auto tuning control as shown in figure (b).



(a) Step response at no load



(b) Phase plane trajectory

Figure 14 Step response of 1 revolution of the system

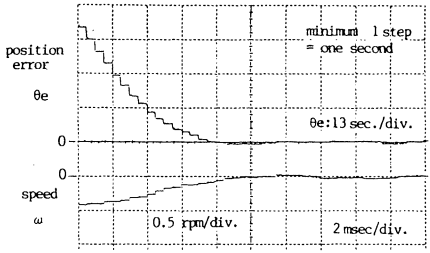


Figure 15 Transient response near the target

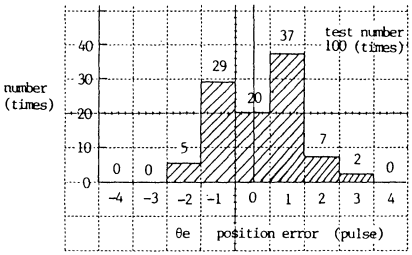


Figure 16 Distribution of position errors

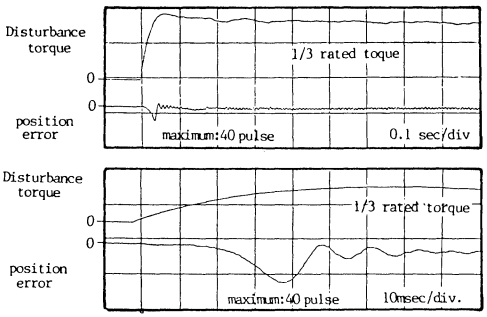


Figure 17 Response to the disturbance torque input

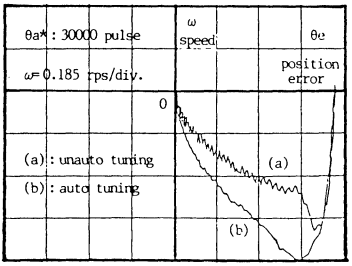


Figure 18 Auto tuning control on a phase plane

7. Conclusion

In this paper, to get the high resolution position control method of an induction motor, skillful control techniques are applied. And the following results are obtained:

1) By using the impulsive torque drive at linear region has a good stability and precision at low speed area.
2) Optimum sliding mode control with the variable gain which is different from conventional one, enables the improvement of an accuracy, responses and robustness.
3) For compensation of a disturbance torque at the stand-still condition, the PI controller with the variable time constant is also employed.
4) The use of the DSP makes simple circuit configurations and a speed sensorless system.
5) The system is made to have flexibility and intelligent ability such as auto tuning control of sliding mode switching line for a parameter variations in the motion control system.

As the results, the proposed motion control method would be available in a high resolution servo under 1 sec. resolution.

Through experimental results, the validity of proposed control is provided to be very promising and skillful techniques to the high resolution position control system.

Acknowledgments

The authors would like to express their appreciation to Mr.S.Asakawa and Mr.S.Tanaka of Sanken Elec.Co.Ltd. and the Power Electronics Laboratory members of Nagaoka University of Technology. Part of the work is supported by Grant-in-Aid for Scientific Research of the Ministry of Educaition and by the foundation of Highly Advanced Mechatronics Technology.

Reference

(1) I.Takahashi, S.Asakawa, " Ultra-wide speed control of an induction motor covered 106 range " IEEE-IAS pp " (1987)
(2) Olle I.Elgerd " CONTROL SYSTEMS THEORY ", International student edition, McGRAW-HILL INC. 1967
(3) I.Takahashi and M.Iwata " High resolution servo system of an induction motor using linear mode sliding control " PCIM'88 (INTELLIGENT MOTION), Japan (1988) p.254
(4) T.Iwakane and T.kume " High performance vector controlled ac motor drives (applications and new technologies) " IEEE-IAS (1985)
(5) R.Lorenz " Tuning of Field-Oriented Induction Motor Controllers for High-Performance Applications " IEEE-IAS (1986)

A TMS32010 Based Near Optimized Pulse Width Modulated Waveform Generator

R.J.Chance and J.A.Taufiq, Dept. Electronic & Electrical Eng., University of Birmingham, Birmingham B15 2TT, UK

Abstract

This paper describes a system for dynamically calculating optimized pulse width modulated (PWM) waveforms for use with voltage source inverter (VSI) fed induction motor drives in railway traction applications. A TMS32010 signal processing microprocessor, capable of fast arithmetic is interfaced to a novel random access memory based waveform generating hardware. This provides the capability to control waveform detail impossible with more conventional microprocessor based systems. Although the paper concentrates on the implementation of a particular algorithm, the design can implement variable pulse widths in multiphase systems and in real time. An important aspect of this work is the role played by microprocessor simulation in testing the design.

Nomenclature

m number of switching angles per quarter cycle of PWM waveform.
α_k kth switching angle
NP1 modulation depth of PWM waveform
Vdc inverter dc link voltage
VSI voltage source inverter
f inverter output frequency

Introduction

With the increasing availability of high power gate turn off (GTO) thyristors, there has been a renewed interest in inverter drives for electric multiple unit, metro and light rail applications. Comparing the GTO inverter and GTO chopper from an economic viewpoint, it is widely accepted that the GTO voltage source inverter (VSI) is the most favourable of all the inverter configurations. The GTO VSI does not require a preconditioning chopper and input voltage fluctuation is compensated by the VSI controller.

From a signalling viewpoint, the main difference between the VSI and the fixed frequency chopper drives is that the former can potentially generate components over a wide range of frequencies as the VSI operates from minimum to maximum frequency. Previous experience with chopper generated interference suggests that methods of control which can theoretically eliminate components at the signalling frequencies will be required by most metro authorities when new equipment like the GTO VSI is considered. In this respect, it has been shown [1,2] that a harmonic elimination optimized PWM based ratio changing scheme which is tailored to suit the type of signalling system used is the best solution. Although other types of PWM scheme such as regular sampled and distortion minimised are more commonly used in industrial AC drives, these are not really the ideal in this particular application.

With these drives, the problem of signalling interference is more pronounced with power frequency type track circuits. As in this case the signalling frequencies are relatively low, typically below 400 Hz, any components at these frequencies generated by the GTO VSI will not be significantly attenuated by the input filter of the traction equipment. Therefore it is essential to ensure that the GTO VSI does not generate any components at these signalling frequencies. In the case of audio frequency type track circuits, the range of signalling frequencies used is usually around 2-10 KHz and with the typical constraint on the maximum inverter switching frequency, it is not possible to eliminate the inverter generated harmonics in this band. However, with typical input filter values, it can be shown that the signalling frequency components in the rails will be much less than the typical threshold levels [1,2].

To date, the implementation of this type of optimized PWM scheme has been limited to a look-up table of the exact switching angle data, which is precomputed for a given inverter input voltage and ratio changing scheme. A high incremental resolution of the angles may be needed which could require substantial memory. Also fluctuations in the DC input voltage can only be compensated by performing an interpolation of exact switching angle data. Therefore the preferred solution would be to generate these switching angles on-line. It has been shown [2] that it is possible to approximate the exact switching angle trajectories by an algorithmic approach, which results in relatively simple equations. This algorithm has also been shown to generate near optimal switching angles for any number of angles per quarter cycle, m. The equations to be computed are derived in [1] and can be summarized as follows:

For odd k,

$$\Delta k = 0.4 \sin\left[\frac{(k-0.5)\times 59.2^\circ}{m} + 60.4^\circ\right]$$

$$\alpha_k = \frac{(k+1)\times 60^\circ}{(m+1)} - \left[\frac{120^\circ \times \Delta k \times NP1}{0.8\,(m+1)}\right] - \Delta Dk \qquad (1)$$

where $\Delta Dk = 0$ for $NP1 \le 0.8$

$$\Delta Dk = \frac{13\,(NP1-0.8)^2}{0.09m} \times \sin\left[\frac{180^\circ \times k}{(m+5)}\right] \text{ for } NP1 > 0.8$$

For even k,

$$\Delta k = 0.4 \sin\left[\frac{k}{(m-1)}\left(58.6^\circ - \frac{12.5^\circ}{(m-2)}\right)\right]$$

$$\alpha_k = \frac{k\times 60^\circ}{(m+1)} + \left[\frac{120^\circ \times \Delta k \times NP1}{0.8\,(m+1)}\right] - \Delta Dk \qquad (2)$$

where $\Delta Dk = 0$ for $NP1 \le 0.8$

$$\Delta Dk = \frac{14\,(NP1-0.8)^2}{0.09m} \sin\left[\frac{180^\circ \times (k-1.5)}{m}\right] \text{ for } NP1 > 0.8$$

Choice of microprocessor

This type of microprocessor based PWM waveform generator design often uses a general purpose microprocessor interrupted by a counter-timer as described in [3]. Three different types of microprocessor, Z80, 8086 and TMS32010 were benchmarked

TABLE 1
PERFORMANCE OF 3 PROCESSORS FOR OPTIMISED PWM SCHEME

	Z80(4MHz)	8086(5MHz)	TMS32010(20 MHz)
multiply time (16x16 bit)	300	30	0.2
divide time (16/8 bit).....	400	40	4
memory/register transfer time (16 bit word)....	3	2	0.2
estimated time to compute (2) (NP1 < .8).....	3000	310	20
on chip RAM (bytes).....	0	0	288
timer peripheral availability.....	good	good	poor
multi-level interrupts.....	good	good	poor

[all times are in microseconds]

for this waveform generator as shown in table 1.

Only the TMS32010 can compute all the switching angles with the algorithm in the desired time of 2 ms. It also has enough random access memory (RAM) on chip for executing the algorithm. The main difficulty with the TMS32010 is the lack of suitable counter/timer peripherals. Therefore a totally different approach has been used for waveform generation.

Waveform Generation Circuitry

One method of generating a waveform is to store it as a binary pattern in RAM. Thus a square wave, for example, can be created by storing N binary ones followed by N zeros and reading these locations at regular intervals. The RAM address becomes in effect the waveform angle or time which can be generated by a binary counter. This method is not efficient for generating a changing waveform because a large number of RAM locations must be continually updated. However the memory based waveform generation hardware used here is based on the storage of identifying codes only at the addresses (switching edges) where a waveform state change occurs. This reduces to a minimum the memory locations used to define the desired waveform. 4096 words of 8 bits are used with two bits of each word per phase, so that six bits can produce a three phase waveform. The waveform is stored as a 'map' of switching angles or times as illustrated in Fig.1.

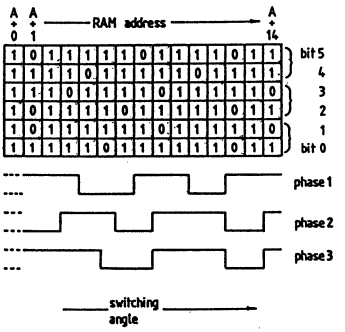


Fig.1 Codes to generate a three phase waveform from RAM

Each address location contains the two bits per phase coded as follows to load an output latch as the address generator (binary counter) is incremented:

11 no change in latch state.
01 set output latch to '1'.
10 reset output latch to '0'.
00 not used.

The outputs of three such latches thus generate the waveforms for a three phase supply. Before using the RAM, all locations are initialised to the 11 (no change) condition. A waveform is created by writing either a 01 or a 10 into RAM locations which correspond to the desired switching angles. To change this waveform, the contents of these addresses must be reset to the no change (11) state before new values are written.

Two methods may be used to generate a waveform of variable frequency. To drive the counter that supplies the RAM address from a variable frequency source while storing a complete waveform cycle or alternatively from a fixed frequency source so that the RAM addresses become equivalent to time delays. The first method requires an inconveniently high frequency source of 70 MHz to give the 1% resolution required at 150 Hz. The second method requires the RAM contents to be changed several times in one cycle and therefore to be updated frequently even when a constant output frequency is generated. The potential accuracy of this generator is high and is the one used here. This method of using RAM external to the processor address space for generating the PWM waveforms for a variable frequency VSI is apparantly novel. Existing waveform generators usually use some form of counter-timer.

TMS32010 waveform generator interface

The implementation of this scheme is shown in Fig.2. The TMS32010 'CLKOUT' signal is divided from 5 MHz by a prescaler to drive the 12 bit binary counter.This rate is currently 1.25 MHz. Thus external RAM address updates by the counter are synchronised to processor operations. During every machine cycle, the TMS32010 produces one of the following MUTUALLY EXCLUSIVE signals:

(1) MEN instruction fetch.
(2) WE port write (output).
(3) DEN port read (input).

In a TMS32010 running at 20 MHz, MEN occurs at a 5 MHz rate except during the input and output operations. This makes it possible for the hardware counter to 'steal' the MEN cycles for updating the waveform from the output RAM while allowing the TMS32010 to access the RAM without constraint. In Fig.2 the following operations are carried out:

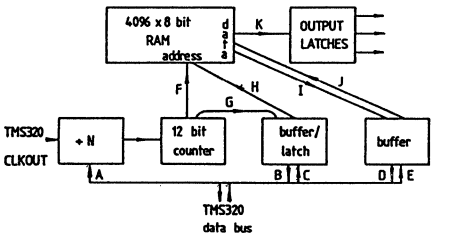


Fig.2 Data pathways in the waveform generation hardware

* Output a new divider value (A)
* Input the current address (B,G).
* Output a new RAM address (C) to the buffer/latch.
* Input RAM contents (I,D) at the previously latched address (H).
* Output new RAM contents (E,J) from the TMS32010 at the previously latched address (H).

The above TMS32010 operations all coincide with either a DEN or WE TMS32010 machine cycle and are interleaved with output latch updates performed in hardware during the MEN (data path K ; address path F) time slots.

Software Overview

The TMS32010 software may be conveniently split into four sections: obtaining m and NP1, calculation of switching angles, conversion of angles to time delays and output RAM update.

Obtaining m and NP1

The inverter output frequency and dc link voltage values are obtained from two analog to digital convertors. From these two inputs, the two variables in (1) and (2), namely m and NP1, need to be deduced. The

required value of m for a given value of inverter output frequency is obtained from a look up table equivalent to Fig.3.

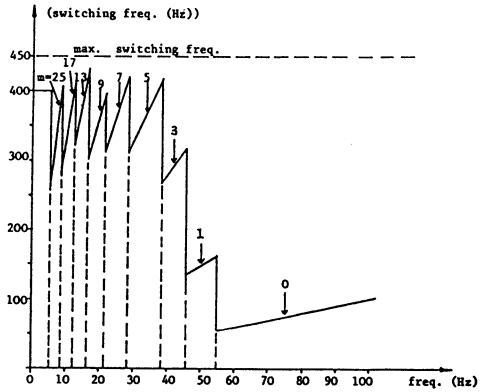


Fig.3 Ratio changing scheme for deriving m from inverter output frequency

This particular ratio changing pattern has been devised to ensure that the VSI drive does not produce any components at typically used signalling frequencies over the entire inverter output frequency range [2]. The required variation of the VSI fundamental line voltage with inverter output frequency is also stored as a table. In railway traction systems, Vdc is subject to a +20% -30% variation. In order to keep the motor line voltage constant, irrespective of this variation, NP1 is altered to compensate.

Switching angle calculation

The angles are calculated from (1) and (2) using integer arithmetic. The output of this stage is a list of switching angles between zero and $\pi/2$ radians. Sine and cosines are derived from a table.

Angle to time conversion

The switching angles must be divided by the inverter output frequency f to give corresponding time delays. A 16 bit word does not give the required 1/300,000 time resolution. This problem has been solved by a form of block floating point [4] which may be efficiently used on the TMS32010. For frequencies above 12 Hz, times are divided by 4, but below 12 Hz, they are divided by -64.

Updating waveform generation RAM

The expanded time value has 22 bits. Bits 0-11 become the RAM address, bits 12-21 represent the number of times that the RAM must cycle through its 4096 addresses to reach the correct switching time.

Table 2 shows the updating of this RAM which is carried out on either the upper or the lower 2048 addresses. While the lower addresses are being accessed by the hardware which reads RAM contents to the output latch, the upper ones can be updated by the TMS32010 and vice versa. Before new switching times can be written to a 2048 word portion of RAM, the times from the previous cycle must be deleted. Note that the TMS32010 has unrestricted access to the chosen half of the memory and therefore updating does not have to be done in a particular order. Addresses containing switching codes for a particular half cycle are saved in TMS32010

TABLE II
OUTPUT RAM ACTIVITY WITH THE PASSAGE OF TIME

RAM ADDRESS/ COUNTER VALUE	WAVEFORM GENERATED	DELETE OLD CODES	INSERT NEW CODES
0 - 2047	from RAM cycle N	from RAM cycle N-1	for RAM cycle N
.....	0-2047	2048-4095	2048-4095
2048 - 4095	from RAM cycle N	from RAM cycle N	for RAM cycle N+1
.....	2048-4095	0-2047	0-2047
0 - 2047	from RAM cycle N+1	from RAM cycle N	for RAM cycle N+1
.....	0-2047	2048-4095	2048-4095
2048 - 4095	from RAM cycle N+1	from RAM cycle N+1	for RAM cycle N+2
	2048-4095	0-2047	0-2047

internal data memory for deletion during the next cycle. As the end of the PWM waveform will hardly ever occur at RAM address zero, the address of this point must be added to the switching times for the next cycle of the waveform.

Testing the System

Practically all the development of this project was performed on simulated rather than real hardware.

The TMS32010 simulator

A simulated TMS32010 [5] was used to test the software to the point where it could be used with confidence in the target hardware (see appendix A). This simulator has particularly versatile means of interfacing to TMS32010 streams of test data held on the host computer. Additionally, real or even non-existant peripheral devices can be simulated in the 'C' language. In this case, the waveform generator hardware of Fig 2 was completely simulated in software.

TMS32010 equation calculations

The scheme shown in Fig. 4 was used to check the TMS32010 equation calculations performed in integer arithmetic against 'accurate' values carried out in floating point. Values of NP1 and m were supplied as input data files to simulated TMS32010 ports. Angles computed by the simulated TMS32010 were saved on file. This file was used as the input to a BASIC program which computed the same angles in floating point arithmetic, and thus the

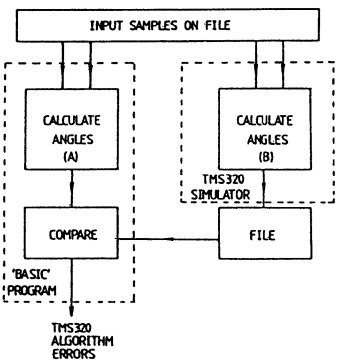


Fig.4 Testing TMS32010 arithmetic for angle calculation

errors in the TMS32010 program. This comparison showed that the worst case error was of the order of 0.03%. Had this been performed in the target hardware, the data (NP1 & m) would have been derived from ADCs and been neither exact nor repeatable. Speed tests conducted on the simulator show that the time to compute (1) on the TMS32010 is 24.8 microseconds (odd k) and for (2) (even k) is 32.4 microseconds with NP1 less than 0.8. The corrections for NP1 > 0.8 add 23.4 microseconds.

Simulating the waveform generation hardware

The TMS32010 simulator allows the "connection" of a simulated peripheral device usually written in 'C' (appendix A). In order to expedite testing of the complete TMS32010 software, a simulation of the waveform generation hardware was created and interfaced to the simulator. It allows testing which would be nearly impossible on the real hardware.

The waveform generation RAM is simulated by an array of integers updated entirely by the TMS32010 program which can be examined by the user or written to file. A count is kept of the number of times that the simulated RAM address passes through zero allowing output pulse widths to be computed and saved on file for test purposes. Note that digital values of long times (equivalent to more than 2048 RAM addresses) are not obtainable from the hardware. They could include errors introduced outside the angle computations e.g. due to the block floating point representation or failure to delete angles from the waveform generation RAM .

The speed of the TMS32010 program in performing the hardware updates is of course very dependant upon the type of waveform and the part of the cycle involved. However as an indication, at 17 Hz, simulation shows that 127 microseconds is necessary to update the hardware during the worst case RAM half cycle. This compares with 1.6 milliseconds which is available while the RAM cycles through 2048 addresses.

Performance in hardware

The functional integrity of the hardware was tested with small programs to excercise the various sections. Thus, by the time the software described above was transferred to the target system, the hardware was known to be capable of executing a TMS32010 program, generating a waveform from the waveform generation hardware and correctly reading the analog to digital converters used to input the dc link voltage and the inverter output frequency. Waveforms produced by this hardware on the first trial, agreed with expectations predicted by simulation.

Design Adaptability

The high speed of the TMS32010 has resulted in spare computing capacity which can be used in several ways.

Non-complementary waveforms

The system described so far can generate the three phase pole switching waveforms to drive a three phase inverter. In each phase, the complementary device can only be switched on at a finite time after the other device in the same phase has been switched off, thus avoiding a dc link short circuit. Therefore, for a definite time interval, both gate drive signals for a phase are off. This delay time td must be greater than the turn-off time tq of the GTO used. tq varies with the type of GTO and the anode current being commutated. Therefore, in this application, tq will vary depending on the point in the inverter current waveform at which the inverter current is turned off. However, for a given application, it is possible to predict the maximum value of tq , and td is set to a constant value which is slightly greater than this. Having derived the ideal complementary gate drive waveforms from the three PWM waveforms, the effect of td is to delay the turn-on edge of these gate drive waveforms and leave the turn-off edge unaltered as shown in Fig. 5.
This delay can be incorporated in hardware but, with this RAM based method of generating the waveforms, the delay can be incorporated in software. This reduces the hardware requirement especially if the delay must be variable. All four binary codes are used with two output latches per phase in the switching angle map as follows:

11 output latches remain unchanged
10 set output latch 1, reset latch 2
01 reset output latch 1, set latch 2
00 reset both latches

With this system, a typical switching sequence might be as follows:

10 device 1 on, device 2 off
11
00 both devices off
11
01 device 2 on,device 1 off

Of course twice as many RAM accesses need to be made but this is no problem with the TMS32010. A PAL e.g. the PAL16R6 is ideal for implementing the Boolean expressions to directly generate the 6 GTO gate drive signals from the RAM contents.

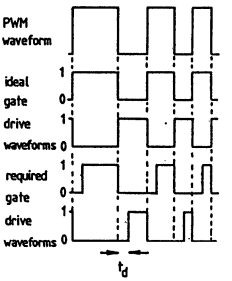


Fig.5 Generation of GTO gate drive signals

Due to the turn-off time tq , the GTO inverter pole switching waveform will be slightly different from the generated PWM waveform. This small error in the power electronics reproduction of these waveforms will result in a slight change in the harmonic spectrum measured in the power circuit compared with the ideal case. It is possible to compensate for this effect if the variation of tq with anode current is known [7]. The spare computing capacity on the TMS32010 means that it will be possible to incorporate a closed loop controller to compensate for the varying GTO turn-off time.

Practical Results

After extensive testing and debugging of the software using the simulation facility already described, the software was evaluated in the target hardware. Prior to interfacing the TMS32010 based waveform generator to a GTO inverter, the harmonic spectra of the near optimal PWM waveform produced were analysed. Fig.6 shows this ideal pole switching waveform spectrum for m = 5 and, as expected, the 5th, 7th, 11th and 13th harmonics (i.e. m-1 harmonics) are almost zero. This confirms the theoretical work carried out on evaluating the accuracy of the algorithm [1]. As explained in [1], the

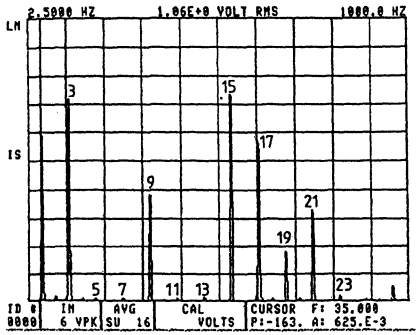
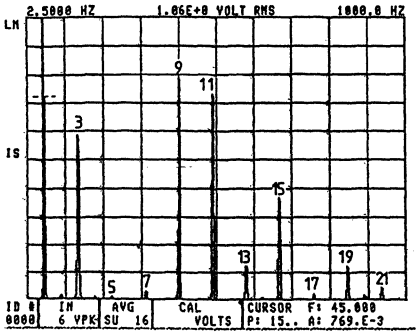
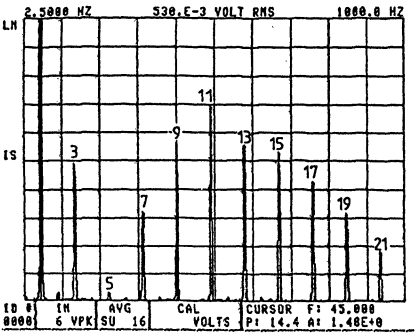


Fig.6 Ideal pole switching waveform spectrum for m = 5

algorithm is least accurate when m = 3, especially for high NP1 values. This is clearly demonstrated by Fig.7, which shows the effect of the increase in the value of NP1 as Vdc decreases to its minimum value.



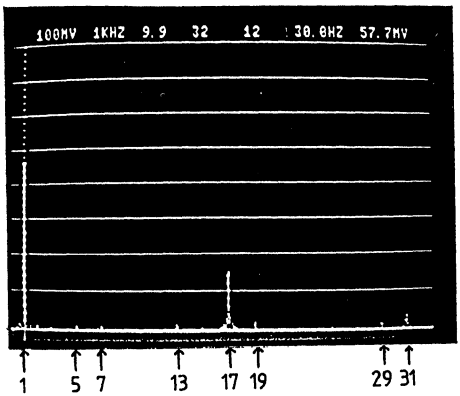
(a) Nominal Vdc



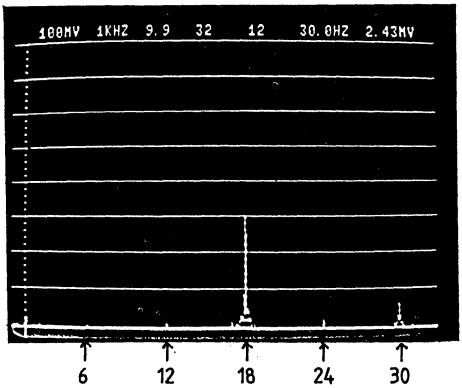
(b) Minimum Vdc

Fig.7 Ideal pole switching waveform spectra for m = 3 showing the effect of a decrease in Vdc

Using the algorithm, if the 7th harmonic in the m = 3 mode is unacceptably high in amplitude, then as explained in [1] the situation can be easily improved by using the exact switching angles instead. The TMS32010 based waveform generator was finally



(a) Line current spectrum



(b) Link current spectrum

Fig.8 Measured spectrum for m = 5

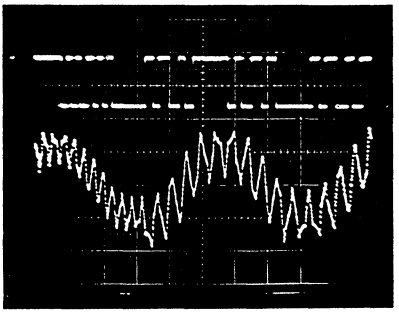
interfaced to a 2kVA GTO VSI driving a small induction motor. The measured line current spectrum for m = 5 is shown in Fig.8a and the first significant harmonic present is the 17th as expected. The corresponding inverter DC link current waveform spectrum is shown in Fig.8b. As anticipated, the 6th and 12th DC side harmonics are almost zero and the first main harmonic is the 18th due to the 17th and 19th AC side harmonics. Similarly accurate results were also obtained for higher values of m.

Further tests were carried out to investigate the transition during gear changes. Fig.9a & b show the inverter line current waveform during the transition from m = 5 to m = 3 and m = 1 to m = 0 (quasi-square) respectively. As can be seen, the transitions occur smoothly and there is no observable transient in the inverter line current waveform.

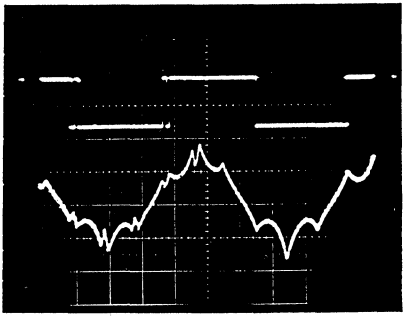
Conclusion

The use of a harmonic elimination optimized PWM ratio changing scheme is essential if railway traction VSI drives are to be compatible with signalling systems. In particular, it is shown that it would be advantageous

if the switching angles could be computed on-line by a generalized algorithm which gives near optimal



(a) Transition from m = 5 to 3



(b) Transition from m = 1 to 0

Fig.9 Inverter line current during ratio changes

switching angles. This paper shows that a high speed signal processing microprocessor can be used efficiently in implementing such an algorithm. The calculation time is extremely small when compared with conventional processors. The fast cycle time makes it desirable to use novel methods for waveform generation. The one described gives exceptional waveform control and a low chip count.

The use of an unusual TMS32010 simulator has not only allowed the testing of TMS32010 algorithms using integer arithmetic but also enabled the exact pulse width output values to be analysed. Due to the unusual output stage, this would have been difficult in real hardware. It has allowed complete debugging of the software before the final implementation.

In this implementation, the computation of each even or odd switching angle takes 24.8 μs or 32.4 μs respectively. When the required fundamental pole switching amplitude is greater than 0.8Vdc PI/2, the correction factor equations add a further 23.4 μs.

References

[1] J.A. Taufiq, B. Mellitt & C.J.Goodman "A Novel algorithm for generating near optimal PWM waveforms for ac traction drives", Proc. IEE vol 133,Pt B, no. 2, pp 85-94 Mar.1986.

[2] J.A. Taufiq, C.J. Goodman & B.Mellitt "Railway signalling compatibility of inverter fed induction motor drives for rapid transit", Proc. IEE vol 133, no. 2,Pt B, pp 71-84.Mar 1986.

[3] S.R. Bowes & M.J. Mount "Microprocessor control of PWM inverters", IEE PROC., vol 128, PT B, no. 6 pp 293-305,Nov. 1981.

[4] A.V. Oppenheim "Realisation of Digital filters using Block floating point arithmetic", IEEE Trans. Audio Electroacoust. vol. AU-18,pp. 130-136, June 1970.

[5] R.J. Chance " Simulation Experiences in the development of software for digital signal processors" Microproc. and Microsys, Vol 10 No 8 pp. 419-426, 1986.

[7] S. Sone , H.Irinatsu "Very precise turn-off timing control of gate turn-off thyristors", IEE PEVD conf. London, pp. 23-26,1-4 May 1984.

[8] R.J. Chance "TMS320 digital signal processor development system", Microprocessors and Microsystems, vol. 9 no. 2, pp 50-56, Mar 1985.

[9] R.J. Chance & B.S.Jones " A Combined Software /Hardware Development tool for the TMS32020 Digital Signal Processor" J. Micro. App., Vol 10, pp 179-197,1987.

Appendix A The TMS32010 Simulator

The Simulator used in this work [5,8] is part of a TMS32010/20/C25 development system written by one of the authors (RJC). It includes a TMS32010 assembler and simulator the normal host being an IBM Personal Computer. The simulator accepts machine code created by the assembler and allows simulated execution of TMS32010 programs. The usual facilities such as break point setting, access to user symbols, instruction timing etc. are provided. This simulator is particularly intended to be used for linking digital data streams to TMS32010 i/o ports or memory for test purposes. One advantage is that values are precise digital values rather than analog signals and are thus repeatable and accurate. In addition, powerful software tools on the host may be easily used to generate or analyse i/o data.

The use of such a simulator would be limited without the ability to simulate essential peripheral hardware e.g. the waveform generation RAM. This simulator is supplied in object library format. The user may create a software simulated peripheral 'device', to be linked at the object code level to the TMS320 simulator. Such a simulated peripheral is usually written in 'C'. This enormously extends the use of the simulator and allows debugging methods impractical in hardware such as the trapping of complex i/o data. Simulated peripherals have been used not only to allow the use of real hardware and imaginary hardware used only for testing. The simulated peripheral concept has also been used [9] in multiple TMS320 simulation, program execution in a mixed real/simulated environment and simulator verification.

DESIGN AND IMPLEMENTATION OF AN EXTENDED KALMAN FILTER FOR THE STATE ESTIMATION OF A PERMANENT MAGNET SYNCHRONOUS MOTOR

Rached Dhaouadi, St. member, IEEE
Ned Mohan, Member, IEEE
Lars Norum Member, IEEE

Dept. of Electrical Engineering
University of Minnesota
Minneapolis, Minnesota 55455

Dept. of Electrical Engineering
Norwegian Institute of Technology
Trondheim, Norway

ABSTRACT:

This paper discusses practical considerations for implementing the discrete extended Kalman filter in real time with a digital signal processor.

The system considered is a Permanent Magnet Synchronous Motor (PMSM) without a position sensor, and the extended Kalman Filter is designed for the on-line estimation of the speed and rotor position by only using measurements of the motor voltages and currents.

The algorithms developed to allow efficient computation of the filter are presented. The computational techniques used to simplify the filter equations and their implementation in fixed-point arithmetic are discussed. Simulation and experimental results using the TMS 320C25 digital signal processor are presented to demonstrate the feasibility of this estimation process.

1. INTRODUCTION:

High performance motion control systems need more computing power than today's standard microprocessors are capable of delivering; sophisticated control laws such as observer based schemes or Kalman filtering in real time require a very fast signal processor specialized and optimized to perform complex mathematical calculations and manipulate large amounts of data.

The extended kalman filter algorithm is an optimal recursive estimation algorithm for nonlinear systems. It processes all available measurements regardless of their precision, to provide a quick and accurate estimate of the variables of interest, and also achieves a rapid convergence. This is done using the following factors:

- A knowledge of the system and measurement device dynamics.
- The statistical description of the system noises, disturbances, measurement errors, and uncertainties in the system model.
- Any available information about the initial conditions of the variables of interest.

The algorithm is computationally intensive, and all of the steps involved require a vector or a matrix operation. Therefore, an efficient formulation of the algorithm needs to be made rather than a straightforward implementation. Moreover, for a practical application of the filter in real time, different aspects of implementation have to be addressed: Among these aspects are the computational requirements for the filter and the constraints imposed by the computer used.

The computational requirements include mainly the computation time per filter cycle and the required memory storage. Knowledge of these quantities in advance will enable the choice of a meaningful data sampling rates and the required memory size for the system. The constraints of the computer to be used are defined by its speed (cycle execution time), its calculation capability (instruction set), the type of arithmetic used (fixed point or floating point), and its wordlength (16-bit or 32-bit).

The extended Kalman filter approach is ideally suited to the state estimation of a Permanent Magnet Synchronous Motor (PMSM). It appears to be a viable and computationally efficient candidate for the on-line estimation of the speed and rotor position. This is possible since a mathematical model, describing the motor dynamics is sufficiently well known. The terminal quantities like voltages and currents can be measured easily and are suitable for the determination of the rotor position and speed in an indirect way.

The paper is organized in eight sections. Section 1 is an introduction. In sections 2 and 3, the state space model of the PMSM is developed, and the extended Kalman filter algorithm is presented. Using these formulations, the computational requirements of the filter and its implementation with fixed point arithmetic are discussed in sections 4 and 5. The results and the practical aspects of implementation are discussed in sections 6 and 7. Section 8 has the conclusion and the future research on the subject.

2. SYSTEM MODEL:

The system considered is a permanent magnet synchronous motor having permanent magnets mounted on the rotor, and a sinusoidal flux distribution. A dynamic model for this motor in a stator-fixed reference frame (α,β), by choosing the current components i_a, i_b, the rotor speed ω_r, and the rotor position θ_r as state variables is as follows:

$$\frac{d}{dt} i_\alpha = -\frac{R_s}{L_s} i_\alpha + \frac{\Phi_f}{L_s} \omega_r \sin(\theta_r) + \frac{V_\alpha}{L_s} \quad (1)$$

$$\frac{d}{dt} i_\beta = -\frac{R_s}{L_s} i_\beta - \frac{\Phi_f}{L_s} \omega_r \cos(\theta_r) + \frac{V_\beta}{L_s} \quad (2)$$

$$\frac{d}{dt} \omega_r = \frac{3}{2} \frac{\Phi_f}{J} (i_\beta \cos(\theta_r) - i_\alpha \sin(\theta_r)) - \frac{B}{J} \omega_r - \frac{T_L}{J} \quad (3)$$

$$\frac{d}{dt} \theta_r = \omega_r \quad (4)$$

where,

R_S: stator per-phase resistance
L_S: stator per-phase inductance
Φ_f: permanent magnet flux linkage
J: rotor moment of inertia
B: viscous damping

The voltage components V_α, V_β, and the average load torque T_L are the deterministic control inputs of the system. Both the voltage and current components are measurable quantities. They are obtained from the three phase stator components by a linear transformation:

$$i_\alpha = \frac{2}{3} (i_a - \frac{i_b}{2} - \frac{i_c}{2}) \; ; \quad (5)$$

$$i_\beta = \frac{(i_b - i_c)}{\sqrt{3}} \quad (6)$$

Similar equations hold for the voltages.

To summarize, the system is driven by the stator voltages V_α, V_β and the resulting outputs are the stator currents i_α, i_β. The state space model [Eqs. 1 through 4] is nonlinear due to the cross product of the state variables ω_r, i_α, i_β and θ_r.

The motor parameters used are listed in Appendix A of the paper.

3. THE EXTENDED KALMAN FILTER ALGORITHM:

The Filter algorithm can be summarized as follows [9]: Let the system of interest be described by the nonlinear dynamic state space model

$$\dot{X}(t) = f[X(t),U(t),t] + w(t) \quad (7)$$

where the initial state vector $X(t_0)$ is modeled as a gaussian random vector with mean X_0 and covariance P_0, U(t) is the deterministic control input vector, and w(t) is a zero-mean white gaussian noise independent of $X(t_0)$, and with a covariance matrix Q(t).

Let the available discrete-time measurements be modeled as:

$$Y(t_i) = h[X(t_i),t_i] + v(t_i) \quad (8)$$

where $v(t_i)$ is a zero-mean white gaussian noise that is independent of $X(t_0)$ and w(t), and with a covariance matrix $R(t_i)$.

The optimal state estimate $\hat{X}(t)$ generated by the filter is a minimum variance estimate of X(t), and is computed in a recursive manner as shown in Fig.1. The filter has a predictor-corrector structure as follows (superscripts - and + refer to the time before and after the measurements have been processed):

STEP 1: Prediction (from t_{i-1}^+ to t_i^-)

The optimal state estimate $\hat{X}$ and the state covariance matrix P are propagated from measurement time (t_{i-1}) to measurement
time (t_i), based on the previous values, the system dynamics, and the previous control inputs and errors of the actual system. This is done by numerical integration of the following equations:

$$\dot{\hat{X}}(t) = f(\hat{X}(t), U(t), t) \quad (9)$$

$$\dot{P}(t) = F^T * P(t) + P(t) * F + Q \quad (10)$$

$$t \in [t_{i-1}^+, t_i^-]$$

starting from the initial conditions: $\hat{X}(t_{i-1}^+)$, $P(t_{i-1}^+)$

where: $$F = \frac{\partial f(X,U,t)}{\partial X} \quad (11)$$

evaluated at $X = \hat{X}(t_{i-1}^+)$

STEP 2: Filtering (from t_i^- to t_i^+)

By comparing the measurement vector, Y, to the estimated one, $\hat{Y}$, a correction factor is obtained and is used to update the state vector.

The filter gain matrix $K(t_i)$ is defined as:

$$K(t_i) = P(t_i^-)*H^T*[H*P(t_i^-)*H^T + R(t_i)]^{-1} \quad (12)$$

where: $$H(t_i) = \frac{\partial h(X,t_i)}{\partial X} \quad (13)$$

evaluated at $X = X(t_i^-)$

The measurement update equations for the state vector and the covariance matrix are:

$$X(t_i^+) = X(t_i^-) + K(t_i)*\{ Y(t_i) - h[X(t_i^-),t_i] \} \quad (14)$$

$$P(t_i^+) = P(t_i^-) - K(t_i)*H(t_i)*P(t_i^-) \quad (15)$$

where, $X(t_i^+)$ represents the optimal state vector estimate.

4. COMPUTATIONAL REQUIREMENTS:

The objective of the design presented in this paper is to minimize the filter cycle time, while obtaining a reasonable accuracy in the filter equations implementation. The method used for the numerical integration of Eq. 9 from one sample time to the next is the first order Euler integration technique:

$$X(t_i^-) = X(t_{i-1}^+) + Ts*f[X(t_{i-1}^+),U(t_{i-1})] \quad (16)$$

where, $T_S = t_i - t_{i-1}$

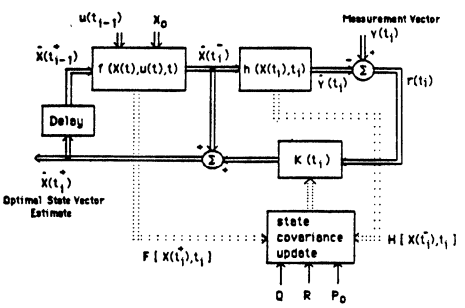


Fig. 1 Block diagram of the extended Kalman filter

In order to achieve a reasonable accuracy, the integration step size which is the sampling period T_S should be appreciably smaller than the characteristic time constants of the system. The choice of the sampling time T_S should be made to meet both the total computation time of the filter and a reasonable integration accuracy. Integration accuracy can be improved by using a second order integration technique, or by dividing the interval $[t_{i-1}, t_i]$ into N subintervals and applying a first order Euler integration technique to each subinterval. This however will result in increased computation time.

The time propagation equation for the state covariance matrix P, (Eq.10), can be solved using the transition matrix technique [9]. This method preserves both the symmetry and the positive definiteness of P, and yields adequate performance:

$$P(t_i^-) = \Phi(t_i,t_{i-1})*P(t_{i-1}^+)*\Phi^T(t_i,t_{i-1}) + Q_d(t_i,t_{i-1}) \quad (17)$$

where,

$$Q_d(t_i,t_{i-1}) = \int_{t_{i-1}}^{t_i} \Phi(t_i,\tau)*Q(\tau)*\Phi^T(t_i,\tau)\, d\tau \quad (18)$$

$\Phi(t_i,\tau)$ denotes the state transition matrix associated with $F(\tau,X(\tau))$ for all $\tau \in [t_{i-1},t_i)$.

$Q_d(t_i,t_{i-1})$ is next evaluated using a trapezoidal integration:

$$Q_d(t_i,t_{i-1}) = [\Phi(t_i,t_{i-1})*Q*\Phi^T(t_i,t_{i-1}) + Q]* \frac{T_s}{2} \quad (19)$$

This form is attractive since it replaces having to know $\Phi(t_i,\tau)$ for all t, by evaluating only $\Phi(t_i,t_{i-1})$.

$$\Phi(t_i,t_{i-1}) = I + F[t_{i-1},X(t_{i-1}^+)]*T_S \quad (20)$$

Clearly, all of the steps involved above require a vector or a matrix operation. These operations consist largely of multiply or multiply-accumulates. Moreover, all these computations must be performed within one sampling interval of the system. This therefore motivates the need for a very fast signal processor with dedicated arithmetic unit and instruction set.

Table 1 shows the different steps and the number of operations needed for the filter computation. The defining equations for the filter can be programmed as shown under the column labeled "computation". The total number of multiplications, additions and divisions for each computation are also listed.

The total computation time of the filter is equal to the total execution time of all multiplications, additions and divisions, plus the total logic time which is the execution

time of the additional steps required for properly controlling and sequencing the different operations. The logic time is very sensitive to the order of the system's model, and can be substantially important since most of the operations include either a matrix multiplication or a matrix addition .

5. IMPLEMENTATION WITH FIXED_POINT ARITHMETIC:

This section presents the methods of solving the Kalman Filter equations, in a computationally efficient manner using fixed-point arithmetic. The microprocessor considered for this application is the TMS 320C25 digital signal processor. It is a 16-bit microcontroller specifically designed and optimized for high speed processing, and is shown to be well suited for high performance control applications.

The dynamic range in fixed point arithmetic with a 16-bit word length is from 2^{-15} to 1. Therefore, to avoid the overflow and underflow problems, all variables in the filter equations (Eq. 9 - 15) must be scaled to values less than one.

The state variables in Eqs. 1 through 4 are scaled with respect to their maximum values. This results in scaled differential equations where all variables are normalized. The subscript n denote a normalized variable:

$$\frac{d}{d\tau} i_{\alpha n} = -\frac{R_{sn}}{X_{sn}} i_{\alpha n} + \frac{\phi_{fn}}{X_{sn}} \omega_{rn} \sin(\theta_r) + \frac{1}{X_{sn}} V_{\alpha n}$$

$$\frac{d}{d\tau} i_{\beta n} = -\frac{R_{sn}}{X_{sn}} i_{\beta n} + \frac{\phi_{fn}}{X_{sn}} \omega_{rn} \cos(\theta_r) + \frac{1}{X_{sn}} V_{\beta n}$$

$$\frac{d}{d\tau} \omega_{rn} = \frac{3}{2} \cdot \frac{\phi_{fn}}{J_n} \cdot (i_{\beta n} \cos(\theta_r) - i_{\alpha n} \sin(\theta_r))$$

$$- \frac{B_n}{J_n} \omega_{rn} - \frac{T_{ln}}{J_n}$$

$$\frac{d\theta_m}{dt} = \frac{\omega_{rn}}{2\pi}$$

where, $\tau = \omega_n . t$, is the normalized time, and ω_n is the normalizing frequency.

The scaling factors of the covariance matrices were determined through computer simulation, by looking at the maximum values of the matrices elements for different simulation runs.

However, the maximum values of the gain matrix elements were found to have a large dynamic range and were difficult to predict. To avoid this scaling problem, the solution used was to update the state vector and the state covariance matrix directly, without explicitly computing the gain coefficient matrix K[1]. The measurement update equations [Eqs. 14 and 15] are therefore transformed and expressed as follows:

$$X(t_i^+) = X(t_i^-) + A*B^{-1}*\tilde{Y} \qquad (21)$$

$$P(t_i^+) = P(t_i^-) - A*B^{-1}*A^T \qquad (22)$$

where,

$$A = P(t_i^-)*H^T \qquad (23)$$

$$B = H*A + R \qquad (24)$$

$$\tilde{Y} = Y(t_i) - H*X(t_i^-) \qquad (25)$$

This formulation resulted in a simpler scaling procedure and a greater numerical precision.

6. FILTER TUNING:

The critical step in a Kalman filter design is to obtain a numerical evaluation of the filter parameters specified by the initial state X_0 and the covariance matrices P_0, Q and R. This process is called tuning and it involves an iterative search for the coefficient values that yield the best estimation performance possible.

The noise covariance Q accounts for the model inaccuracy, the system disturbances and, the noise introduced in the voltage measurements (sensor noise, A/D converters quantization). The rounding and truncation errors in the computations due to the fixed word length of the processor can corrupt the filter performance, and are considered as additional sources of system noise. The noise covariance R on the other hand, reflects the measurement noise introduced by the current sensors, and the coding effects of the A/D converters.

Changing the covariance matrices Q and R affects both the transient duration and the steady state operation of the filter. Increasing Q would indicate either stronger noises driving the system or increased uncertainty in the model. This will increase the values of the state covariance elements. The filter gains will also increase thereby weighting the measurements more heavily, and the filter transient performance is faster. Similarly, increasing the covariance R indicates that the measurements are subjected to a stronger corruptive noise and should be weighted less by the filter. Consequently the values of the gain matrix K will decrease, and the transient performance is slower.

For the initial state covariance matrix P_0, the diagonal terms represent variances or mean squared errors in knowledge of the initial conditions. Varying P_0 yields a different magnitude transient characteristic. The transient duration will be the same and the steady state conditions are unaffected.

Table 1

Variable	Defining equation	Computation	# of Mult	# of Add	# of Div
$X(t_i^-)$	$X(t_i^+) + T_s*f(X(t_i^+))$		14	11	3
F	$\frac{\partial f}{\partial X}$		14	1	
Φ	$I + T_s*F$		16	4	
Q_d	$(\Phi*Q*\Phi^T+Q)*\frac{T_s}{2}$	$Q*\Phi^T$	16	0	
		$\Phi*(Q*\Phi^T)$	64	48	
		$\Phi*(Q*\Phi^T) + Q$	0	4	
		$(\Phi*Q*\Phi^T+ Q)*\frac{T_s}{2}$	16	0	1
$P(t_i^-)$	$\Phi*P(t_i^+)*\Phi^T + Q_d$	$P(t_i^+)*\Phi^T$	64	48	
		$\Phi*(P(t_i^+)*\Phi^T)$	64	48	
		$\Phi*P(t_i^+)*\Phi^T + Q_d$	0	16	
K(ti)	$P(t_i^-)*H^T*(H*P(t_i^-)*H^T + R)^{-1}$	$P(t_i^-)H^T$	32	24	
		$H*(P(t_i^-)*H^T)$	16	12	
		$H*P(t_i^-)*H^T + R$	0	2	
		$[H*P(t_i^-)*H+T + R]^{-1}$	2	1	4
		$P(t_i^-)*H^T*[.]^{-1}$	16	8	
		$H*X(t_i^-)$	8	6	
		$Y - H*X(t_i^-)$	0	2	
$X(t_i^+)$	$X(t_i^-) + K*(Y-H*X(t_i^-))$	$K*(Y-H*X(t_i^-))$	8	4	
		$X(t_i^-) + K*(.)$	0	4	
$P(t_i^+)$	$(I - K*H)*P(t_i^-)$	$K*H$	32	16	
		$I - K*H$	0	4	
		$(I - K*H)*P(t_i^-)$	64	48	
Total			446	311	8

The covariance matrices Q, R and P_0 are assumed to be diagonal for the lack of sufficient statistical information to evaluate their off-diagonal terms. In the following simulation, the best filter performance was obtained with:

$Q = 0.02*I(4)$; $P_0 = 0.01*I(4)$; $R = 0.1*I(2)$

where, I(2) and I(4) are the 2x2 and 4x4 identity matrices.

7. SIMULATION AND EXPERIMENTAL RESULTS:

Simulation results:

The filter algorithm was first simulated to get all the influences of the system parameters on the filter performance. The computer program developed simulates fixed-point arithmetics with a 16-bit word length. The control input voltages and motor currents are also simulated. They are assumed to be real-time measured values obtained from a PMSM running in steady state, as shown in Fig. 2. A random noise was added to the currents to simulate the measurement noise.

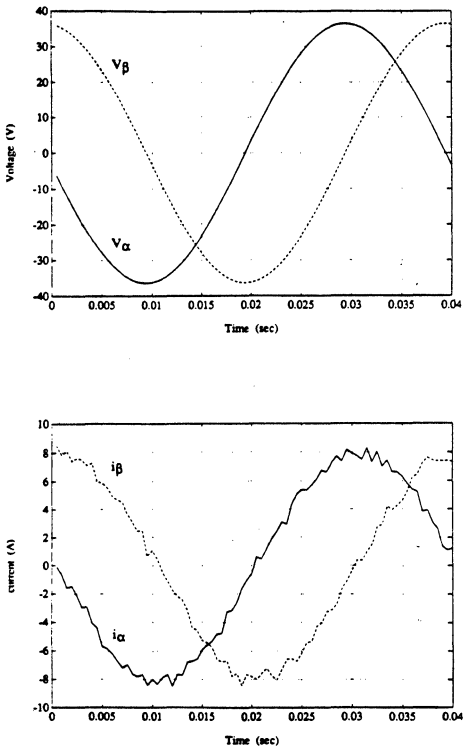


Fig. 2 Simulated voltages and currents for the PMSM running at constant speed (1500 rpm).

The filter starts from rest with the motor already in steady state. The initial values of the currents are assumed to be known to the filter and are set equal to the initial measured values. The starting value of the speed was set to zero, and the simulation starts when the actual rotor position reaches $\theta_r = 0$. The initial position used by the filter was used as a variable.

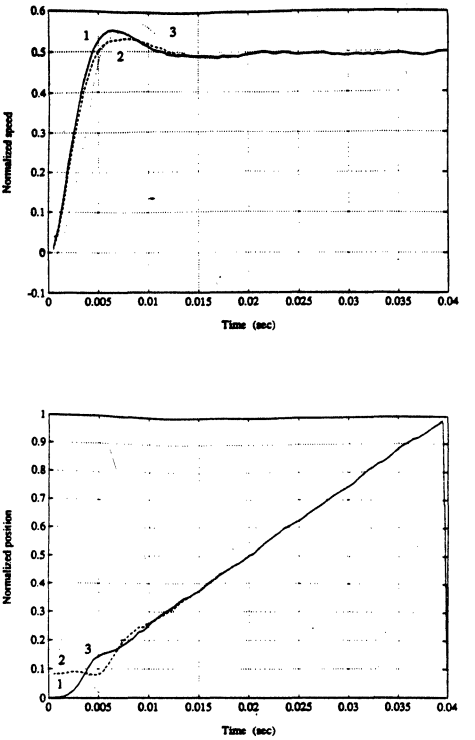


Fig. 3 Transient and steady state behavior of the estimated speed and rotor position. 1) $\theta_r(0) = 0$ deg. 2) $\theta_r(0) = 30$ deg. 3) $\theta_r(0) = 60$ deg.

Figure 3 shows the behaviour of the estimated values of the speed and the rotor position.The actual steady state normalized speed is equal to $\omega_{rn} = 0.5$. It can be seen that even though the initial speed used by the filter is $\omega_r(0) = 0$, the estimated speed follows closely the actual speed after an initial transient. Similarly, the estimated position go through a transient and then converges to follow the actual position. The magnitude and duration of the transient and the steady state performance are adjusted by the values of

the covariance matrices Q, R and P_0. The transient duration is about 15 msec.

It is clear that the Extended Kalman filter tracks very well the speed and rotor position of the motor. The precise modeling of the system, and a good estimate of the initial conditions will improve further the performance of the filter.

Experimental results:

Implementation of the Kalman filter in real time was carried out using the TMS 320C25 digital signal processor. The system hardware (data acquisition system) and software was developed and tested using the XDS/22 Hardware Emulator and its supporting program tools.

The total filter algorithm was performed in (284 msec). Table 2 lists the various execution times for the different steps involved in the filter computation. The processing time of the measurement vector $Y(t_i)$ and the control input vector $U(t_{i-1})$ is not included in the filter computation. These variables are assumed to be available to the filter at no computing expense. Clearly, the largest execution time is taken by the covariance matrix computations. This computation time can be further reduced by computing only the lower triangular form of the symmetric matrices.

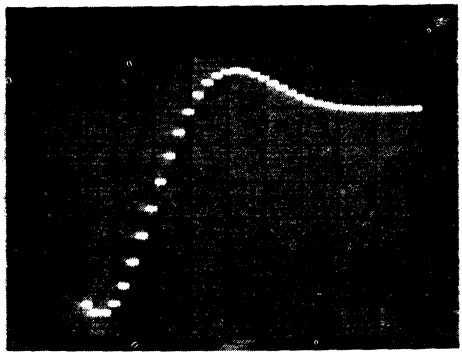
Table 2

Operation	Execution time in μsec
Compute the cosine of the position angle	7.2
Compute the sine of the position angle	5.3
Compute the transition matrix	7.3
Time propagation of the state vector	4.4
Time propagation of the state covariance matrix	99.0
Measurement update of the state vector and the state Covariance matrix	160.7
Total	283.9

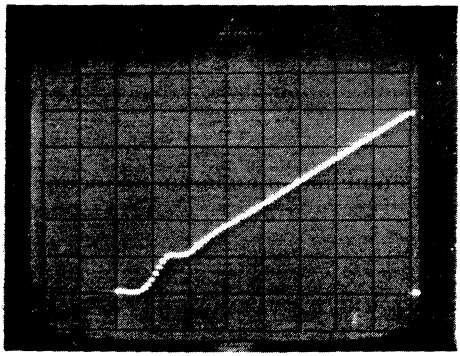
The filter can operate in a system having a maximum sampling frequency of 3.52 kHz, or a theoretical system bandwidth of 1.76 kHz. This high bandwidth allows the Extended Kalman Filter implemented on the TMS 320C25 to be used in high performance real-time motion control systems.

Figure 4 shows the behaviour of the estimated values of the speed and the rotor position, using numerically simulated currents and voltages waveforms. These results are comparable to the off-line simulation results in Fig. 3.

The above results show that the extended Kalman filter can be efficiently implemented in real time to estimate the speed and rotor position of the PMSM.



a)



b)

Fig. 4 Experimental results for the estimated speed and rotor position with zero initial conditions.
a) Speed: 300 rpm/div, Time: 2 msec/div
b) Position: 72 deg/div, Time: 5 msec/div.

8. CONCLUSION:

In this paper, the design and implementation of an extended Kalman filter with a digital signal processor has been investigated. A systematic and analytic approach for developing the algorithm was presented. The computational techniques used to simplify the filter equations and their implementation in fixed-point arithmetics are discussed. The filter was tuned by varying the parameters Q, R, P_0, X_0, to meet the desired transient and steady state performance. The discrete Extended Kalman Filter have been found to be well suited to the speed and rotor position estimation of a Permanent Magnet Synchronous Motor. The proposed approach has been validated using computer simulation and actual implementation in real time with the TMS320C25 digital signal processor.

The next step in this research project will be to test the filter will the actual currents and voltages of a PMSM drive system, and use the estimated position for the control of the PMSM instead of a position sensor. This will be reported in a following paper.

ACKNOWLEDGMENT

The authors wish to thank Texas Instruments Inc. for providing the TMS320C25 development tools. The financial support of this project by the University of Minnesota Center for Electric Energy is gratefully acknowledged.

APPENDIX A

Motor parameters:
Maximum speed = 3000 rpm
Rated torque = 2.2 N.m
$R_S = 0.7\ \Omega$
$L_S = 5$ mH
$\Phi_f = 0.193$ V.sec/rad
$J = 9 \cdot 10^{-5}$ kg.m^2
$B = 0$

REFERENCES:

[1] J. C. Wauer, "Practical Considerations in Implementing Kalman Filters", AGARD-LS-82, NATO Advisory Group for Aerospace Research and Development, London, May 1976. pp. 2.1-2.11.

[2] J. M. Mendel, "Computational Requirements for a discrete Kalman Filter", IEEE Transactions on Automatic Control, Vol. AC-16, No.6, December 1971, pp. 748-758.

[3] U. Kirkberg, Ph. K. Sattler, "State Estimation of an Inverter Fed Synchronous Motor", European Conference on Power Electronics and Applications, Brussels, October 1986, pp. 3.229 - 3.234.

[4] B. Gallwitz, F. Hillenbrand and Ch. Landgraph, "A Proposal For Avoiding The Direct Measurement of Speed And Angular Position of The Synchronous Machine", IFAC Control in Power Electronics and Elect. Drives, Lausanne, Switzerland,1983, pp. 63-68.

[5] A. M. N. Lima, B. de Fornel, Mrs. Pietrzak-David, "On Stochastic Filtering Techniques and its Applications to AC Numerical Drive Systems", Proceeding of the EPE, 1987, pp. 683-688.

[6] J. Tan and N. Kyriakopoulos, "Implementation of a Tracking Kalman Filter on a Digital Signal Processor", IEEE Transactions on Industrial Electronics, Vol. 35, No. 1, February 1988.

[7] J. Sethuram, David Squires, "Application of Digital Signal Processors in Motion Control", Conference on Applied Motion Control, June 1987.

[8] C. T. Leondes (ed.), "Theory and Application of Kalman Filtering", AGARDograph No. 139, NATO Advisory Group For Aerospace Research and Development, London, Feb. 1970.

[9] P. S. Maybeck, "Stochastic Models, Estimation and Control", Vol. 1 and Vol. 2, Academic Press, New york, 1982.

[10] W. Leonhard, "Control of Electrical Drives", Springer-Verlag, Berlin, 1985.

[11] Texas Instruments Inc., "Digital Signal Processing Applications with the TMS320 Family", 1986.

[12] Texas Instruments Inc., Second Generation TMS320 User's Guide, 1987.

[13] Texas Instruments Inc., XDS/22 TMS320C2x Emulator User's Guide, 1987.

TRENDS OF DIGITAL SIGNAL PROCESSING IN AUTOMOTIVE

KUN-SHAN LIN
Microprocessor Microcontroller Products Division
Texas Instruments Inc.

ABSTRACT

The advent of single-chip programmable digital signal processors (DSP) has expanded digital signal processing into automotive applications. Digital signal processors, compatible in cost to general-purpose microcomputers, offer much higher throughput in performing computationally intensive tasks. Because of this advantage, closed loop control, adaptive control, and digital audio processing can now be implemented in a cost effective manner. Some of the DSP automotive applications include combustion feedback engine control, active suspension systems, anti-skid brakes incorporating traction control and digital audio-based entertainment systems. The availability of digital signal processors is changing many aspects of automotive designs. Early adopters of this technology for innovative automotive products will enjoy leadership and financial benefits over their competition. The impact of DSP to the automobile has just begun and will continue beyond the year 2000 (1). This new technology has also presented tremendous challenges to both automotive and semiconductor industries.

This paper first discusses digital signal processing characteristics. After reviewing historical digital signal processing solutions, the paper focuses on single-chip programmable digital signal processors and compares their architectural designs to general-purpose microprocessors and microcontrollers. Deficiencies of early digital signal processors are discussed and trends of DSP are explored. Performance and cost benefits of using DSP in digital control systems are explained. Automotive applications of digital signal processors are discussed in areas of powertrain, body and chassis control, and entertainment systems. The last part of the paper discusses the challenges presented to both automotive industry and semiconductor vendors.

CHARACTERISTICS OF DIGITAL SIGNAL PROCESSING

Digital signal processing is concerned with the representation of signals by sequences of numbers, and the transformation, processing or controlling of such signal representations by numerical computation procedures. Digital signal processing encompasses a broad spectrum of applications. Some application examples include digital filtering, speech coding, image processing, spectral analysis, radar signal processing, robotic control, and missile guidance. The recent development of programmable single-chip digital signal processors has further expanded the field of DSP applications into high volume consumer products: digital audio, consumer toys, and automotives.

These applications and those considered digital signal processing (2-10) have several characteristics in common:

-Mathematically intensive algorithms,
-Realtime operation,
-Sampled data implementation, and
-System flexibility.

Let's illustrate these characteristics in the following paragraphs:

Mathematically Intensive Algorithms

A common DSP equation that has to be computed (and often repeatedly computed in each time critical loop) takes the form of (5-10):

$$y(n) = \sum_{i=0}^{N-1} a(i) * x(n-i) + \sum_{k=1}^{M} b(k) * y(n-k)$$

where y(n) = present output,
y(n-k) = past outputs,
x(n) = present input,
x(n-i) = past inputs, and
a(i), b(k) = weighting factors.

This equation basically says that any output y can be computed as a weighted

sum of the input at the present time n, past inputs x(n-i), and past outputs y(n-k). Terms a(i) and b(k) are the weighting factors. If output y is independent of the past outputs, a simplified version of the equation can be obtained:

$$y(n) = \sum_{i=0}^{N-1} a(i) * x(n-i)$$

$$= a(0) * x(n) + a(1) * x(n-1) + \ldots + a(N-1) * x(n-N+1)$$

In digital signal processing terminology, this is the general form for Finite Impulse Response (FIR) filter and also the convolution of two sequences of numbers, a(i) and x(i). Both FIR filtering and convolution are fundamental to digital signal processing. They also have some physical significance. For example, an FIR filter is a common technique used to eliminate the erratic nature of stock market prices. When the day-to-day closing prices are plotted, it is sometimes difficult to obtain the desired information, such as the trend of the stock, because of the large variations. A simple way of smoothing the data is to calculate the average closing values of the previous five days. For the new average value each day, the oldest value is dropped and the newest value added. Each daily average value would be the sum of the weighted value of the latest five days, where the weighting factor is 1/5. In equation form, the stock average value is determined by:

$$\text{average}(n) = \frac{1}{5} * x(n) + \frac{1}{5} * x(n-1) + \frac{1}{5} * x(n-2) + \frac{1}{5} * x(n-3) + \frac{1}{5} * x(n-4)$$

where x(n-i) is the daily stock closing price for the (n-i)th day. This equation assumes the same form as the FIR filter and sometimes is referred to as the moving average.

A digital signal processor has to be optimized to quickly compute N multiplications and additions or sums of products as indicated in the above equations. This capability is enhanced with DSP instructions, such as multiply (MPY), addition (ADD), and multiply and accumulate (MAC). Furthermore in recent DSP, each of these instructions can be executed in a single machine cycle.

Realtime Processing

In addition to being mathematically intensive, DSP algorithms must be performed in realtime. Realtime can be defined as a process that is accomplished by the DSP without creating a delay noticeable to the user. In the stock market example, as long as the new average value can be computed prior to the next day when it is needed, it is considered to be completed in realtime. In digital signal processing applications, processes happen faster than on a daily basis. In the FIR filter example, the sum of products must be computed usually within hundreds of microseconds before the next sample comes into the system. A second example is in a speech recognition system where a noticeable delay between a word being spoken and being recognized would be unacceptable and not considered realtime. Another example is in image processing, where it is considered realtime if the processor finishes the processing within the frame update period. If the pixel information cannot be updated within the frame update period, problems such as flicker, smearing, or missing information will occur.

Because of this realtime requirement, a digital signal processor often implements DSP functions, such as MPY, ADD and MAC, with on-chip hardware, rather than software or microcode as in general-purpose microprocessors and microcomputers. This hardware intensive approach allows most of the DSP operations to be executed in single machine cycles. To further increase the processor capability for realtime processing, multiple instructions are often being executed in parallel, revealing a high degree of parallelism.

Sampled Data Implementation

The application must be capable of being handled as a sampled data system in order to be processed by digital processors, such as digital signal processors. The stock market is an example of a sampled data system. That is, a specific value (closing value) is assigned to each sample period or day. Other periods may be chosen, such as hourly prices or weekly prices. In FIR filter, the output y(n) is calculated to

be the weighted sum of the previous N inputs. In other words, the input signal, x(n), is sampled at periodic intervals (1 over the sample rate), multiplied by weighting factor, a(i), and then added together to give the output result of y(n). Examples of sample rates for some typical sampled data applications (2-5) are shown in Table 1.

Table 1. Sample Rates vs. Applications

Application	Normal Sample Rate
Control	1 KHz
Telecommunications	8 KHz
Speech Processing	8-10 KHz
Audio Processing	40-48 KHz
Video Frame Rate	25-30 Hz
Video Pixel Rate	14-18 MHz

In a typical DSP application, the processor must be able to effectively handle (input, output, and store) sampled data in large block quantity and also perform arithmetic computations on these data in realtime. Note that the higher the sample rate required by the application, the more demand on the processor throughput to meet the realtime requirement.

System Flexibility

The design of the digital signal processing system must be flexible enough to allow improvements in the state-of-the-art. We may find out after several weeks of using the average stock price as a means of measuring a particular stock's value that we need to adapt our methods to get better results. Some of the adaptations may include a different method of obtaining the daily information, different daily weightings, a different number of periods over which to average, and a different procedure for calculating the result. Enough flexibility in the system must be available to allow for these variations. In many DSP applications, techniques are still in the developmental phase, and therefore the algorithms tend to change over time. As an example, speech recognition is presently an inexact technique requiring continual algorithmic modification. From this example we can see the need for system flexibility so the DSP algorithm can be updated.

A programmable DSP system can provide this flexibility to the user. This capability is further enhanced with large on-chip EPROM for the ease of prototyping and field testing of new products.

HISTORICAL DSP SOLUTIONS

Over the past several decades, digital signal processing machines have gone through several evolutions incorporating these characteristics. Large mainframe computers were initially used to process signals in the digital domain. Typically, because of state-of-the-art limitations, this was done in non-realtime. As the state-of-the-art advanced, array processors were added to the processing task. Because of their flexibility and speed, array processors have become the accepted solution for the research laboratory, and have been extended to end-applications in many instances. However, integrated circuit technology has matured, allowing the design of faster microprocessors and microcomputers. As a result, many digital signal processing applications have migrated from the array processor, to microprocessors and microprocessor subsystems (i.e., bit-slice machines). This migration has brought the cost of the DSP solution down to a point that allows pervasive use of the technology. The recent introduction of single-chip digital signal processors with their increased performance and relatively low cost have further expanded digital signal processing from traditional telecommunication and military to consumer audio and automotive applications.

SINGLE-CHIP PROGRAMMABLE DIGITAL SIGNAL PROCESSOR

As noted previously, the underlying assumption regarding a digital signal processor is fast arithmetic operations and high throughput to handle mathematically intensive algorithms in realtime. In a typical single-chip digital signal processor (11-15), this is accomplished by using the following basic concepts:

-Harvard architecture,
-Extensive pipelining,
-A dedicated hardware multiplier,
-Special DSP instructions, and
-A fast instruction cycle.

Let's explain the benefit of incorporating these concepts in DSP architectural design:

Harvard Architecture

The Harvard architecture (16) is used for speed and flexibility, in which the on-chip program and data lie in two separate spaces and are carried in parallel by two separate buses. This permits a full overlap of instruction fetch and execution. In a typical general-purpose microprocessor, Von Neumann (16) architecture is used, where program and data are carried sequentially on the same bus. Instructions are therefore executed in serial.

Extensive Pipelining

In conjunction with the Harvard architecture, pipelining is used extensively to reduce the instruction cycle time to its absolute minimum, and to increase the throughput of the processor. In pipeline operation, the instruction prefetch, decode, and execute operations are handled in parallel, thus allowing the execution of instructions to overlap. As a result of this extensive pipelining, multiple DSP operations, such as multiply, add, shift, and data move (MACD), can be executed in one single machine cycle.

Dedicated Hardware Multiplier

As we saw in the general form of an FIR filter, multiplication is an important part of digital signal processing. For each filter tap (denoted by i), a multiplication and an addition (MAC) must take place. The faster a multiplication can be performed, the higher the performance of the digital signal processor. In general-purpose microprocessors, the multiplication instruction is constructed by a series of additions, therefore taking many instruction cycles. In comparison, the characteristic of every DSP device is a dedicated hardware multiplier. Important DSP operations, such as MPY and MAC, can be executed in single machine cycle as a result of the on-chip multiplier and extensive piplining. In a typical general-purpose microprocessor, these operations are typically executed in 30 to 40 machine cycles.

Special DSP Instructions

Special instructions resembling typical DSP operations are created to ease DSP algorithm development and speed up machine throughput. Examples of these DSP instructions are: MAC (multiply and accumulate), DMOV (data move, 1 DMOV represents a delay of 1 sample period), RPT (repeat, for repeating instructions), BLKD (block move of data) and BLKP (block move of program).

Fast Instruction Cycle

The realtime processing capability is further enhanced by the raw speed of the processor in executing instructions. The characteristics which we have discussed, combined with optimization of the integrated circuit design for speed, give DSP devices instruction cycle times approaching 50 nsec (nanosecond). This includes executing complicated DSP operations, such as MAC and MACD, within one single machine cycle.

Since the invention of the single-chip DSP in early 80's, many semiconductor vendors have introduced generations of digital signal processors into the market. One of the most popular family of digital signal processors, TMS320, now has three generations and over 15 members of devices available for the automotive designer to choose from (11-15). The early programmable digital signal processors were designed in NMOS. These devices now have been redesigned in CMOS to take advantage of lower power consumption and increased speed. Newer generations of DSP have also been added with further improvements in speed, throughput, and device density. As a point of reference in comparing DSP to general-purpose microprocessors, Table 2 lists the clock speed, throughput (in MIPS, million instructions per second), MAC execution, and device density for the Texas Instruments TMS320 DSP family and Intel 80386 (17-18), one of the most popular general-purpose microprocessors in the market today.

Table 2. TMS320 DSP vs. 80386 Microprocessor

	sample date	bit size	clock speed	throughput	MAC execution	device density
TMS32010 (1st generation)	1982	16 integer	20 MHz	5 MIPS	400 nsec	58,000 transistors
TMS320C25 (2nd generation)	1986	16 integer	40 MHz	10 MIPS	100 nsec	160,000 transistors
TMS320C30 (3rd generation)	1988	32 integer/ fltg. pt.	40 MHz	20 MIPS/ 40 MFLOPS	50 nsec	695,000 transistors
Intel 80386	1985	32 integer	16 MHz	4 MIPS	1,375 nsec	275,000 transistors

TRENDS OF SINGLE-CHIP PROGRAMMABLE DIGITAL SIGNAL PROCESSOR

The early versions of digital signal processors have made significant contributions to telecommunication and military applications (2-5). They also exihibit some deficiencies:

- Unfamiliar architecture to microprocessor designers
- Lack of friendly development support tools
- Device too costly for large volume applications

These early deficiencies have started being resolved by some of the DSP vendors. The following DSP improvements and technology trends have begun and will continue into the 1990's.

- Merging with general-purpose microprocessor/microcontroller features
- Lower cost, especially lower system cost
- Higher performance

Merging with General-purpose Features

Driven by the latest 1-u CMOS semiconductor processing technology and improvement in the DSP architecture (14-15), the latest digital signal processors are now featuring 32 bit architecture, fixed and floating-point operations, 50-nsec cycle time, large on-chip 4K x 32 ROM and 2K x 32 RAM, instruction cache, concurrent Direct Memory Access (DMA), and large 16M x 32 address area in one continuous memory space. These devices offer more, and expanded digital signal processing functions and run at much higher speed than their predecessors. The throughput of these devices has reached 20 MIPS or 40 megaflops (millions floating-point operations per second), previously unobtainable except with supercomputers. These architectural improvements coupled with more general-purpose instruction set, high level language support (like C), added third parties, and installed software base are making digital signal processors much easier to use.

Another aspect of the development is integrating more microprocessor and microcontroller type of peripherals on-chip for DSP spinoffs. These peripherals include better memory management, timers, serial ports, co-processor interface,.. features quite familiar to microprocessor designers. A few DSP vendors have also started offering processors with on-chip EPROM (12) for ease of prototype development, field testing, and early production runs.

Lower Cost

Early digital signal processor chips were sold for hundreds of dollars. Some

of these devices have been redesigned in CMOS with small geometry, which offers a reduction in size, cost and power consumption, and also an increase in throughput. DSP devices are now available in sub-five dollars range for high volume applications. Further system cost reduction is possible by integrating more peripherals on chip for semi-custom solutions. This enables DSP to be used in cost sensitive applications, such as compact discs (CD), intelligent toys, and computer disk drivers.

Figure 1 shows the DSP price in dollars per MIP ($/MIP) over a period of six years using the first-generation of the Texas Instruments TMS320 DSP (12) as an example:

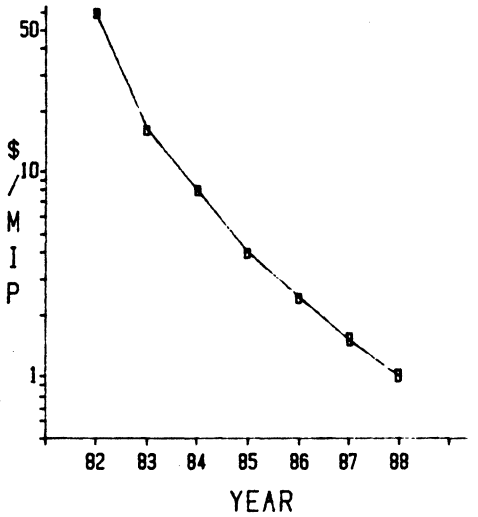


Figure 1. First-generation TMS320 $/MIP vs. Year

The cost cutting trend shown in Figure 1 will certainly be continued beyond the 1990's and will definitely benefit automotive designers in choosing DSP as solutions for their applications.

Higher Performance

Performance can be measured based on cycle time, algorithm benchmarks, applicatons throughput, or the combination of several or all of them.

One of the major DSP design emphases is to be able to compute MAC (multiply and accumulate) quickly. This capability has been improved substantially over the last few years, demonstrated in Figure 2 comparing the MAC (16-by-16 multiply and add) execution speed between the TMS320 devices and some of the popular general-purpose microcontrollers (GP uC) and microprocessors (GP uP) (17-22). The performance of DSP will continue to be increased and remain one of the primary driving forces for future DSP generations.

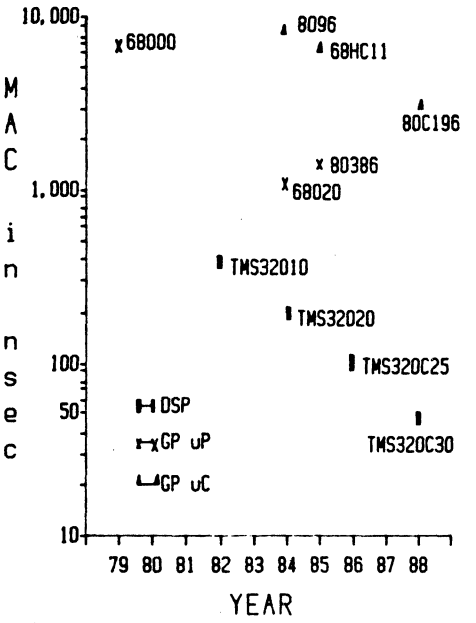


Figure 2. DSP and uC/uP MAC Execution vs. Year

DSP IN DIGITAL CONTROL SYSTEMS

Control systems have traditionally been implemented in analog form. With the availability of microprocessors and microcontrollers, digital control systems are taking over most applications from analog systems. Digital control systems have numerous advantages over analog systems. Most analog systems are limited to control using single-purpose characteristics of

the error signal like P (proportional), I (integral), or D (derivative), or a combination of these characteristics. Digital systems allow greater computational activity, thus making it possible to implement more sophisticated algorithms. Since digital systems are programmable, they can be used for control systems requiring online update of algorithms and process parameters to compensate for system changes. Digital systems are insensitive to component aging and temperature drifts. They also allow the same processor to be used to control multiple processes or implement multiple functions.

Digital control systems are used for applications in areas of robotics, numerical control, disk drive control, and engine control. These applications are increasingly requiring implementation of sophisticated control algorithms to meet their performance requirements. However, traditional 8/16-bit microprocessors and microcontrollers lack the speed of numerical calculations to meet some of these requirements (23-27). This has resulted in some compromise in control system design, such as using table lookup for algorithms and open loop controls.

The digital signal processors introduced previously have been specifically designed for these demanding applications, and offer a new tool for automotive design engineers. The computational speed available from a digital signal processor allows automotive designers to now execute control algorithms mathematically instead of using look-up tables. More sophisticated control algorithms, such as feedback control, multi-variable control, observer models and adaptive control can all be implemented with a single-chip digital signal processor (23-27).

The benefits offered with this new approach are

lower system cost:

- Because of the replacement of the large lookup table and a microcontroller with a single-chip DSP, system cost can be reduced.
- Furthermore, when Observer Model (a software model to estimate control system parameters) is implemented, some expensive sensors in the system can be eliminated and replaced by software estimation of these parameters using DSP.

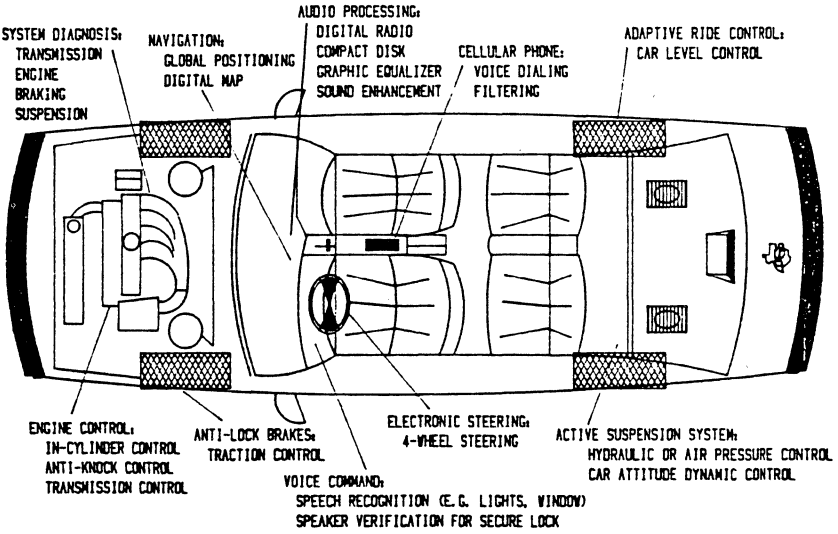


Figure 3. DSP Automotive Applications

and better system performance:

- More accurate control values can be obtained with mathematical calculation using a 16/32-bit DSP processor than those obtained from an 8/16-bit microcontroller. Interpolation between two values from the lookup table is not needed with DSP numerical calculation.
- State-variable control is now possible, while adaptive control will ensure the system performs optimally.
- Realtime diagnostics can be added to the DSP control software to check the proper performance of the controller, sensors and actuators.
- Much faster system response time can be achieved with the DSP. This means the vehicle can be designed to respond to the driver or driving environment more promptly, safely, and reliably.

DSP IN AUTOMOTIVE APPLICATIONS

Many automotive applications can benefit from using digital signal processors. This ranges, shown in Figure 3, from engine/transmission control, active suspension, adaptive ride control, antiskid braking, traction control to digital audio processing.

Powertrain Control Applications

The performance of the current electronic engine control can be improved by a closed-loop control system incorporating a DSP and sensors, such as in-cylinder pressure sensors reporting the precise operating status of each cylinder at every cycle (1,28).

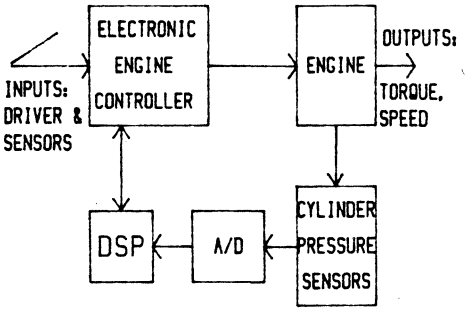


Figure 4. DSP Closed-loop Engine Control

The DSP in the system, Figure 4, performs engine pressure waveform analysis and determines the best spark timing, firing angles, and the optimal air/fuel ratios. The closed loop engine control scheme can tolerate external turbulances, aging, wearing,.. and maintains optimum engine performance and fuel efficiency (28).

Vehicle Control Applications

One of the most exciting new concepts in the vehicle control area is the active suspension system (29-31). Conventional vehicle suspension, utilizing dampers and springs, is insufficient to control the vehicle properly and incapable of responding to the rapidly changing forces inputted from the road conditions and car attitude changes. Improvement can be made with the programmed ride control suspension by introducing variable damping ratios into dampers. This type of system, typically utilizing an 8-bit microcontroller, has the deficiency of slow system response time and is unable to completely overcome external forces inputted to the vehicle.

The first active suspension system introduced by Lotus incorporates a TMS320 digital signal processor controlling four hydraulic actuators (29-31). This system is described in the following figure:

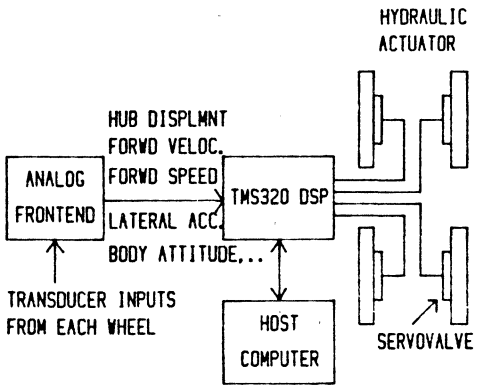


Figure 5. Lotus Active Suspension System

The TMS320 DSP takes into consideration body dynamics, such as pitch, heave and roll, and controls the 4 hydraulic actuators independently and dynamically to counter the external forces and car attitude changes. The TMS320 performs control algorithms with system parameters adaptively updated to achieve optimal comfort ride and road handling. Lotus race cars installed with the active suspension won 1987 Grands Prix in both Monaco and Detroit. The excitement has spurred much interest in introducing the active suspension design into commercially produced vehicles. As the cost of the entire system (both electronic and mechanical portions) is reduced, we can expect active suspension to appear in many vehicles in the 90's (1,31).

Another important vehicle control is the anti-skid braking (ABS) system. In the current ABS design, typically an 8-bit microcontroller is incorporated to read the wheel speed from sensors, calculate the skid, and control the pressure in the wheel brake cylinders. Traction control has been experimentally added to the ABS to control the vehicle in extreme conditions (wheel lock and spinning) in order to further increase vehicle stability, steerability and drivability. Added new features to the ABS, such as traction control and more diagnostics software, will demand more processing capability on the controller. Since 8-bit microcontrollers are running short of power, digital signal processors will become a prime candidate for the next generation ABS design.

It is interesting to note that the DSP based control system for the active suspension can be extended to control the skid and spinning of the wheels by simply adding anti-skid and traction control software. The DSP should have enough processing capability to perform all of these functions simultaneously.

Entertainment System Applications

The vehicle entertainment system has evolved from the traditional radios offering AM and AM/FM reception to the added features, such as cassette tape drives, graphics equalizers and power amplifiers. Recent advancements in CD (compact disc) and DAT (digital audio tape) have started impacting the entertainment system design. The audio systems produced for the 90's will be required to have better sound quality and higher bandwidth in order to reproduce the high fidelty in CD (with 44 kHz sample rate) and DAT (with 48 kHz sample rate). The entertainment system for the 90's will also evolve to become an information center. Communication between vehicles and stations, and between vehicles will be increased. This will demand information processing capability be built into the entertainment system. These requirements will demand DSP be integrated into the future entertainment/information center as shown in the following block diagram:

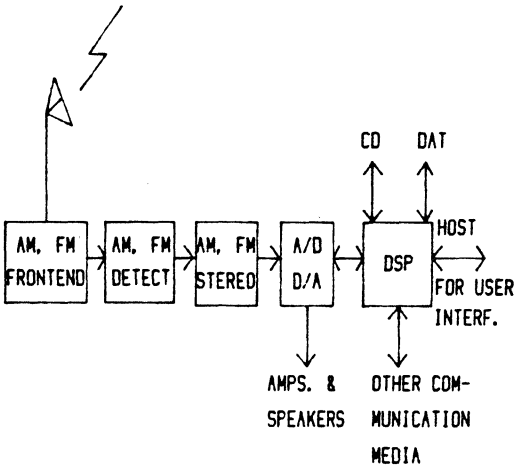


Figure 6. DSP Based Entertainment/ Information Center

Some of the functions performed by the DSP in this system are:

current existing features:

- graphics eqalization
- tone, base, and volume control
- noise reduction

added new features:

- automatic volume control
- acoustics noise cancellation
- radio personalization, such as station search, music/speech search,..
- RDS (radio data system)
- speech recognition for controlling non-critical functions
- speaker verification for anti-theft

The DSP based entertainment/information center will not only preserve high fidelity in CD and DAT, but also add friendliness, personalization, communication, and security to the user.

Other DSP Applications

There are many other automotive applications that could benefit from DSP. Some of these are: image processing for collision avoidance, voice dialing and noise filtering for cellular phones, Kalman filtering for Global Positioning Systems (GPS), realtime combustion analysis (32), exhaust noise and vibration cancellation (33), and acoustics noise suppression (34).

CHALLENGES TO THE AUTOMOTIVE INDUSTRY

While the DSP technology is becoming more mature, it is presenting tremendous challenges to the automotive industry. Control designers, who are used to a lookup table approach in designing a system, are now having to adjust to a new design practice. Elements in the control lookup table are usually generated by large computer simulation or simply trial-and-error. DSP, being an order of magnitude faster than 8/16-bit microcontrollers, implements control strategies in mathematical equations. This means control engineers must have the precise model or mathematical description for the system that he is controlling.

In the current control system design, designers are limited to simple control strategies, such as open-loop control, PID, and single or limited variable control. With the speed improvement offered by DSP, control designers can now implement more sophisticated algorithms: closed-loop control, state-variable control and adaptive control. To take full advantage of DSP, control designers must familiarize themselves with these digital control algorithms. The increased degree of freedom and potential benefit will demand more R&D effort to not only understand the engine model and vehicle dynamics, but also control them with better strategies to achieve the optimal vehicle performance.

For the entertainment system designer, the greatest challenge is not to replace the current matured analog solution, but instead to use DSP for added functionality. This additional functionality must have a perceived value by the user. Otherwise, the DSP product will not be successful. A good example of a misuse of the technology is to simply replace warning lights and buzzers with 'annoying' speech synthesis, and buttons and switches with 'inaccurate' speech recognition.

CHALLENGES TO DSP VENDORS

Some vendors are able to provide commercially available DSP in military specifications (12-13). The same, if not more stringent requirements, will also be needed to meet automotive applications for the harsh underhood and underbody environment. To be successful in automotive electronics, vendors need to be able to supply DSP devices 'with military specification quality, but at a consumer affordable price'.

Continual pressure from automotive designers will push DSP vendors for more support in documentation, software, and development tools. More general-purpose features will continue to be added to the future DSP devices. Semicustom DSP solutions will also be needed for high volume, cost sensitive automotive applications.

CONCLUSIONS

Digital signal processors offer an order of magnitude higher performance than general-purpose microcontrollers and microprocessors for time critical and numerical intensive tasks. Closed loop control and more sophisticated control and digital signal processing algorithms can now be implemented for automotive applications. There are some learning curves that both DSP vendors and the automotive industry have to go through in order to take full advantage of this new technology. The benefits will lead to more reliable, safer, higher performance, better handling and drivable vehicles.

REFERENCES

(1) J. G. Rivard, "Automotive Electronics in the Year 2000," SAE Paper # 861027, Proceedings of the Convergence 1986.

(2) L. R. Rabiner and B. Gold, Theory and Application of Digital Signal Processing, Prentice-Hall, 1975.

(3) A. V. Oppenheim, ed., Applications of Digital Signal Processing, Prentice-Hall, 1978.

(4) L. R. Rabiner and R. W. Schafer, Digital Processing of Speech Signals, Prentice-Hall, 1978.

(5) K. S. Lin, ed., Digital Signal Processing Applications with the TMS320 Family, Volume I, Prentice-Hall, 1987.

(6) A. V. Oppenhiem and R. W. Schafer, Digital Signal Processing, Prentice-Hall, 1975.

(7) T. Parks and C. Burrus, Digital Filter Design, Wiley, 1987.

(8) C. Burrus and T. Parks, DFT/FFT and Convolution Algorithms, Wiley, 1985.

(9) J. Treichler, C. Johnson, and M. Larimore, A Practical Guide to Adaptive Filter Design, Wiley, 1987.

(10) P. Papamichalis, Practical Approaches to Speech Coding, Prentice-Hall, 1987.

(11) K. S. Lin, et. al., "The TMS320 Family of Digital Signal Processors," Proceedings of the IEEE, Vol. 75, No. 9, September 1987.

(12) First-generation TMS320 User's Guide, Prentice-Hall, 1988.

(13) Second-generation TMS320 User's Guide, Prentice-Hall, 1988.

(14) TMS320C30 User's Guide, Texas Instruments, 1988.

(15) R. Simar, et. al., "A 40 MFLOPS Digital Signal Processor: The First Supercomputer on a Chip," Proc. of IEEE Int. Conf. on Acoustics, Speech, and Signal Processing, April 1987.

(16) H. Cragon, "The Elements of Single-Chip Microcomputer Architecture", Computer Magazine, Vol-13, No. 10, pp. 27-41, Oct 1980.

(17) H. D. Toombs and F. H. Phail, "Trends in VLSI and Its Affect on Powertrain Electronics," SAE Paper # 861034, Proceedings of the Convergence, 1986.

(18) Microprocessor and Peripheral Handbook, Volume I, Intel, 1987.

(19) Microcontroller Handbook, Intel, 1986.

(20) Embedded Controller Handbook, Intel, 1988.

(21) MC68000 Programmer's Reference Manual, Prentice-Hall, 1984.

(22) MC68020 User's Manual, Prentice-Hall, 1985.

(23) Y. Wang, M. Andrews, S. Butner and G. Beni, "Robot-Controller System," Symposium on Incremental Motion Control Systems and Devices, pp. 17-26, June, 1986.

(24) I. Ahmed and S. Lindquist, "Digital Signal Processors," Machine Design, September 10, 1987.

(25) I. Ahmed and S. Lindquist, "DSPs Tame Adaptive Control," Machine Design, November 26, 1987.

(26) H. Hanselmann, "Using Digital Signal Processor for Control," Proc. IEEE Industrial Electronic Conf., September, 1986.

(27) H. Hanselmann, "Implementation of Digital Controllers - A Survey," Automatica, Vol 23, 1987.

(28) C. Anastasia and G. Pestana, "A Cylinder Pressure Sensor for Closed Loop Engine Control," SAE Paper # 870288, SAE Conference Record, 1987.

(29) P. G. Wright and D. A. Williams, "The Application of Active Suspension to High Performance Road Vehicles," SAE Paper # C239/84, Proceedings of the Convergence 1986.

(30) D. A. Williams and S. Oxley, "Application of the Digital Signal Processor to an Automotive Control System," Proceedings of the Sixth International Conf' on Automotive Electronics, (U.K.), October 1987

(31) C. Csere, "Lotus Active Suspension," Car and Driver, June, 1988.

(32) E. Beck, K. Hahn, and M. Miller, "A Realtime Combustion Analysis Instrument," SAE Paper # 880689, Int'l Congress and Exposition, 1988.

(33) G. Chaplin, "The Cancellation of Repetitive Noise and Vibration by Active Methods," Application Note, Noise Cancellation Technologies Inc., Great Neck, N.Y.

(34) S. Cropley, "Breaking the Sound Barrier," Car Magazine (U.K.), January, 1988.

APPLICATION OF THE DIGITAL SIGNAL PROCESSOR TO AN AUTOMOTIVE CONTROL SYSTEM.

D.A. Williams.

Cranfield Institute of Technology.

S. Oxley.

Texas Instruments Ltd.

INTRODUCTION.

A conventional vehicle suspension, consisting of four dampers and four (or, possibly, six) springs has two serious deficiencies. Firstly, the number of elements is insufficient to control properly even the eight "ideal" vehicle degrees of freedom; secondly, the suspension reacts to all forces applied to it. Improvements in performance can be made by introducing non-linearities in both dampers and springs, and by adding passive rubber "isolators" at the suspension attachment points. However, none of these features is free from undesirable side effects. Essentially, a conventional passive suspension achieves, at best, a compromise solution to the problem of controlling body and wheel responses to external inputs. It is worth noting, perhaps, that this criticism also applies to most of the present generation of "active" suspension systems.

The Lotus Active Suspension system overcomes many of the deficiencies of a conventional suspension by replacing the springs, dampers, anti-roll bars, etc. by four irreversible hydraulic actuators. Measurements of a range of vehicle and actuator parameters are then used to control the position and velocity of each actuator in real time so as to synthesize an "ideal" vehicle suspension.

Control over the actuators is achieved by computing required actuator velocities, and feeding the appropriate command to an Electro-Hydraulic Servo Valve (EHSV) connected to each actuator. The necessary computations can be performed in one of several ways. The most versatile method is to implement the algorithms in a high speed digital processor. All the controllers used recently to implement the Lotus Active Suspension system have been based upon a family of Digital Signal Processors (DSP) designed and produced by Texas Instruments Inc.

This paper contains an outline history of the Lotus system, a discussion of the requirements for the controller, an introduction to the family of DSP's selected, and descriptions of two active suspension controllers which incorporate Digital Signal Processors. The paper concludes with a description of a possible production standard control system.

BACKGROUND.

The Lotus active suspension system was conceived in 1981 as one way of providing a ground effect racing car with a controlled ride without the suspension reacting to aerodynamic downforce, which at that time had a maximum value of around three times vehicle static weight. After their own elegant (mechanical) solution to the problem had been declared illegal, a prototype active suspension system was commissioned by Lotus. This was designed by the Flight Systems and Measurement Laboratories, CIT, and installed in a Turbo Esprit for evaluation. The prototype used commercial hydraulic components and a purpose built analogue computer to control the actuators. Hydraulic power was provided by an engine driven pump fitted with an hydro-mechanical pressure control system.

Aerodynamic downforce was, of course, small in the prototype installation. Therefore, in order to demonstrate the capability of the system to respond to loads selectively, the suspension was programmed to be insensitive to inertial forces which disturb a vehicle during manoeuvres.

It was clear from the start that the vehicle handled well when inertia components were removed from load measurements, allowing an average improvement in cornering speed of around 10 percent. It was also discovered that significant improvements in primary ride could be achieved without affecting handling. The obvious disadvantages of the system were increased complexity, weight and power consumption.

Information gathered from the prototype was used to specify a racing version of the system. This featured an optimized hydraulic system and a hybrid controller in which the gains of hard wired analogue control loops were set digitally by an eight bit processor. The processor had access to many of the measurements, and was programmed to modify loop gains adaptively, iterating at about 250 Hz. A single racing system was produced and tested extensively. It completed two Grands Prix before being removed, primarily for financial reasons. Although the car was not competitive in absolute terms, the handling improvements demonstrated in the prototype were confirmed in the racing system.

The publicity given to the active suspension race car resulted in contracts to convert several types of road vehicle to active suspension for research purposes. The requirement to be able to modify control laws stimulated the development of a fully digital controller. The controller developed for this application was based upon a TMS32010 Digital Signal Processor (DSP). It has been fitted to a total of seventeen cars during the past four years, and continues to give reliable service.

Certain installations have enhancements such as four wheel steer and four wheel drive. When such enhancements are fitted, these systems have a duplex version of the controller, which gives the potential for monitoring independently the performance of the various systems.

One further development has taken place during the last year. This is the development of a "lightweight" controller for the current generation of Lotus Formula One racing cars. The controller for this system is based upon the TMS32020 DSP. At the time of writing, five vehicles have been fitted with the system. All are operating successfully in environments where vibration levels exceed 20 gn RMS.

Development of the Lotus Active Suspension System continues with the objective of defining, under contract to a major manufacturer, a version of the system for installation in production vehicles.

CONTROLLER REQUIREMENTS.

A digital controller used in an active suspension system is required to sample analogue measurements in a chosen sequence and with a precise time interval between successive samples of a measurement. It must be capable of transforming the samples into actuator commands, and must, in general, be capable of converting those commands into analogue drive signals. The controller must also have discrete control over hydraulic fluid supply so that hydraulic energy can be dissipated directly by the controller in the event of a detected fault. The controller was also required to control the swashplate angle of the hydraulic pump in order to limit supply pressure, with over-riding limits on flow rate, and/or power consumption. The over-ride facilities were provided to enable the characteristics of different pumps to be emulated. Although not strictly part of the controller, transducer signal conditioning units were combined with the controller in order to minimize size and weight of the installation.

The sampling specification was dictated by the requirement to minimize variations in transport delay between one channel and another, and between the "frame" of samples and the corresponding actuator commands. The latter implied that interrupts should be avoided during control law calculations and, incidentally, control law code should be structured so as to minimize both transport delay and variations in transport delay.

Frequency bandwidth requirements for the system were unknown at the start of the project. However, tests carried out on conventional dampers revealed that the performance of such devices could be expected to deteriorate rapidly at frequencies above 20 Hz. Further tests carried out on a "single corner" test rig suggested that phase errors might be detectable if the cut-off frequencies (two pole) of certain transducers were to fall much below 100 Hz. It was concluded from the tests that an iteration interval of one millisecond would be required. With a 32 channel multiplexer, this implied an ADC sample rate in the order of 50 KHz.

The dynamic range required of the controller was dictated by a requirement to achieve control down to frequencies of the order 0.1 Hz. with a one millisecond iteration interval, together with the requirement for a load resolution of around 0.02 percent of full scale.

It was considered to be impractical to achieve both the required load resolution and the required full scale load, at least outside the laboratory. The solution to the difficulty was to scale load measurements so that half full scale corresponded to the ADC maximum. This arrangement gave a resolution close to the ideal. The reduced maximum observable load was not considered to be a serious restriction, since a load of this magnitude would normally cause maximum actuator velocity to be demanded.

It became clear from the above that a high speed 16 bit processor would be required for the application, having the capability of working to 32 bits when integration was required. The processor required the capability of accessing several peripheral channels, as well as handling at least two types of interrupt (Power up RESET and ADC End of Frame.) A secondary requirement was for a processor which could be integrated into a practical system with a minimum number of additional components.

THE DIGITAL SIGNAL PROCESSOR.

The processor chosen initially for evaluation was a TMS32010, manufactured by Texas Instruments. At the time hardware was available in sample quantities only. However, a proprietary Evaluation Board was purchased. The board allowed the basic performance of the processor to be assessed, and enabled engineers to become familiar with its internal architecture. The processor was not actually designed for real time process control applications, but its capabilities appeared to match the requirements for a suspension controller almost exactly. The internal architecture of the processor is shown in figure 1.

Evaluation of the processor showed that its performance, in a typical control application, was around 15 times that of an Intel 8086. However, certain factors could reduce effective performance dramatically. If the 144 internal registers were insufficient for the application, then additional workspace could be obtained by interchanging blocks between internal and external RAM. This turned out to be a lengthy operation, requiring 7 cycles (1.4 microseconds) per word moved. The second factor affecting performance would occur if the 4K word program space became insufficient for the control program. This would require program segments to be overlaid, and would make the processor unattractive for control applications.

The TMS32010 was incapable of "pausing" to accommodate true dual porting, or slow store. This feature presented difficulties to the system designer, but not otherwise. Overall, the processor has proved to be remarkably easy to use, and is almost completely immune to interference from external sources.

Many of the constraints imposed by the TMS32010 were removed with the introduction of the TMS32020 processor. The latter has 544 internal registers, sixteen read and sixteen write ports, can address 64 Kwords of programme and data store, and can be paused to accommodate various types of store. In addition, the processor incorporates a high speed serial port, an internal timer, additional interrupt vectoring, and has an enhanced instruction set. A diagram of the TMS32020 internal architecture is shown in figure 2.

The TMS32020 has the same clock rate as the TMS32010. However, the expanded internal store and instruction set means that, in the Active Suspension application, the control algorithm executes in about half the time required by the TMS32010. The sixteen bit address width and the ability to pause gives the system designer much improved flexibility in interfacing peripherals.

The TMS320 has been expanded into three distinct generations of DSP's (figure 3). The move to CMOS technology provides the designer with the advantages of low power consumption, wide temperature range and general suitability to the hostile automotive environment. Another trend which has increased the adaptability of the family is the addition of internal timers and serial interfaces. What started as a simple DSP with a small quantity of on-chip memory has become a truly versatile micro-controller.

CONTROLLER APPLICATION.

The logical arrangement of the Lotus TMS32010 controller is shown in figure 4. The control algorithm is stored, together with an initial set of parameters, in EPROM, which is connected to the processor via two ports. The EPROM board contains an address latch which is reset when Port 0 is accessed, and which is incremented after each word transfer. A ROM, mapped to the first 32 words of program space, contains a small program to transfer the control program and parameters from EPROM into external RAM whenever a RESET interrupt occurs. An R-C network is used to force a RESET whenever the controller is powered up.

The sampling requirement is achieved with an independent Analogue/Digital Converter (ADC) sub-system. The sub-system includes its own timing circuits and Dual Access Random Access Memory (DARAM). The sub-system samples each measurement in a preset sequence, storing each sample in RAM at the appropriate address. On completion of a frame, the sub-system interrupts the control processor and then idles for a preset time before restarting the sampling sequence exactly one millisecond after starting the previous one.

The Control Processor (CP) executes an "Idle" loop until interrupted by the ADC sub-system. After receipt of the interrupt, the CP transfers the last frame from DARAM to internal memory, calculates new actuator velocity commands and outputs these to the Digital/Analogue Converters (DAC). The CP continues to perform various "housekeeping" chores such as updating transducer and EHSV bias estimates before returning to the Idle loop to await the next interrupt. A timing diagram for the controller is shown in figure 5.

The five Digital/Analogue Converters are interfaced to the processor via two write ports. Port 5 is used to set the address latch to access the appropriate DAC; Port 6 is used to output the data to the DAC.

An optional Pulse Code Modulation (PCM) encoder is interfaced to output Port X. This allows the control program to output any accessible set of data to an external magnetic tape recorder. PCM bit rate is variable, but has been set to 36 Kbits per second for the present application. This has been used to sample 28 channels (30, including two sync words) at a rate of 100 per second.

The controller includes a communications routine which will accept Commands via an eight bit latch, data from one sixteen bit latch, and inputs data from another 16 bit latch. The latches are interfaced to an Intel 8031 eight bit processor, which controls an LCD display, and receives commands either from an RS232C serial link or from a simple keyboard. The 8031 is programmed to display any four TMS32010 registers, and to pass on commands either from the keyboard or from the serial link. A simple communications protocol is used to allow the operator to modify parameters, or even code, on-line.

For formal tests, files of parameters are stored in a portable general purpose computer. A small BASIC program executing in the portable computer is used to communicate with the controller via the RS232C link, and to transfer parameters between files and the controller as specified by the user.

The TMS32020 version of the Active Suspension controller differs from the earlier version in a number of respects. The internal timer is used to clock the ADC, and channel selection and conversions are initiated from an Interrupt Service Routine. Copies of suspension parameters are held in EEPROM, and are transferred during the RESET routine to internal RAM. Parameters are then copied in sets to a "work" page for execution. When a parameter is changed, both the EEPROM master and the internal copy are modified, so that the new value is preserved even if the controller is switched off. Each DAC is mapped directly to an output port. Communication with the controller is via a UART which is mapped into a Data address area. A diagram of the controller is shown in figure 6.

A novel feature of the controller is the adoption of solid state memory for recording data. This uses dynamic RAM controlled by a second TMS32020 which is programmed to refresh the RAM and to organize data storage. The two processors communicate via the high speed serial link. Data are transferred continuously until the driver selects "data off". Data are recovered, again via the high speed serial link, to a similar, off-board, dynamic RAM board, which is later interrogated by a Hewlett Packard general purpose computer. This fairly complicated arrangement developed to minimize the time required to transfer data from the vehicle. The entire 256 Ksamples are transferred from the vehicle in about 5 seconds and plots of the data, scaled into engineering units, are available within one minute.

The ADC arrangement used in the TMS32020 controller reduced the size of the system, but is now considered to be inferior to the arrangement used in the TMS32010 controller. The reasons are that time jitter can be introduced because certain instructions temporarily disable interrupts, and that full context switching is required within the service routine. Context switching in the TMS32020 is a relatively lengthy process.

Again, no serious difficulties have been experienced in using the TMS32020 processor in a (severe) automotive environment. The time required to move the design from inception to production prototype was around four months, and the production prototype first worked some two months before running in its first Grand Prix. By then three vehicles were fully operational, backed up by two complete sets of spares. Since that time a similar system has been fitted to another type of racing car.

FUTURE DEVELOPMENTS.

To date, only "research" Active Suspension systems have been designed. Work is proceeding to develop a "production" system to improve both performance and safety of passenger cars. One possible arrangement for such a system could be to integrate a controller into each strut, complete with transducers and signal conditioning. Communications between each strut controller would be via serial links attached to a fifth controller mounted centrally. A diagram of a possible arrangement is shown in figure 7.

Each strut controller would be programmed to manage its strut in isolation, and the central controller would be programmed to modify strut parameters (when required) and to modify strut responses as necessary; the central controller would, for example, simulate the action of roll bars. The failure mode, if serial communications were lost, would thus be relatively "soft." The central controller would also service any driver controls, displays, etc. The five controllers could be identical, each comprising a processor, memory, an eight channel ADC, two DAC's, five 100 KHz. serial links, and a discrete output latch. In a production system, signal conditioning would be integrated into each transducer.

Quite clearly, the controller would have to be very small and rugged (ideally on a single chip), and environmental sealing would have to be effective. Computing power available, if TMS320 series DSP's were used, would be considerable, and would admit the possibility of integrating other vehicle functions, such as anti-lock braking, engine management, four wheel steer, torque control, etc. into a single, distributed system.

Technologies under development at Texas Instruments could make the proposed arrangement feasible within the next five years. These include Application Specific Integrated Circuits (ASIC) which integrate precise analogue functions with complex digital circuits, and Application Oriented Controllers (AOC). The latter is a new family of standard modules which can be combined in a single chip to produce custom processors quickly and efficiently.

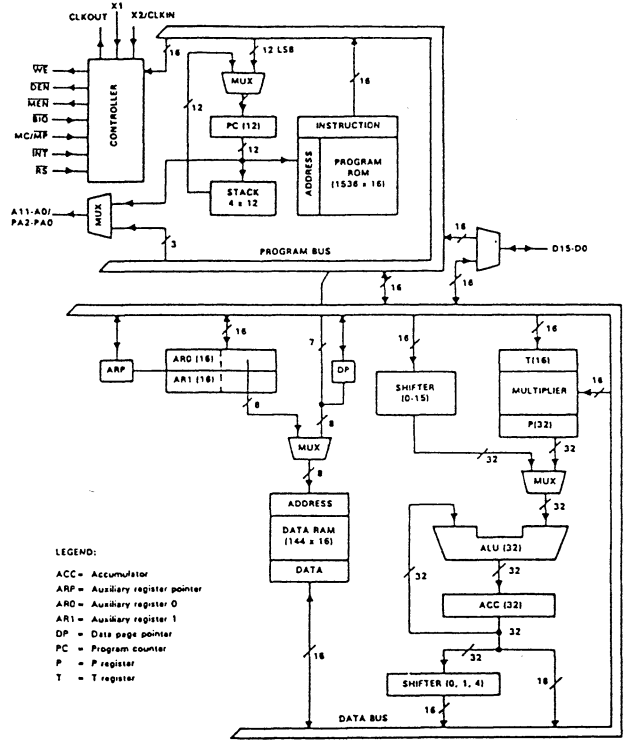


Figure 1 Functional Block Diagram of the TMS32010

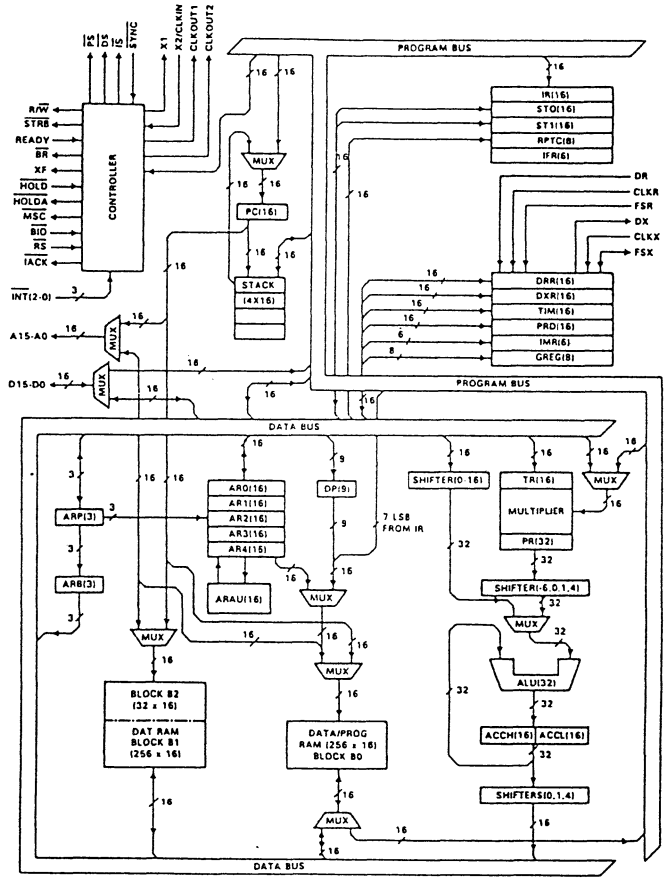


Figure 2 Functional Block Diagram of the TMS32020

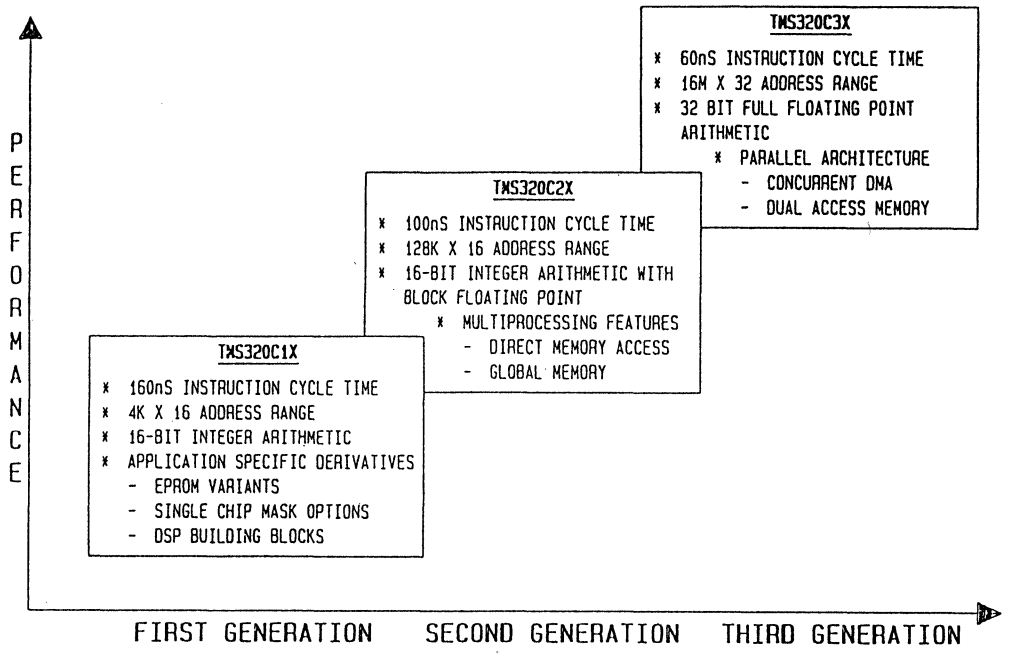


Figure 3. TMS Family

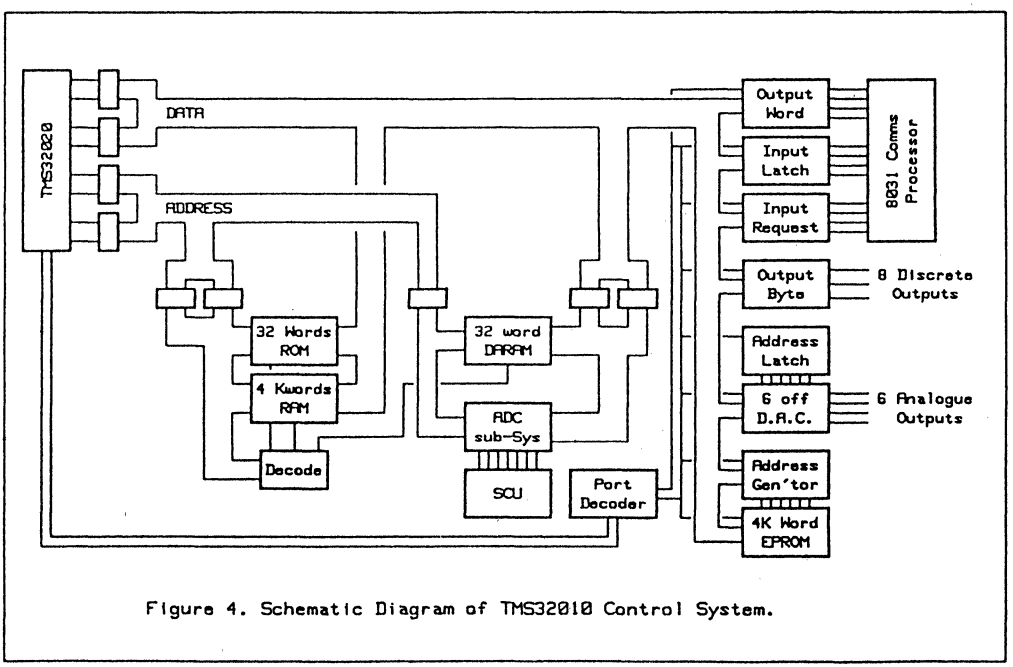


Figure 4. Schematic Diagram of TMS32010 Control System.

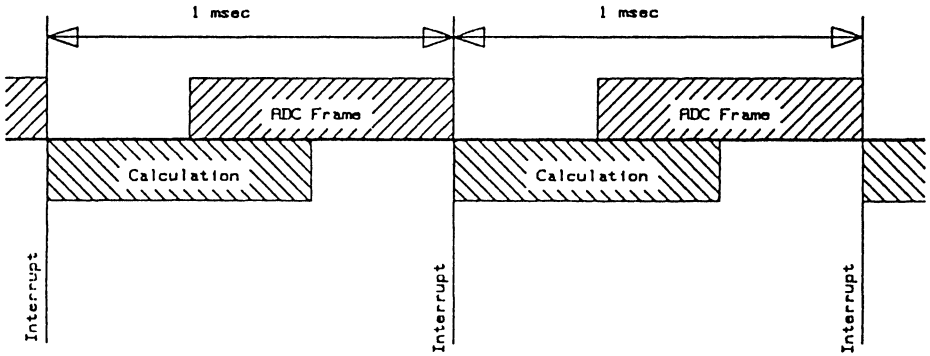


Figure 5. Controller Timing Diagram.

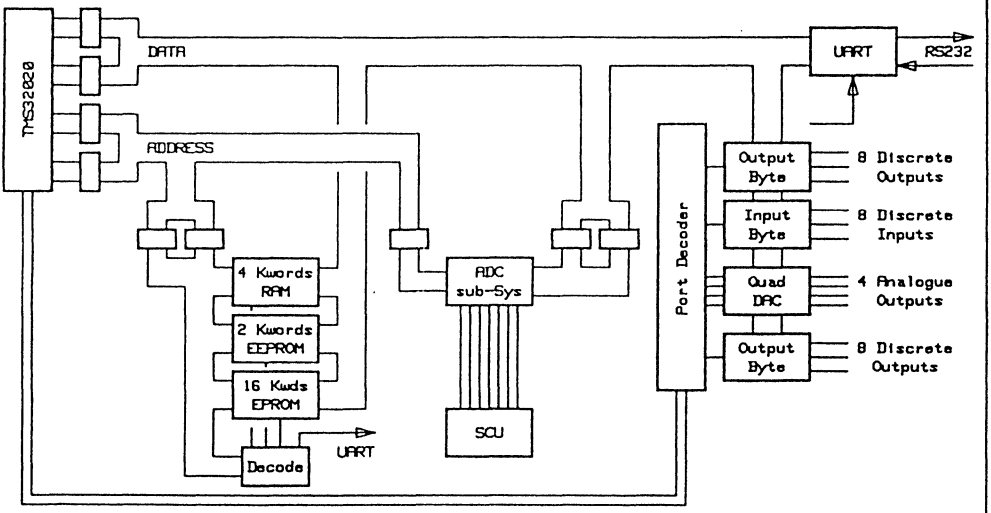


Figure 6. Schematic Diagram of TMS32020 Controller.

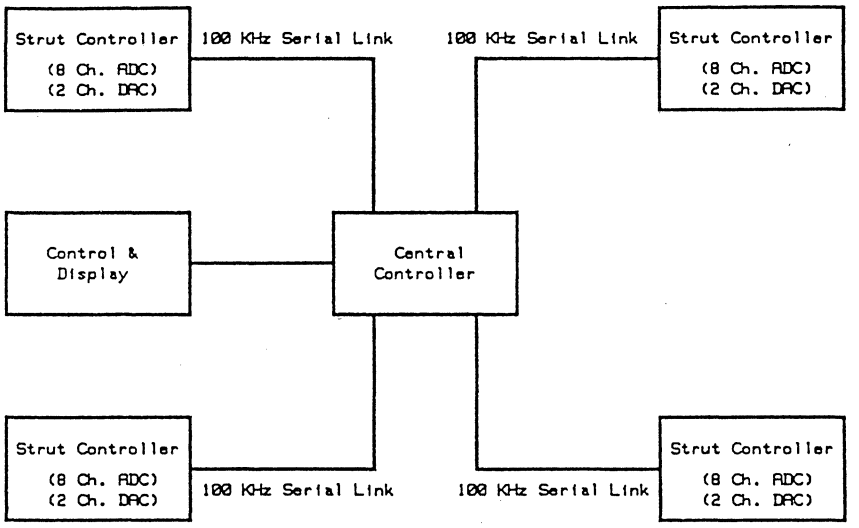


Figure 7. Schematic Diagram for a Distributed Controller.

Dual-Processor Controller with Vehicle Suspension Applications

KAMAL N. MAJEED, MEMBER, IEEE

Abstract—**A dual-processor controller suitable for computation-intensive control and signal-processing algorithms is described in this work. The controller is architectured around a general-purpose microcontroller and a digital signal processor (DSP). The main goal of the design is efficient computation of mathematically oriented algorithms with the ability to interface with sensors and actuators. The effectiveness of controllers incorporating DSP chips is demonstrated with two applications of suspension-sensor data frequency-domain analysis and state estimation of a "quarter-car" suspension test rig.**

I. INTRODUCTION

LOW-COST general-purpose microcontrollers became widely used in the 1970's, in applications ranging from vending machines to engine control. In the 1980's, a new generation of microprocessors evolved: the general-purpose digital signal processors (DSP). These processors were distinguished for their impressive speeds in numerical computations such as multiplication and product accumulation. Also, the instruction set is designed for complex algorithms requiring intensive numerical calculations. Initial use of these micro's has been in the area of signal processing such as digital filters and fast Fourier transforms. However, in recent years an increasing number of advanced control algorithms has been implemented using digital signal processors. In such applications, the update loop time must be small enough for proper implementation (one millisecond is not uncommon). To achieve that goal, the dual-processor controller discussed in this work was designed to optimally utilize each microprocessor in its area of strength.

The paper is organized as follows. Section II contains the controller architecture and design. Section III covers a fast Fourier transform applied to road characterization. In Section IV, a state estimator of a "quarter-car" test rig is described.

II. CONTROLLER ARCHITECTURE AND DESIGN

The electronic controller is architectured around the digital signal processor (Texas Instruments TMS320C15) and microcontroller (Motorola 68HC11) (Fig. 1). The 68HC11 is used most effectively in input/output (I/O) operations, data moving, and logical microcontroller functions. This results in relieving the digital signal processor from any I/O overhead to optimally utilize it in the algorithmic and numerical computations. The interprocessor communication is done through a dual-port random-access memory (DP-RAM).

Manuscript received January 15, 1990; revised April 13, 1990.

The author is with the Delco Products Division, General Motors Corporation, P.O. Box 1042, Dayton, OH 45401.

IEEE Log Number 9036995.

Referring to Fig. 1, the sensor signals are conditioned with anti-aliasing filters, then multiplexed through a 24-channel multiplexer to a sample-and-hold (S/H) and a 12-bit analog-to-digital (A/D) converter. The analog signals are converted by the 68HC11 and then stored in the dual-port RAM. The TMS 320 reads this set of raw sensor data and processes the algorithm computations. The resulting control signals are then stored in the dual-port RAM. The 68HC11 processor outputs these signals to the system drivers. The 68HC11 can output any data memory as digital on/off pulse-width-modulated (PWM) or as an analog signal through the digital-to-analog (D/A) converter. The 68HC11 has a serial communication port which is used to communicate with a portable personal computer (PC) for system monitoring.

The analog signal conversion to digital form is shown in Fig. 2. There are three degrees of pipelining (channels $K+1$, K, and $K-1$ are processed concurrently). This results in a maximum of 9-μs conversion time from analog signal at the sensor to digital data in the DP-RAM. In Fig. 3, an opposite operation is done for the analog outputs. Eight output channels share the same D/A through the use of eight S/H integrated circuits. Here again a maximum of 9 μs is achieved for total conversion time.

The dual-port RAM data can be read from either port simultaneously. However, a write access to the same address by both processors at the same time can have unpredictable results. For this reason, the software program must ensure a no-conflict access of the DP-RAM.

Fig. 4 shows the flowchart of a typical application program for the dual-processor controller. The 68HC11 starts by converting all the analog sensor signals, then sends a synchronization signal to the DSP processor. The TMS320 reads the converted sensor signals from the dual-port RAM and processes them through the application algorithm. If a control signal is generated, the DSP stores it in the DP-RAM and branches to the beginning of the loop. The 68HC11 reads the control signal and sends it to the appropriate drivers. After processing the display and communication routines, the microcontroller waits to complete the specified loop time and restarts another cycle.

III. FFT ROAD CHARACTERIZATION

With the emergence of automatically variable damping vehicles, there is a need for a damping adjustment mechanism. The use of the frequency domain [1] has the desirable property of distinguishing between the body and wheel-hop frequencies (about 1 Hz and 10 Hz, respectively). The midrange frequen-

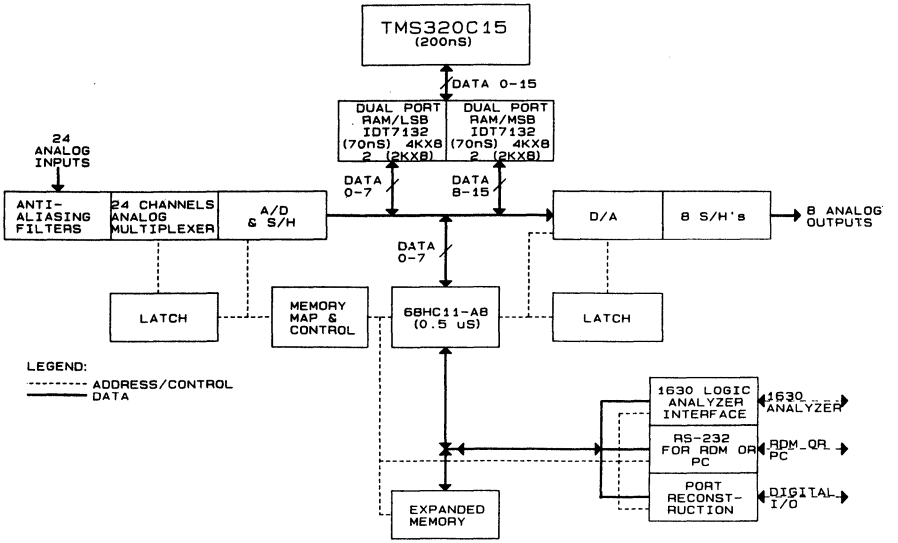


Fig. 1. Controller block diagram.

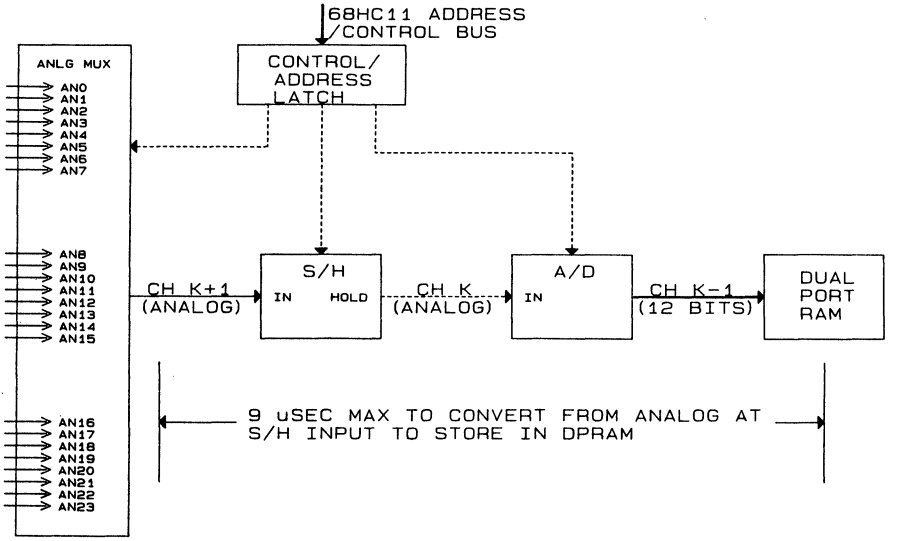


Fig. 2. Analog-to-digital conversion.

cies (about 5 Hz), where the human sensitivity to vibrations is high, can also be used to influence the damping adjustment.

A fast Fourier transform (FFT) of a vehicle body-to-wheel displacement was implemented in real time [1]. This was possible mainly due to the computing power of the TMS320 DSP chip. Fig. 5 shows the results of different test roads. It is noticed that the "chatterbumps" test road excites both body and wheel resonant frequencies. Thus the two peak at about 1 Hz and 10 Hz (corresponding to the body and wheel resonance frequencies, respectively). On the other hand, the "waves" test road (at certain car speeds) excites mainly the body mode. The "smooth concrete" test road has a flat spectrum, which is obviously the result of the small displacements in the suspension.

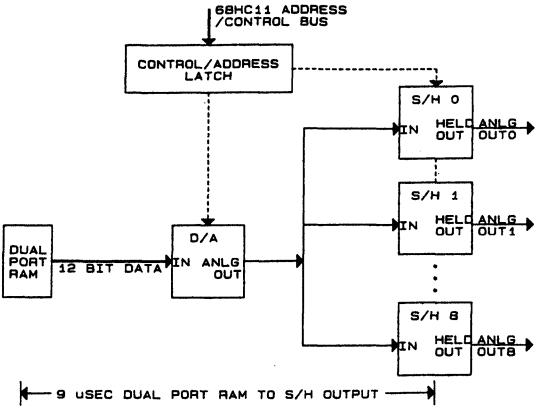


Fig. 3. Digital-to-analog conversion.

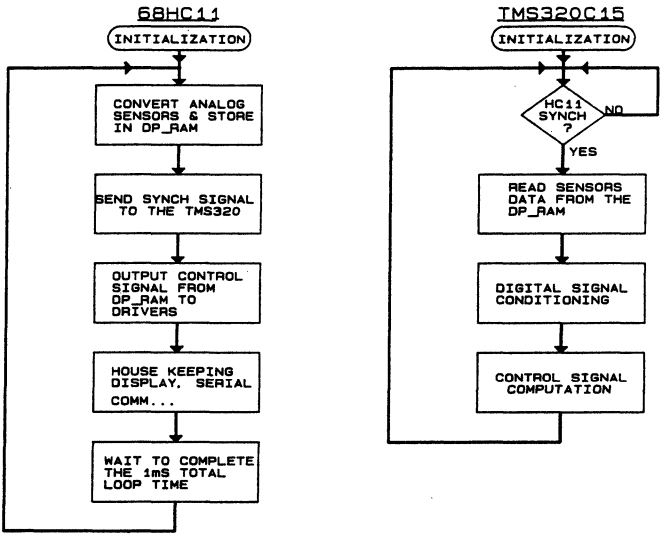


Fig. 4. Dual-processor flowchart.

Proper application of the FFT to produce an accurate spectrum of a signal depends on a good parameter selection [2]. This selection is governed by the relationships outlined next.

- T Sampling time.
- f_s Sampling frequency $= 1/T$.
- F Frequency resolution of the FFT.
- t_p Record length (effective signal period).
- f_h Highest frequency in spectrum.
- N Number of samples in record.

To avoid aliasing and the distortion of the FFT spectrum, it is necessary that

$$f_s > 2f_h \tag{1}$$

or

$$T < 1/2f_h. \tag{2}$$

The increment between the output FFT spectral components F is determined by the record length

$$F = 1/t_p. \tag{3}$$

Thus a long enough t_p must be selected for the specified frequency resolution. The number of time samples N (same as FFT frequency components number) is

$$N = f_s/F = t_p/T. \tag{4}$$

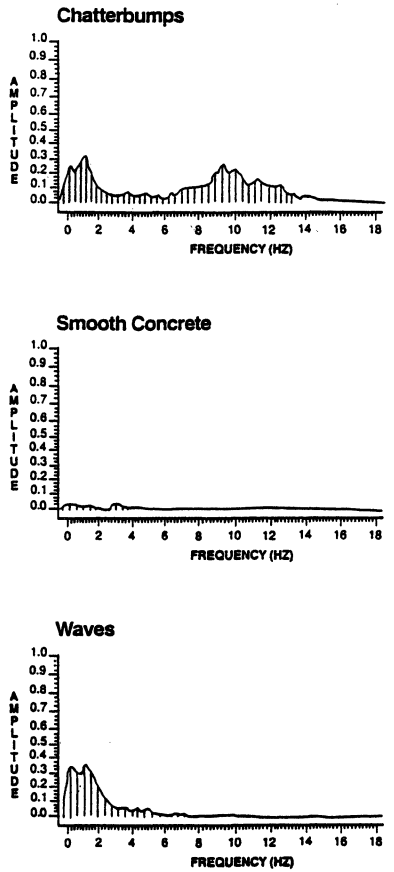


Fig. 5. Road-test FFT.

Equations (2)–(4) yield

$$N > 2f_h/F. \tag{5}$$

Thus for a given highest "appreciable" signal frequency f_h and a specified frequency resolution F, (5) determines the required number of time samples N.

The results of Fig. 5 are based on an $N = 128$ point FFT. The 256- and 512-point FFT's were found to produce only marginal road signature improvements when compared to the 128-point FFT. A sampling frequency f_s of 38 Hz was adequate to process the signal frequencies in the range (0–10 Hz). This sampling frequency must be increased if the signal level is appreciable at frequencies beyond 19 Hz (as mandated by (1)). For this reason anti-aliasing low-pass filters were used to prevent the distortion of the FFT spectrum by attenuating the spectrum beyond the wheel-hop frequency of 10 Hz. To protect against the aliasing problem, (4) shows that one can increase the sampling frequency at the expense of lower resolution of the FFT components (for the same number of points N) in the frequency range of interest.

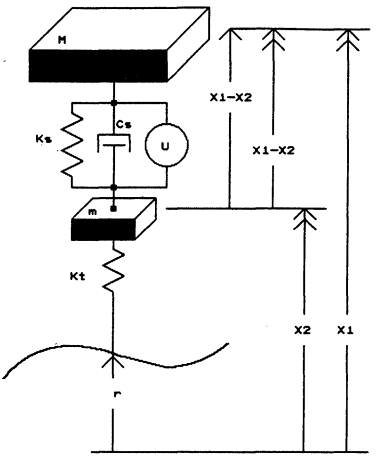


Fig. 6. Quarter-car test rig.

The processing of the 128-point FFT (using the TMS 320C15 processor) takes about 3 ms while it takes $t_p = 3.35$ s (at 38-Hz sampling frequency) to fill the 128-point time data buffer. The frequency resolution of the resulting FFT spectrum F is about 0.3 Hz.

IV. Quarter-Car Test Rig State Estimation

Recently there has been increasing interest in electronically controlled suspensions. One such suspension is the active suspension where a force generator, usually a hydraulic actuator, is commanded by a controller to achieve desired suspension characteristics. A set of sensors is read and the appropriate control is determined by an algorithm. Some control algorithms use full state feedback control where the whole state of the system is fed back. This approach is used in the works of Thompson [5] and Chalassani and Alexandridis [6], where the active suspension control law is derived using the linear quadratic Gaussian (LQG) optimal control theory. Another practical approach is to use limited state feedback to control the active suspension system, as in the work of Majeed [7]. In either case, some of the states of the system can be measured while other states are estimated.

An experiment was performed using the dual-processor of Section II and the quarter-car test rig of Fig. 6 . The rig consisted of a sprung mass ($M = 500$ kg), spring ($Ks = 18$ kN/m), passive damper ($Cd = 1$ kN·s/m), and the unsprung wheel ($m = 70$ kg) with a tire spring ($Kt = 250$ kN/m)). The resonance frequencies of the sprung and unsprung masses are about 1 Hz and 10 Hz, respectively. Simulated road inputs were available and all system states were directly measured for evaluation of the estimator results. The state estimator was fed the measurements of sprung and unsprung mass inertial velocities $X1$ and $X2$ (integrated accelerations), body-to-wheel relative velocity ($X1$-$X2$) and displacement ($X1$-$X2$).

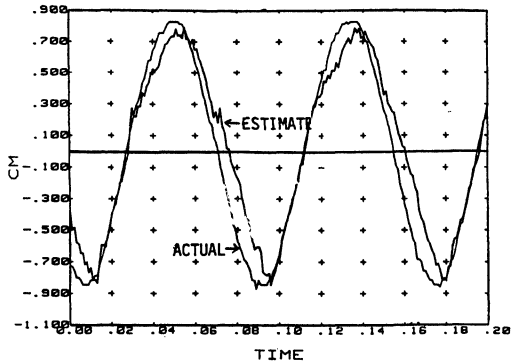


Fig. 7. State estimator—sprung mass displacement.

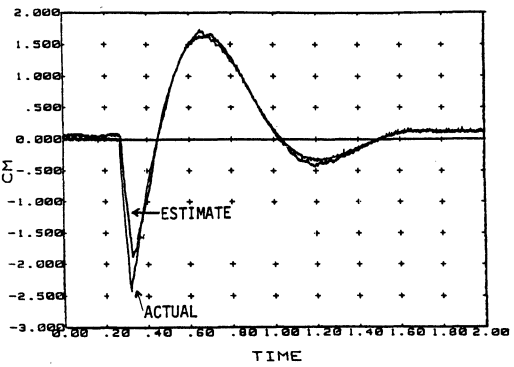


Fig. 8. State estimator—wheel displacement.

The two main estimated states of interest were the body (or sprung mass) displacement $X1$ and the tire deflection ($X2$-r).

The full state estimator test results are shown in Figs. 7 and 8. Fig. 7 shows the estimated sprung mass displacement versus the actual (measured directly to the floor) for a step road input. Fig. 8 shows the wheel displacement for a 10-Hz sine-wave road input.

The quarter-car system can be represented in a state space form as follows:

$$\dot{X} = AX + BU \tag{6}$$

$$Y = CX \tag{7}$$

where

- $\dot{X}$ system states rate vector (4 × 1),
- X system states vector (4 × 1),
- A system matrix (4 × 4),
- U actuators force vector (1 × 1),
- B control matrix (4 × 1),
- Y system outputs vector (4 × 1),
- C output matrix (4 × 4).

The control vector U for the full-state feedback [5], [6] is computed from

$$U = K_c X \tag{8}$$

while for limited-state or output feedback [7]

$$U = K_c Y \tag{9}$$

where K_c is the optimal gain matrix.

To estimate the states of the system, a Luenberger observer or a discrete Kalman filter is used [3].

The steady-state gain discrete Kalman filter is given by

$$X_{n+1} = \phi(T)X_n + K_f(Y_n - CX_n)\,, \tag{10}$$

where

- X_{n+1} estimated state vector at time $n+1$,
- T sampling time,
- K_f steady-state Kalman filter gain,
- $\phi(T)$ system transition matrix,
- Y_n measurement vector at time n.

The Kalman filter gain K_f is computed using a control software package such as Matlab [11]. The Kalman gain computation is based on the solution P of the Riccati equation:

$$A^T P - PBR^{-1}B^T P + PA + Q = 0. \tag{11}$$

The Riccati equation is solved with Q and R representing the plant and sensor Gaussian noise processes of zero mean. The Q and R values represent the compromise between the plant noise and uncertainty and the sensor noise, respectively.

In real-time implementations of state space equations, such as (8)–(10), it is desirable to create vector and matrix-times-vector macros. This greatly simplifies the programming task and improves appreciably the execution speed compared to the subroutine approach. The penalty of using macros is a larger program memory requirement. Equation (10) is computed on-line iteratively every loop time. One millisecond was used for the quarter-car test.

V. Conclusion

A dual-processor controller incorporating a microcontroller and a digital signal processor was successfully developed for implementation of computation-intensive algorithms. The goal of very short loop time was achieved by optimum use of the two processors' resources. The effectiveness of such controller was demonstrated by a real-time FFT application to road characterization and by state estimation of a quarter-car test rig. Also, the capability of such controllers to handle modern control theory requirements was shown by the quarter-car test-rig state estimation.

Acknowledgment

The author is grateful to Nick Kapsokavathis, Reed Hanson, Richard Longhouse, and Donald Graham for their support during the period of this work.

References

[1] K. N. Majeed and D. E. Graham, "Controlled vehicle suspension using FFT," Research Disclosure Pub., no. 28659, Feb. 1988.

[2] W. D. Stanley, *Digital Signal Processing*. Reston, VA: Reston Publishing, 1984.

[3] R. G. Brown, *Introduction to Random Signal Analysis and Kalman Filtering*. New York: Wiley, 1983.

[4] J. J. D'Azzo and C. H. Houpis, *Linear Control System Analysis and Design*. New York: McGraw-Hill, 1981.

[5] A. G. Thompson, "An active suspension with optimal linear state feedback," *Vehicle Syst. Dynam.*, vol. 5, pp. 187–203, 1976.

[6] R. M. Chalasani and A. A. Alexandridis, "Ride performance potential of active suspension systems, Part 2," ASME Monograph, AMD—vol. 80, DSCDSC—vol. 2, 1986.

[7] K. N. Majeed, "Centralized/local optimal output feedback control and robustness with application to vehicle active suspension," Ph.D. dissertation, University of Dayton, Dayton, OH, Dec. 1989.

[8] *First Generation TMS320 User's Guide*, Texas Instruments, 1987.

[9] *MC68HC11A8 HCMOS Single Chip Microcomputer Manual*, Motorola, 1985.

[10] K. N. Majeed, "Dual-processor automotive controller," presented at IEEE Workshop on Automotive Applications of Electronics, Dearborn, MI, Oct. 1988.

[11] Matlab, *Control System User's Guide—Version 2.2*. The Math Works Inc., South Natick, MA, Aug. 1986.

Kamal N. Majeed (M'86) received the B.S. degree in electrical engineering from the University of Basrah, Iraq, in 1976, the M.S. degree in electrical engineering from The Ohio State University, Columbus, in 1982, and the Ph.D. degree in electrical engineering from the University of Dayton, Dayton, OH, in 1989.

From 1977 to 1980 he was a Lecturer in the department of electrical engineering of Basrah Technical Institute, Iraq. From 1980 to 1981 he worked with Schlumberger S.A. as a Well Logging Engineer, International Staff. He is currently a Development Engineer with Delco Products Division of General Motors Corporation, Dayton, OH, where he has been working since 1983. He is currently an Adjunct Professor in the Department of Electrical Engineering, University of Dayton. He has designed and implemented a variety of control algorithms and electronic controllers for advanced vehicle suspension systems and other automotive control applications. His interests are in microprocessor-based digital control, adaptive control, optimal control, and digital signal processing applications.

An advanced racing ignition system

T MEARS, BSc, AMIEE
Lucas Automotive, Birmingham
S J OXLEY, BSc
Texas Instruments, Bedford

SYNOPSIS This paper describes the rationale and development of a high performance racing ignition controller based on a Digital Signal Processor. Applying new techniques such as these in the high pressure racing environment allows companies such as Lucas to develop strategies for production engine management systems in the 1990's. The vastly increased processing power available allows designers to begin to consider control techniques previously considered impractical for low cost production systems.

1 INTRODUCTION

The current generation of high performance racing engines have been developed to such a degree that a 15000 r/min V12 engine is now a reality. Such an engine, by its very nature, requires full electronic control of both fuelling and ignition in order to extract the maximum performance. Traditional electronic engine management systems (EMS) are unable to provide accurate control for such an engine - the main barrier being processing speed. Lucas have applied a single chip Digital Signal Processor (DSP) from the Texas Instruments TMS320 family to achieve distributorless mapped ignition for high performance racing engines. In future, mapped sequential fuelling will be added with the DSP controlling a slave processor. The alternative to using a DSP was to implement the system as a multiprocessor configuration which is both inelegant and difficult to develop and maintain as a reliable system.

2 SYSTEM REQUIREMENTS

The system being described is required to be able to control a Capacitor Discharge Ignition (CDI) system on a variety of engines up to a V12, 15000 r/min Formula 1 version. Additionally, the system must be able to be tailored to a variety of engine geometries and firing orders.

The dominant factor for such engines is the operating speed of the system - the V12 engine referred to above, with a 40 degree V-angle and 10 degree timing markers, requires processing of degree markers that are a mere 111uS apart at full speed. It is obvious that conventional microcomputers with minimum instruction cycle times of 2 to 4uS (complex instructions such as multiply may take 10 times this period to execute) could not be used to implement a single processor system.

The speed requirement is the reason for using CDI on racing engines - conventional inductive coil based systems would be unable to build up sufficient energy in the time between sparks at full engine speed.

In order to obtain the maximum performance from an engine, the following time critical operations must be performed accurately in real-time:-

1 Record the period between adjacent teeth on a timing wheel mounted on the engine - typically at 10 degree. intervals.

2 Recognise and maintain synchronisation with a missing tooth on the timing wheel and an independent TDC marker.

3 Trigger the CD circuit at an angle defined by a three-dimensional map (16 by 64 points - throttle angle against speed). Full interpolation is provided between the discrete points on the map with modifying functions applied for temperature, boost, pressure, etc.

Points 1 and 2 require precise measurement of the tooth intervals without latencies caused by interrupt actions which can give an uncertainty at least as long as the longest instruction. The output function, point 3, requires rapid mathematical processing to allow the ignition timing to be based on the most up-to-date information as possible. It then requires an output to be driven at a precise time after a specified tooth number.

In the short term a separate fuelling controller is being used, with the

Reprinted by permission of the Council of the Institution of Mechanical Engineers from *IMechE*, 1989.
On behalf of the Institution of Mechanical Engineers.

ignition controller passing speed and synchronisation information to allow mapped sequential injection to be achieved.

In order to meet these requirements without using a processor with the speed of a DSP, a multiprocessor system would be mandatory. Multiprocessing creates many additional problems in terms of synchronisation, data sharing and overall maintainability. There are systems available with up to 10 processors in one controller - a nightmare to develop and use in the high pressure racing world.

3 ATTRIBUTES OF THE DSP BASED MICROCONTROLLER

The device at the heart of the ignition system is the TMS320E14 from Texas Instruments. This device takes the first generation CMOS DSP core from the industry standard TMS320 family and adds the functions found in more complex microcontrollers.

Firstly let us define a DSP (1) - it is generally accepted that such a device must be a single chip with on-chip memory (RAM/ROM) and a single cycle hardware multiplier. In its original form the DSP was intended for real-time digital processing of analogue signals. In essence it was designed to perform filter functions which can be treated discretely as a sum of products. The same mathematical functions are required for many digital control systems used today - see Fig 1, PID implementation. What at first may appear rather odd instructions, are in fact functions that normally take several instructions to implement in conventional microcontrollers, i.e. LTD.

LTD - loads Register T with data from memory

- adds Register P contents into the Accumulator

- data in memory is copied to next higher address

This type of instruction is very useful for map interpolations to derive values between map sites.

The fact that all instructions execute in a single cycle means that with a 25MHz crystal each instruction takes 160nS.

However, the reason that the DSP has not been used in automotive systems to any great extent is due to its previous requirement for several support chips to handle I/O and timing functions. In the TMS320E14 an event manager has been added that provides input capture and output compare facilities in hardware - this ensures that critical time related functions occur independently of the CPU, thus avoiding the associated latencies. Additionally there are 16 I/O lines which may be manipulated independently under software control. An on-chip serial port and Watchdog timer complete the additions that have turned the DSP into a microcontroller - see Fig. 2, TMS320E14 block diagram.

4 SYSTEM IMPLEMENTATION

The TMS320E14 is an EPROM device with 4K words of EPROM and 256 words of RAM. Whilst the controller could easily be implemented without additional memory, the capability to address a further 4K words of off-chip memory has been exploited. The off-chip memory comprises 2K words of EPROM, used for map storage and 2K words of non-volatile RAM for diagnostic and telemetry functions.

Outputs used to drive the CD circuits are driven from 4 of the 6 output compare registers - the system being able to multiplex in software these 4 signals on to the 12 outputs required.

A block diagram of the system is given in Fig 3 which serves to highlight the integration of I/O functions on to the DSP chip, thus minimising the requirement for support circuitry.

The programme is a conventional real time control implementation in which time critical responses are performed in the foreground (interrupt driven) routine, and non-time critical calculations and management tasks are performed in the background routine.

The primary foreground task is an interrupt routine triggered by the signal from a 36 tooth wheel with ten degree tooth spacing. The flywheel has one missing tooth situated at T.D.C. on the reference cylinder. Since on a V8 engine, there are two crankshaft revolutions for a complete firing of each cylinder, it is not enough to simply detect the missing tooth to synchronise the engine. A second signal derived from a half engine speed sensor situated on the camshaft is used to indicate the cycle.

The software is designed to operate in the range of 51 to 16000 r/min on engine configurations up to and including V12. At high speeds the frequency of interrupts from the crankshaft sensor is given by:-

Engine speed = 16000 r/min

= (16000 X 6) degrees/s

Time for 10 degrees = 104uS

Hence frequency = 9600Hz

. At this speed, it is the phenomenal processing power of the TMS320E14 that enables control to be achieved. Running at 16MHz, single cycle execution is 250nS, enabling 416 instructions to be executed in the tooth period. This enables both the interrupt task and a significant proportion of the background task to be completed in one tooth period. For example, on a V8 engine, there are approximately 9 tooth periods between sparks, the processor is easily capable of cylinder by cylinder update of the advance angle.

The background task uses engine speed and throttle angle to address the main ignition map. This map has 16 load (throttle position) and 64 speed sites making a 1K map. Only 8 bits are needed, but since the DSP is word oriented, each memory location contains two contiguous map values. This means that the 16 by 64 site main ignition map actually uses 512 words of memory. The hardware uses an external 8 bit A/D converter to measure the throttle angle. This raw value is filtered using the equation:

$$\text{Filtered position} = \frac{(3 \times \text{previous filtered position}) + \text{measured position}}{4}$$

The hardware multiplier plus very simple divide mechanism enables extremely fast and reliable implementations of the above type of algorithm. Since the microprocessor does not have a right shift instruction, the author tends, where possible, to avoid using left shifts to do division because the load accumulator with shift instruction is sign extended. This, where the dividend has a one in bit 15, requires masking of the extended bits. It is far simpler to use the subtract with carry instruction in this application. In general terms, processing speed is high enough to use slower algorithms in order to conserve memory.

The filtered throttle position is used to derive the load site and load interpolation steps. These are both numbers in the range 0 to 15, and fix precisely the load sites accessed on the ignition map. Load breakpoint preshaping is programmable, enabling the load breakpoints to be grouped closer together in an area of the map in which close throttle preshaping is required. Usually, the breakpoints are grouped closer together where the throttle first begins to open. The breakpoints are spaced wider as the throttle is opened further.

Engine speed is measured by timing the tooth period and filtering in a similar way to the throttle position. This parameter is global to the background task and the speed site is calculated as a number between 0 and 64. For a speed range of 16000 r/min, the speed breakpoints are fixed at 250 r/min, but for a reduced range, the breakpoint separation is programmable. Basically, the time for 250 r/min is divided by the tooth period to produce the speed site. The subtract with carry divide instruction is very useful here, because the remainder from the division is conveniently located in the high part of the accumulator. This is then used to calculate the speed interpolation steps.

Load and speed sites together with the interpolation steps are fed into the main ignition interpolation routine, which uses the four surrounding map sites to the engine operating position to calculate the interpolated ignition advance map value. This routine contains eight multiply and 4 additions, as well as data manipulation, and executes in 63 cycles, which is 15.75uS. For comparison this is 15 times faster than the same algorithm on the Motorola 6805 running at 4MHz.

The background task also handles diagnostics and telemetry via a serial communications routine, and measurement, filtering, and preshaping on the following ignition modifiers.

1) Air temperature
2) Coolant temperature
3) Barometric pressure
4) Overall trim
5) Boost pressure
6) Air humidity

The foreground task performs time critical control tasks, including the conversion of the ignition angle into a timer value which is loaded into the output compare structure. The primary tasks carried out in the input capture interrupt routine are as follows.

Synchronisation is initiated and maintained using missing tooth detection. On receipt of the tooth interrupt, the period between this tooth and the previous tooth is read from the input capture FIFO. If the missing tooth has either not been initially detected, or is expected then the missing tooth detection algorithm is implemented. A successful detection is valid if

$$\text{Tooth period} < 5/8 \times \text{previous period}.$$

After successful synchronisation, the tooth is identified, and calculations for the engine cycle are performed. Basically, the first tooth after TDC is called tooth 0, the next tooth 1 etc, up to tooth 9, on a V8 engine, when the cycle repeats itself for the next cylinder.

On tooth zero, the tooth to fire count is calculated, and decremented on each successive interrupt, until the

firing tooth is arrived at. The tooth to fire counter is calculated from the advance angle by dividing it by fifty.

The remainder from this division is used to calculate the advance degrees. When the tooth to fire count has decremented to 1, the time for ten degrees at this point is used to calculate the timer value, by multiplying the advance degrees by the period timer, and dividing this result by ten.

When the tooth to fire count is zero, the angle timer value is loaded into the appropriate compare register, and the action register is enabled, and the correct channel selected. When the timer matches the compare register, compare output will go high, triggering the CD circuit, and sparking.

In conclusion, the input capture interrupts are used for mathematical manipulation, loading timer values, counting teeth, and selecting the correct channel for the relevant cylinder. The overhead of these tasks is easily managed by the TMS320E14 at very high speed, whereas other conventional microprocessors simply cannot perform them in time. Hence, a single DSP can be used in place of a multiprocessor system.

5 FUTURE DEVELOPMENTS

Having achieved the ignition control performance required by current racing engines, Lucas are working on expanding the system to full engine management. With the current DSP microcontroller there are insufficient output lines to control 12 injectors as well as 12 channels of CDI. Consequently, it is the I/O limitation rather than CPU power that requires a slave processor to handle the fuel injector outputs. The intention is to use a TMS370 8 bit microcontroller to drive the injectors sequentially under direct control of the DSP controller. The majority of the fuelling calculations will take place in the current ignition controller with the slave processor being passed the appropriate injection timing information.

Whilst we have concentrated on the racing applications in this paper, Lucas have used this programme to measure the effectiveness of the DSP for Automotive engine control. New control strategies are being developed to enhance the performance of engine control for passenger cars in order to both increase efficiency and decrease emissions.

One of these strategies is adaptive ignition control whereby the control system applies small perturbations to the engine's running condition to determine the optimum torque/speed point. Lucas have great experience of such a technique and expect it to be applied in future production systems (2).

Another technique yet to be exploited in production is that of cylinder pressure sensing (3). In this case a pressure waveform is used to provide closed loop control of the engine. At present the main barrier to this technique is the availability of a robust, cost effective sensor. It is well known that there are several developments under way including ones internal to Lucas. However, the pressure signal inside a cylinder is of a complex form that requires much filtering and processing. DSP's have already been used in research applications to extract the information contained in this complex system - speed, combustion quality, engine health, etc. Again it is the real-time digital filtering ability of the DSP that is its strength for this function. The controller described appears to have sufficient spare capacity to be able to handle a cylinder pressure sensor - the story is common, the electronics are available and cost effective, but it is the sensor technology that is lacking.

6 CONCLUSIONS

It has been effectively demonstrated that a microcontroller with DSP functions included can provide the core for a high performance ignition controller. The efficiency of the instruction set coupled with its speed of operation would allow engine management to be carried out on a single chip - the limiting factor is the amount of timer driven I/O available on the current device. The merits of having a very fast processing core can be summarised as:-

1) Control data updated closer to the time it is used.

2) No tradeoff of control functions against engine speed.

3) Opportunity to include new control techniques in single processor system, i.e. cylinder pressure sensing.

The TMS320E14 is the first step at availing the DSP functionality in a microcontroller device - it is expected that the lessons learnt from this and other automotive control projects will further enhance its capability as new devices are brought to production.

This system should not be viewed solely as a faster version of current systems, but rather one which may be used to effectively apply the more complex strategies required of engine management systems in the 1990's (4) - and in a reliable and cost effective manner.

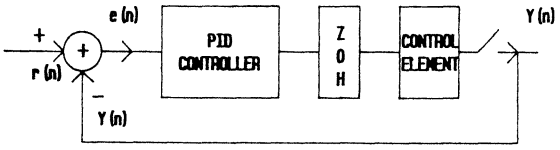
REFERENCES

(1) LIN, K. Trends of digital signal processing in automotive. International Congress on Transportation Electronics, Dearborn, 17-18 October, 1988.

(2) HOLMES, M. and COCKERHAM, K. Adaptive ignition control. 6th International Conference on Automotive Electronics, London, 12-15 October, 1987.

(3) ANASTASIA, C.M. and PESTANA, G.W. A cylinder pressure sensor for closed loop engine control. SAE International Congress and Exposition, Detroit, 23-27 February, 1987.

(4) HATA, Y. and ASANO, M. New trends in electronic engine control - to the next stage. SAE International Congress and Exposition, Detroit, 24-28 February, 1986.



$y(t) = Kp*e(t) + Ki\ e\ dt + Kd*de/dt$

e(t) = error signal. Kp, Ki & Kd = PID constants.

Converting into discrete form (using rectangular approx.):

$y(n) = y(n-1) + K0*e(n) + K1*e(n-1) + K2*e(n-2)$

K0 = Kp + Kd/T + Ki*T, K1 = -Kp -2Kd/T, K2 = Kd/T

Where T = sampling interval.

CODE IMPLEMETATION

```
IN    E0, PA0    GET NEW SAMPLE
MPYK  0          CLEAR P REG
LAC   YN         ACC=y (n-1)
LT    E2
MPY   K2
LTD   E1         ACC=y (n-1) +K2e (n-2)
MPY   K1
LTD   E0         ACC=y (n-1) +K1e (n-1) +
MPY   K0                        K2e (n-2)
APAC             ACC=y (n-1) +K0e (n) +
SACH  YN               K1e (n-1) +K2e (n-2)
OUT   YN, PA1
```

EXECUTION TIME = 2.2uS @ 25MHz

Fig 1 PID control algorithm

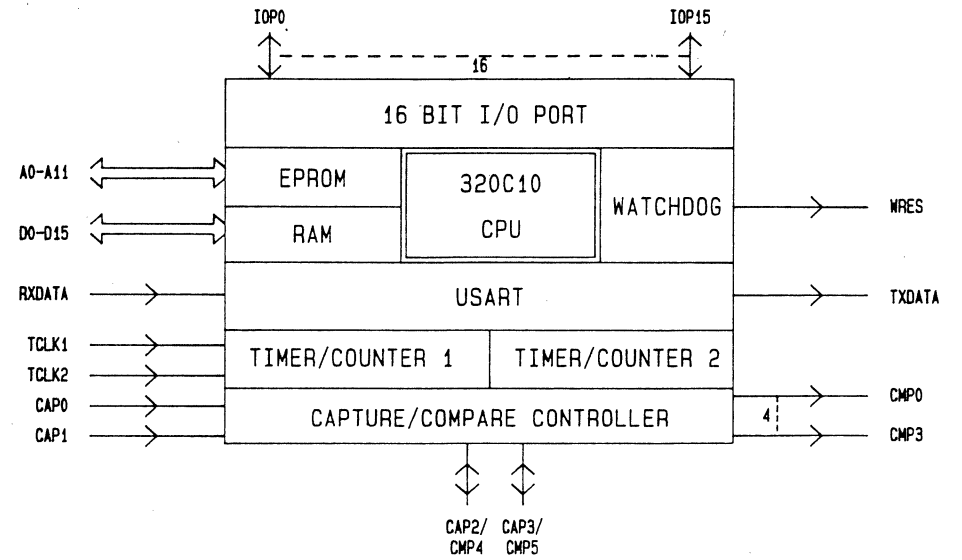


Fig 2 TMS320E14 hardware organisation

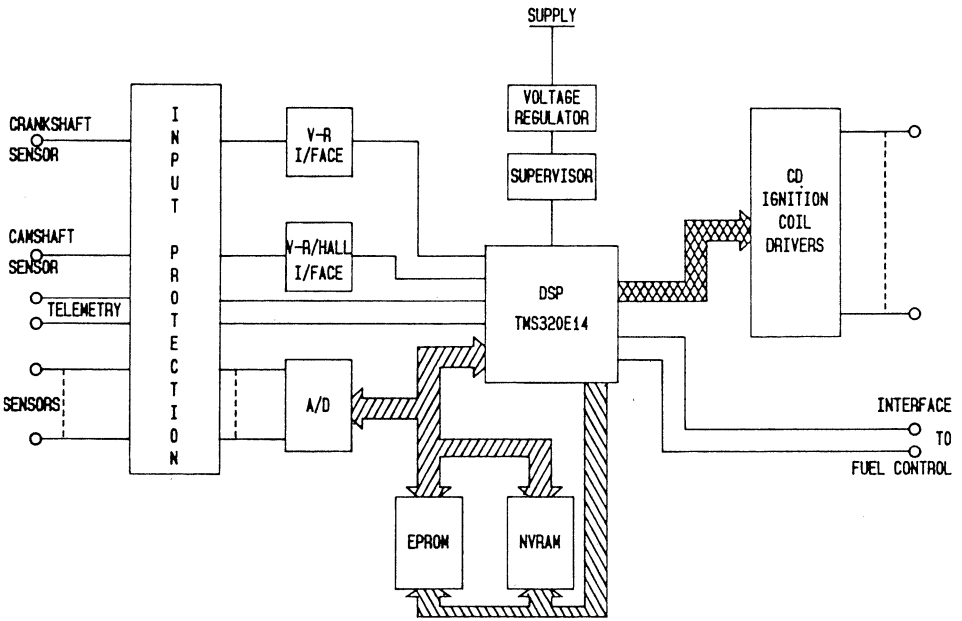


Fig 3 Lucas racing CDI system

Active Reduction of Low-Frequency Tire Impact Noise Using Digital Feedback Control

Mark H. Costin and Donald R. Elzinga

ABSTRACT: Feedback control theory is used to develop an active noise control system to reduce transient-induced road noise in a vehicle interior. The system consists of a detector microphone, a high-speed digital controller, amplifiers, an analog smoothing filter, and a headphone. The digital control algorithm uses the output of the microphone combined with the past history of the control signal to calculate the current value of the control signal. This signal is passed through a low-pass filter (to smooth the steps resulting from the digital-to-analog conversion) and then amplified and sent to the headphone near the driver's ear. Two control algorithms are evaluated. A proportional-integral controller reduced the noise by about 5 dB over the 20–60 Hz range. A modified generalized minimum variance controller was able to reduce the noise by about 10 dB for the 25–60-Hz range.

Introduction

This paper presents a system for reducing broadband, low-frequency noise. The particular application described here is the reduction of road noise in passenger vehicles; however, the concept can be used for other applications as well.

Typical noise-reducing strategies, such as the use of acoustic absorbing material, work well on high-frequency sounds, but have little effect on noise in the 20–200 Hz range. This frequency range is important because the vehicle's tires and suspension act as low-pass filters. This results in a rough road–induced sound spectrum, which typically peaks around 100 Hz. The problem is especially acute in vehicles with large passenger compartments and large amounts of body structural motion, which create high levels of low-frequency noise. Station wagons and vans are good examples of such vehicles.

The method investigated here to reduce the unwanted low-frequency sound is known as active noise control. This technique consists of broadcasting sound with the same amplitude as, but 180 deg out of phase with, the objectionable noise, thereby canceling it. In actual practice, reductions of over 20 dB have been obtained for periodic noise and for broadband noise in a duct [1]. Reductions of this magnitude require the canceling signal to match the unwanted sound fairly precisely; small errors in the gain or phase rapidly degrade the performance of the system.

Active noise control has been implemented for the case of periodic noise by assuming that the waveform within the upcoming period is identical to that which preceded it. In the case of a duct, feedforward control can be used. The noise is measured at one point upstream by a detector microphone; the canceling signal is then sent to a speaker positioned downstream. The noise reaches the speaker just as its antiphase counterpart is being generated.

In the case of nonperiodic broadband noise in a vehicle's interior, there is no way to measure the offending sound before it reaches the driver. For this case, the use of a feedback controller to reduce the noise is examined. The use of a feedback controller for this situation was first proposed by Olson and May [2]; however, their feedback consisted simply of an analog amplifier (gain), which does not take into account any system dynamics. Ffowcs Williams [3] reported that attempts to duplicate Olson and May's experimental results have revealed severe instability problems. The approach outlined here tries to address the instability problem by detailed system modeling, for both the system electronics and noise characteristics, and by designing a digital control algorithm using minimum variance control theory.

System Configuration and Modeling

The experiments described in this report were carried out in a midsized station wagon passenger vehicle. A foam-rubber ball was positioned where the driver's head would normally be located. A microphone was embedded in the ball at the driver's right ear location. A production active noise control system would normally use a speaker as the canceling noise actuator. However, to simplify the system for this feasibility study, a headphone set was placed over the ball (and microphone) to act as the canceling speaker.

The control algorithms were implemented using a 320/PC digital signal processing (DSP) board made by Atlanta Signal Processors, Inc. This is an add-in board for the IBM PC-AT. It includes a Texas Instruments TMS 32010 DSP chip to perform the required high-speed arithmetic and a digital-to-analog (D/A) converter and an analog-to-digital (A/D) converter for control input and output. Modeling data and control results were collected using a Metrabyte Dash-16 A/D board.

All the experiments were performed with the vehicle stationary in the lab. "Road noise" was generated by striking the right front tire with an air hammer. This generated a repeatable input that approximately simulated driving over a bump in the road about the size of a 2-cm-high tar strip at 50 km/hr.

The block diagram of the feedback control system is given in Fig. 1, with a microphone as the sensor and headphones as the actuator. Figure 1 also includes a shaping filter after the D/A. This filter is necessary because the output of the D/A is a series of steps that change in level at the discrete sampling intervals. If not removed, these discrete steps create a buzzing noise. Therefore, an analog low-pass filter (a fourth-order Bessel filter with a cutoff frequency of 300 Hz) was used in the system. The output of the filter goes into an amplifier that is used to adjust the gain of the system.

To produce the desired canceling signal, models for the noise to be canceled and the dynamics of the components shown in Fig. 1 must be determined. Using time-series analysis, a discrete-time model of the following form can be developed [4], where

Mark H. Costin is with the Electrical and Electronics Engineering Department and Donald R. Elzinga was with the Engineering Mechanics Department, General Motors Research Laboratories, Warren, MI 48090. Donald Elzinga's current address is CPC Engineering North, 895 Joslyn Road, Pontiac, MI 48058.

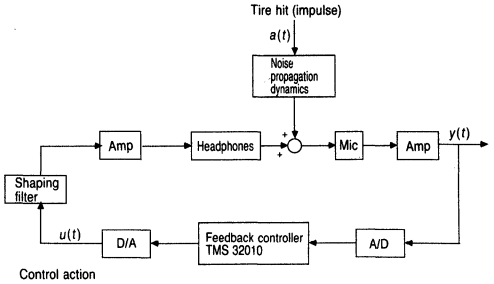


Fig. 1. System block diagram.

$y(t)$ is the measured variable, $u(t)$ the system input, $a(t)$ a random-noise component, t the integer number of sampling intervals, and f the integer-valued system delay. The functions $\omega(q)$, $\delta(q)$, $\theta(q)$, and $\phi(q)$ are polynomials in the backward shift operator q^{-1}, where $q^{-1}u(t)$ equals $u(t-1)$.

$$y(t) = \frac{\omega(q)}{\delta(q)}\,u(t-f-1) + \frac{\theta(q)}{\phi(q)}\,a(t) \quad (1)$$

The first term on the right-hand side of Eq. (1) is referred to as the system transfer function and the second term the noise transfer function.

The noise detected by the microphones resulting from the "bump" (striking of the right front tire) is given in Fig. 2. The noise model, the second part of Eq. (1), was determined from these data at a data sampling rate of 2 kHz using the commercial time-series analysis package "SCA" [5]. This yielded Eq. (2), with $\theta = 0.38$, $\phi_1 = 1.97$, and $\phi_2 = -0.98$, where $P_{ol}(t)$ (open-loop pressure) is the time series of the signals measured by the microphones during the "bump."

$$P_{ol}(t) = F_a(q)\,a(t) = \frac{(1-\theta q^{-1})}{(1-\phi_1 q^{-1}-\phi_2 q^{-2})}\,a(t) \quad (2)$$

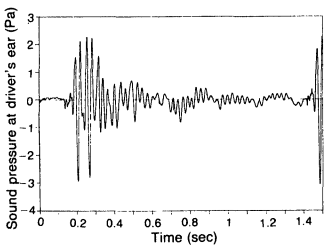


Fig. 2. Response of microphone to striking the tire twice, with the second tire strike occurring at approximately 1.4 sec.

The physical interpretation of Eq. (2) is that the time series $a(t)$ is an impulse corresponding to striking the tire. $P_{ol}(t)$ is the impulse response of the transfer function, which represents the filtering of the impulse by the tire and the structure of the vehicle.

The denominator of Eq. (2) models a damped sinusoid. Box and Jenkins [4] give the formulas below for determining the frequency that the ϕ coefficients are modeling, where f_0 is the frequency of the sinusoid in cycles per sample and d the damping factor.

$$\phi_1 = 2d\cos(2\pi f_0)$$

where

$$d = -(\phi_2)^{1/2}$$

Therefore, the model given by Eq. (2) corresponds to a slightly damped sinusoid ($d = 0.99$) with a frequency of 34.0 Hz. The magnitude-versus-frequency plot for this model is given in Fig. 3, which compares very well to the same plot for the data of the tire strike, Fig. 4. Note that the cutoff frequency is approximately 60 Hz.

Next, the system transfer function (representing the shaping filter, canceling signal amplifier, headphone, microphone, and microphone amplifier), the first part of Eq. (1),

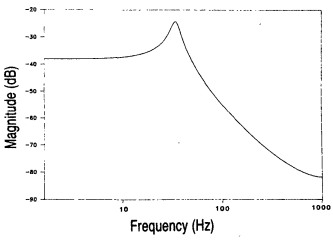


Fig. 3. Magnitude-versus-frequency plot of Eq. (3) (amplitude at 5 Hz normalized to approximately the same value as in Fig. 4).

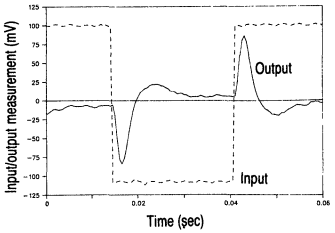


Fig. 4. Spectrum of tire hit.

was modeled. Figure 5 shows the response of this system to a 40-Hz square wave. To identify this system, it was excited by white noise filtered at 500 Hz, sampled at 2 kHz (the sampling rate of the controller). A least-squares analysis was performed between the measured output and the input yielding Eq. (3). A gain term represented by k was included to represent the variable gains of the amplifiers.

$$kF_u(q)\,u(t-2) = \frac{k(1+0.14q^{-1})(1-0.99q^{-1})}{(1-1.53q^{-1}+0.67q^{-2})} \cdot u(t-2) \quad (3)$$

The denominator of Eq. (3) represents an underdamped sinusoid with a natural frequency of 116.6 Hz. The first polynomial of the numerator $(1 + 0.14q^{-1})$ models a partial delay, which, along with the one whole period of deadtime, models the shaping filter. The $(1 - 0.99q^{-1})$ term is very close to $(1 - q^{-1})$, which corresponds to a derivative in the model. The presence of a derivative is expected because of the almost zero steady-state gain observed in the square-wave tests of Fig. 5.

The overall system model is obtained by combining Eqs. (2) and (3), leading to the following expression for $y(t)$, where k and

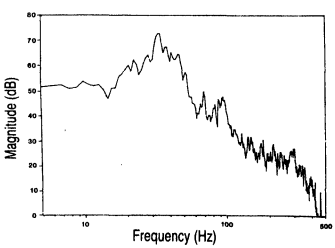


Fig. 5. Step response of shaping filter-amplifier-headphone-microphone-amplifier system.

the variance of $a(t)$ depend on the gains of the amplifiers.

$$y(t) = kF_u(q)_u(t-2) + F_a(q)\,a(t) \quad (4)$$

Minimum Variance Control Theory

For systems modeled in the form of Eq. (1), an effective controller design methodology is minimum variance control theory [6]. This technique finds the control that minimizes the variance of the measurement $y(t)$.

The minimum variance controller for Eq. (1) can be shown as

$$u(t) = \frac{-\delta(q)\,T(q)}{\omega(q)\,\phi(q)\,\Psi(q)}\,y(t) \quad (5)$$

The functions $T(q)$ and $\Psi(q)$ are polynomials derived from the identity

$$\frac{\theta(q)}{\phi(q)} = \Psi(q) + \frac{T(q)\,q^{-f-1}}{\phi(q)} \quad (6)$$

where

$$\Psi(q) = 1 + \Psi_1 q^{-1} + \cdots + \Psi_f q^{-f}$$

The minimum variance controller [Eq. (5)] has the property of reducing $y(t)$ to $\Psi(q)\,a(t)$.

The closed-loop controller described by Eq. (5) also has its limitations. Equation (5) has the term $\Psi(q)$ in its denominator. If $\Psi(q)$ has any roots in q^{-1} inside the unit circle, the controller will be unstable. This means that, theoretically, to achieve minimum variance control, the variance of $u(t)$ will be unbounded. In a practical implementation, the high output energy of an unstable controller will cause the entire system to be unstable.

Unfortunately, for the model described by Eq. (4), $\Psi(q) = (1 + 1.96q^{-1})$ has its root inside the unit circle. For cases with unstable controllers, many modifications to minimum variance control have been proposed. One of the most popular is called the *generalized minimum variance* (GMV) *controller* [7], where the variance of $P(q)y(t)$ is minimized instead of the variance of $y(t)$ [$P(q) = P_N(q)/P_D(q)$ is a digital filter, and $P_N(q)$ and $P_D(q)$ are polynomials in q^{-1}].

The minimum variance controller [Eq. (5)] also has the disadvantage that it minimizes the measured signal with equal weighting on all frequencies. This is not always desirable. For example, since sounds below 20 Hz are virtually inaudible and difficult to reproduce with a small speaker, we are not interested in controlling these frequencies. The $P(q)$ filter of the GMV algorithm can also be used to selectively weight certain frequencies. $P(q)$ could, for example, be made to approximate the A-weighting curve, which approximates the filtering done by a typical human's auditory system. The controller would then minimize the noise as sensed by the vehicle's occupants.

The GMV controller can be shown as

$$u(t) = \frac{-\delta(q)\,\bar{T}(q)\,y(t)}{P_D(q)\,\omega(q)\,\phi(q)\,\bar{\Psi}(q)} \quad (7)$$

The functions $\bar{T}(q)$ and $\bar{\Psi}(q)$ are polynomials derived from the following identity, which is similar to Eq. (6).

$$\frac{\theta(q)\,P_N(q)}{\phi(q)\,P_D(q)} = \bar{\Psi}(q) + \frac{\bar{T}(q)\,q^{-f-1}}{\phi(q)\,P_D(q)}$$

where

$$\bar{\Psi}(q) = 1 + \bar{\Psi}_1 q^{-1} + \cdots + \bar{\Psi}_f q^{-f}$$

The minimum variance controller [Eq. (5)] is a special case of Eq. (7), where $P(q) = 1$.

The application of controller (7), and also a simple proportional-integral (PI) controller, to the active noise control problem is described in the following section.

Closed-Loop Control Results

To more easily describe and compare the controllers, all the controllers were implemented as $u(t) = \alpha(q)\,y(t)/\beta(q)$, where $\alpha(q)$ and $\beta(q)$ are polynomials of controller parameters and N and M are the maximum number of controller coefficients.

$$\alpha(q) = \alpha_0 + \alpha_1 q^{-1} + \cdots + \alpha_N q^{-N}$$

$$\beta(q) = 1 + \beta_1 q^{-1} + \cdots + \beta_M q^{-M}$$

In the remainder of this section, when describing an individual controller, a parameter's value is zero unless it is indicated explicitly.

Two types of controllers were implemented in the TMS 32010. The first controller tried was a PI controller, where $N = 1$, $M = 1$, $\beta_0 = 1$, and $\beta_1 = -1$. The parameters α_0 and α_1 were determined by trial and error as $\alpha_0 = 2.94$ and $\alpha_1 = -1.94$. The denominator was implemented as $\beta_1 = -0.98$ (as opposed to $\beta_1 = -1$) to reduce numerical round-off problems.

The second controller, a GMV controller, was designed for Eq. (4) using Eq. (7). $P(q) = (1 - 0.6q^{-1})/(1 + 0.6q^{-1})$ was used to force $\Psi(q)$ to be stable. This form for $P(q)$ was selected by computer simulation of system model (4). The resulting controller was determined to have the following parameter values: $\alpha_0 = 0.955$, $\alpha_1 = -1.97$, $\alpha_2 = 1.20$, $\alpha_3 = 0.0014$, $\alpha_4 = -0.151$, $\beta_1 = -1.83$, $\beta_2 = -0.0361$, $\beta_3 = 1.26$, $\beta_4 = -0.103$, $\beta_5 = -0.263$, and $\beta_6 = -0.0321$.

Unfortunately, the GMV controller designed for Eq. (4) performed very poorly. When implemented, the system became unstable even before the disturbance was introduced by hitting the tire. The instability is thought to be caused by the controller's very high gain at low frequencies. This high gain is due to the formulation of the minimum variance controller, which inverts the model transfer function. For the case of Eq. (3), the term $(1 - 0.99q^{-1})$ models very low gains at low frequencies, resulting in a minimum variance controller with very high gains at low frequencies. Because of this low gain and the modeling technique used, the low-frequency components of the system probably were modeled inaccurately. This could cause any minimum variance controller designed from this model to exhibit undesirable properties at low frequencies (i.e., inadequate gain and phase margins).

To remedy this, an ad hoc controller design was performed that involved recalculating the GMV controller using Eq. (4) after the $(1 - 0.99q^{-1})$ term was removed from the transfer function. The result was a modified GMV controller with $\alpha_0 = 0.955$, $\alpha_1 = -1.97$, $\alpha_2 = 1.20$, $\alpha_3 = 0.0014$, $\alpha_4 = -0.151$, $\beta_0 = 1$, $\beta_1 = -0.839$, $\beta_2 = -0.867$, $\beta_3 = 0.405$, $\beta_4 = 0.298$, and $\beta_5 = 0.0325$. This controller was stable and provided good control. Although this modified GMV controller is probably not optimal, as discussed later, it performed better than the PI controller, demonstrating the feasibility of the concept and the model-based controller design.

Figure 6 shows the spectrum of the tire hit for the uncontrolled sound, and PI and modified GMV controllers. The PI controller shows a reduction of 5–10 dB for the 20–60 Hz interval. The modified GMV controller shows a 10–20 dB reduction between 25 and 60 Hz. The reduction is limited above 60 Hz because striking the tire introduces very little noise above this frequency (the cutoff fre-

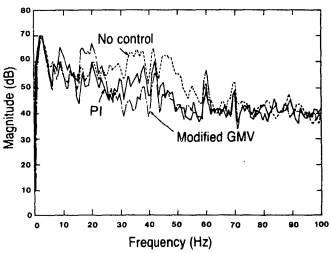


Fig. 6. Comparison of open-loop, PI control, and modified GMV control tire hit spectra.

quency of Figs. 3 and 4). From about 15 to 20 Hz, the modified GMV controller exhibits an amplification of 5–10 dB; however, these frequencies are on the border of the audible range and would not be heard by most people.

Acknowledgments

The authors acknowledge the contributions to this work of L. J. Oswald, Section Manager of the Engine Dynamics and Acoustics Section, Engineering Mechanics Department, who generously provided background and insight into this and other active noise control problems.

References

[1] C. F. Ross, "A Demonstration of Active Control of Broadband Sound," *J. Sound and Vibration*, vol. 74, no. 3, pp. 135–140, 1981.

[2] H. F. Olson and E. G. May, "Electronic Sound Absorber," *J. Acoustical Soc. Amer.*, vol. 25, no. 6, pp. 1130–1136, 1957.

[3] J. E. Ffowcs Williams, "Anti-Sound," *Proc. Roy. Soc. Lond., Ser. A*, vol. 395, pp. 63–88, 1984.

[4] G. E. P. Box and G. M. Jenkins, *Time Series Analysis: Forecasting and Control*, revised edition, San Francisco, CA: Holden-Day, 1976.

[5] L.-M. Lui and G. B. Hudak, *The SCA System for Univariate-Multivariate Time Series and General Statistical Analysis*, ver. II, DeKalb, IL: Scientific Computing Associates, 1985.

[6] K. J. Astrom, *Introduction to Stochastic Control Theory*, New York, NY: Academic Press, 1970.

[7] D. W. Clarke and B. A. Gawthrop, "Self-Tuning Control," *Proc. IEE*, vol. 126, pp. 633–640, 1979.

Mark H. Costin received the B.Eng. degree in chemical engineering with a minor in computer science from McGill University in 1979, the M.Eng. degree in chemical engineering from McMaster University in 1981, and the Ph.D. degree in systems engineering from Case Western Reserve University in 1984. Since 1984, he has been a member of the Electrical and Electronics Engineering Department, General Motors Research Laboratories, where he has been performing research in control theory and application for various automotive and manufacturing systems. His current research interests concern system identification, adaptive control, and failure detection.

Donald R. Elzinga received the B.S. and M.S. degrees in mechanical engineering from Michigan Technological University in 1984 and 1985, respectively. From 1985 to 1987, he was a member of the Engineering Mechanics Department, General Motors Research Laboratories, working in the field of vehicle noise and vibration. Since 1987, he has worked at the CPC Division of General Motors. His current professional interests include acoustics and modal analysis.

Implementation of a Tracking Kalman Filter on a Digital Signal Processor

JIMFRON TAN AND NICHOLAS KYRIAKOPOULOS

Abstract—**A Kalman filter for tracking moving objects has been implemented on a TMS32010 digital signal processor. Tracking accuracy and quantization effects of the implementation have been measured by comparing the filter to one implemented on a general purpose computer with a 32-bit word length. The filter design has been optimized to minimize the program memory requirements and execution speed. Although the filter has been implemented on a specific signal processing chip, the design is general enough to be applicable to any other digital signal processor. The filter can be used for tracking objects for industrial or other applications where range and bearing measurements are available. For motion on a plane, the filter can be used to track objects where the maximum system bandwidth is 1680 Hz; for three-dimensional motion the system bandwidth is 1120 Hz. Using the approach presented in this paper higher system bandwidths can be accommodated through higher speed digital signal processors.**

I. Introduction

THE theory of Kalman filters is by now well covered in the literature [1], [2], [3]; applications can be found in any area where the problem can be modeled as a dynamic process. Implementation of even the most efficient algorithms requires rather heavy computational capacity. Memory requirements are dominated by program instructions for systems with a small number of states, and by matrix storage for large state sizes. For program execution, the largest amount of time is taken up by multiplications and additions in the computation of the covariance matrix; in general, the number of these multiplications is proportional to the third power of the state size [4].

Recently, the trend in special purpose signal processing devices has been toward the integration of array multipliers with the ALU; as a result, there has been an improvement in the multiplication time compared to the software implementation of the multiply instruction. The improvement in the instruction execution time makes the on-line, real-time implementation of Kalman filters for industrial applications a realistic possibility. This paper discusses the implementation of such a filter on a special purpose digital signal processor. Some of the currently available devices such as the NEC μPD7720, and the Texas Instruments TMS320 are capable of 4.0×10^6 and 5.0×10^6 multiplications per second, respectively. The Kalman filter described in this paper is implemented on the TMS320.

Since the multiplication time and memory capacities of these devices are fixed, the objective of the design presented in this paper is to minimize cycle time and maximize the state size of the filter. At the same time, the arithmetic roundoff errors are bounded.

The filter implemented in this paper is a tracking filter where the state variables are position and velocity. Tracking filters have a wide range of applications from performing mechanical operations such as controlling the motion of robot arms to the sensing of objects through radar or sonar. The implementation presented in this paper is in terms of a normalized system of units; it is thus applicable to any problem formulated as object tracking.

Section II describes the formulation of the tracking problem. Section III gives the development of the Kalman filter. The details of the program design are given in Section IV, while the evaluation of the program is described in Section V.

II. Formulation of the Tracking Problem

The tracking problem considered in this paper assumes motion of an object on a plane; three-dimensional motion can be handled through repeated use of the two-dimensional system. The problem can be viewed either as an object moving with respect to a sensor or the converse; the two situations are handled through a simple coordinate transformation. It is assumed that range and bearing are measured independently; therefore, these two measurements are decoupled and the polar coordinate system is used. This decoupling of states is essential to the optimization of the computer program since the number of multiplications for the covariance matrix is proportional to the third power of the state size; thus the number of multiplications for estimating position and velocity in three-dimensional motion would be $6^3 = 216$ for coupled systems versus $3 \times 2^3 = 24$ for a decoupled one.

Consider an object moving on a plane. Let the sampling frequency be high enough so that the object speed between any two sequential sampling instances can be considered constant; every change occurs at the sampling instances, and those changes are disturbed by random accelerations. In the polar system, along each coordinate, there is the associated variable $x_1(k)$ and its corresponding rate of change $\dot{x}_1(k) = x_2(k)$. For each coordinate the equations of motion are

$$\boldsymbol{x}(k+1) = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \boldsymbol{x}(k) + \begin{bmatrix} \frac{T^2}{2} \\ T \end{bmatrix} w(k)$$

$$= F(k)\boldsymbol{x}(k) + G(k)w(k) \tag{1}$$

Manuscript received April 25, 1986; revised March 5, 1987.

J. Tan is with the Harbin Shipbuilding Engineering Institute, Harbin, Peoples Republic of China.

N. Kyriakopoulos is with George Washington University, Washington, DC.

IEEE Log Number 8718157.

where T is the sampling period, $w(k)$ is the random acceleration disturbance, and

$$F(k)=\begin{bmatrix} F_{11}(k) & F_{12}(k) \\ F_{21}(k) & F_{22}(k) \end{bmatrix}=\begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix}.$$

The measurements consist of range and bearing readings; each measurement variable $z(k)$ is corrupted by additive noise $v(k)$; therefore

$$\begin{aligned} z(k)&=[1\ 0]x(k)+v(k) \\ &=H(k)x(k)+v(k) \end{aligned} \tag{2}$$

where

$$H(k)=[H_1(k)\ H_2(k)].$$

III. Development of the Kalman Filter

The equations describing the Kalman filter for the system described by (1) and (2) have been developed in the literature [1], [2]. The most time consuming process in the implementation of the filter is the multiplications required for computing the error covariance matrix and the filter gain. For the tracking problem considered in this paper the number of operations is minimized by converting the filter matrix equations into scalar forms. These scalar equations require fewer assembly language instructions than the original form of the filter.

Designating the error covariance matrix as

$$P=\begin{bmatrix} P_{11} & P_{12} \\ P_{12} & P_{22} \end{bmatrix}$$

the extrapolated covariance matrix as

$$S=\begin{bmatrix} S_{11} & S_{12} \\ S_{12} & S_{22} \end{bmatrix}$$

and the extrapolated state estimate as y, the filter equations are

$$y=F\hat{x} \tag{3}$$

$$S=FPF^T+GQG^T \tag{4}$$

$$\Gamma=S\ H^T\ [H\ S\ H^T+R]^{-1} \tag{5}$$

$$\hat{x}=y+\Gamma[z-Hy] \tag{6}$$

$$P=[I-\Gamma H]S \tag{7}$$

where Γ is the filter again. The scalar equations (3)–(7) are the basis for the assembly language program of the digital signal processor. The flowchart for the filter implementation on the TMS320 processor is shown in Fig. 1.

IV. Program Design

The development of the filter equations has been influenced by the architecture of the digital signal processor. If the Kalman filter is implemented in terms of array operations, a single subroutine executing vector multiplication can be defined; such a subroutine can be called whenever vector operations are needed. If, on the other hand, the filter equations are implemented in scalar form, the call and return times for the subroutines are eliminated; the program memory

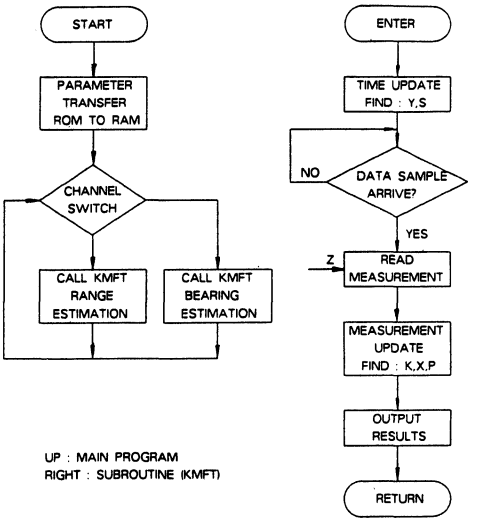


Fig. 1. Kalman filter implementation on the TMS320 digital signal processor.

requirements for the scalar implementation are much greater than those for the vector implementation. This imbalance is corrected by the improvement in the execution time. The total time required for generating an output data array can be measured in instruction cycle times. For array multiplication the total number of instruction cycle times consists of the time required to operate on the array elements, the time required to call and return the array subroutine, and the number of times the array subroutine is called. For scalar operations, the call and return times are saved. Thus if the total number of instruction cycles for executing an array operation is comparable to the instruction cycles required for calling and returning a subroutine, the scalar formulation of the equations yields a faster filter implementation. For example, the inner product of two n-dimensional arrays requires n multiplications and $(n-1)$ additions. Since the instruction cycle times nT is typically equal to the multiply time, the total time required for such an operation is nT. If the inner product is programmed as a subroutine, the call and return times require more than one instruction cycle time, or αT where $\alpha > 1$; the total time required for calling and executing the subroutine is $(nx\alpha)T$. Consider now the multiplication of two $n \times n$ matrices using the array multiplying subroutine; the total number of cycle times would be $n(n+\alpha)T$. In a scalar implementation of the same operation, the total number of cycle times would be n^2T; thus the ratio of the scalar implementation execution time to the vector implementation is $n/(n+\alpha)$. It is seen that as $n \gg \alpha$ the ratio tends to 1 and the saving in execution time is insignificant compared to the increased amount of memory required to store the instruction required for the scalar implementation. For the two degrees of freedom under consideration in this paper, $n = 2$, and the ratio becomes $2/(2+\alpha)$; since α is always greater than 1, the scalar implementa-

tion of the Kalman filter will be at least twice as fast as the corresponding vector realization.

A. The Digital Signal Processor

The Texas Instruments TMS32010 [5] was chosen for the implementation of the Kalman filter described in the previous sections. The approach to the filter design described in this paper is equally applicable to any other signal processing chip. There are, however, some general observations that would be useful in choosing a particular DSP chip for implementing a filter. It has been seen that the decrease in processing time is accompanied with an increase in program memory requirements. Some DSP chips partition the ROM into a program ROM and a data ROM in addition to the data RAM; others have only data RAM and program ROM. The Kalman filter algorithm is instruction intensive with minimal requirements for data storage; therefore a DSP architecture with a large nonpartitioned ROM and relatively small RAM would allow the implementation of a larger dimension Kalman filter in the scalar form, thus decreasing the computation time for obtaining the state estimates. From the faster available devices the NEC μpD7720 has 512 $\times$ 23 program ROM and 512 $\times$ 23 data ROM, while the TMS 32010 has 1536 $\times$ 16 program ROM only; therefore, on the basis of the previous considerations it was considered more suitable.

B. Word Length Considerations

The finite length registers of the DSP chip affect the accuracy of the filter; the fixed point representation of the numbers can cause overflow or underflow. The effects of parameter quantization can be studied either analytically or by comparing the performance of the filter to a similar one with much larger word length. For this paper, the latter option has been chosen; the reference filter has been implemented on a 32-bit general purpose machine in both integer and floating point realizations; the results are discussed in a subsequent section.

To avoid the overflow or underflow problems associated with integer arithmetic the filter equations were scaled appropriately. Proper scaling factors for each variable and parameter were determined from the implementation of the filter on the general purpose computer using floating point arithmetic. The filter variables and parameters for each sample point were printed out under a wide range of operating conditions. Ideally, the statistics of these variables should be determined using Monte Carlo simulation; in practice, however, reasonable values can be obtained by operating the filter under conditions close to the maximum range and bearing. Once the ranges of values for the variables have been obtained from the large word length computer, considered the reference standard, the scaling factors for the DSP implementation are chosen by also taking into account the required accuracy for each variable.

Let $x_{\max}$ be the maximum value of a variable x, and Δx the corresponding accuracy. The number of bits, M, required to accommodate $x_{\max}$ would be $\log_2 x_{\max} \leq M$; to obtain the required accuracy, the number of necessary bits must satisfy $2^{-N} \leq \Delta x$. For a 16-bit DSP, $M + N = 15$ since one bit is reserved for sign. The dynamic range of the system is $x_{\max}/\Delta x = 2^{15} = 32768$. Obviously, there is a tradeoff between the maximum range of a variable and the corresponding accuracy.

TABLE I
INITIAL CONDITIONS, SCALING FACTORS AND THE NORMALIZED VALUES USED IN IMPLEMENTING THE KALMAN FILTER IN THE TMS 320

Parameters or Variables	Initial Values	Scaling Factors	Normalized Initial Values
F_{11}	1	1	1
F_{12}	0	1	0
F_{21}	2^{-4}	2^4	1
F_{22}	1	1	1
G_1	2^{-4}	2^4	1
G_2	2^{-9}	2^9	1
H_1	0	1	0
H_2	1	1	1
X_1, X_2	0	2^7	0
P_{11}	100	2^4	1600
P_{12}	0	2^8	0
P_{22}	100	2^8	25600
S_{11}		2^4	
S_{12}, S_{22}		2^8	
Y_1, Y_2, Z, V		2^7	
D_1, D_2, W		2^4	
σ_Q^2 (range)	900	2^{-4}	56
σ_R^2 (range)	1	2^4	16
σ_Q^2 (bearing)	360^2	2^{-4}	8100
σ_R^2 (bearing)	9	2^4	144

For the system under consideration the maximum values for the range and bearing were 200 units and 180°, respectively; the corresponding accuracy was 0.01. Therefore, $\log_2 200 < 8 = M$, and $2^{-7} < 0.01$ so that $N = 7$. The scaling factors for the computed variables are determined by examining the range of these variables from the simulation runs. For example, the maximum value of P_{11}, (4) was approximately 1000 so that $M \geq \log_2 1000$, or $M = 10$ bits. The error covariance at every sample point depends on the assumed initial values for this variable; for the present simulation only a few different values of initial conditions were used so that there is a certain degree of uncertainty for the value $P_{11\,\max}$. To minimize possible computational errors due to this uncertainty, M was increased to 11 bits, implying a $P_{11\,\max}$ of 2048. Under these conditions, $N = 15 - M = 4$ bits and the scaling factor for P_{11} is taken as $2^4 = 16$. Similarly, the scaling factor for S_{11} has been computed as 2^4. The scaling factors for all the variables are shown in Table I. Application of these factors to the filter equations (3)–(7) yields the scaled version of the Kalman filter given in Table II, which has been implemented in the TMS320 DSP.

C. Reference Program

The performance of the filter has been evaluated by comparing it to a filter implemented on an IBM 370 machine.

TABLE II
NORMALIZED KALMAN FILTER EQUATIONS

$$y_1 = F_{11}x_1 + F_{12}x_2$$

$$y_2 = F_{12}x_1 + F_{22}x_2$$

$$S_{11} = F_{11}^2 P_{11} + F_{11}F_{12}P_{12}/8 + F_{12}^2 P_{22}/16 + G_1^2\sigma_Q^2$$

$$S_{12} = F_{11}F_{12}P_{11} + F_{12}F_{21}P_{12}/16 + F_{11}F_{22}P_{12} + F_{11}F_{22}P_{22} + G_1G_2\sigma_Q^2/2$$

$$S_{22} = F_{21}^2 P_{11}/16 + F_{21}F_{22}P_{12}/8 + F_{22}^2 P_{22} + G_2^2\sigma_Q^2/64$$

$$D_1 = S_{11}H_1 + S_{12}H_2/16$$

$$D_2 = S_{12}H_1/16 + S_{22}H_2/16$$

$$W = H_1D_1 + H_2D_2 + \sigma_R^2$$

$$\gamma_1 = D_1/W$$

$$\gamma_2 = D_2/W$$

$$V = z - (H_1y_1 + H_2y_2)$$

$$\hat{x}_1 = y_1 + \gamma_1 V$$

$$\hat{x}_2 = y_2 + \gamma_2 V$$

$$P_{11} = S_{11} - D_1^2/W$$

$$P_{12} = S_{12} - D_1D_2/(W \cdot 16)$$

$$P_{22} = S_{22} - D_2^2/(W \cdot 16)$$

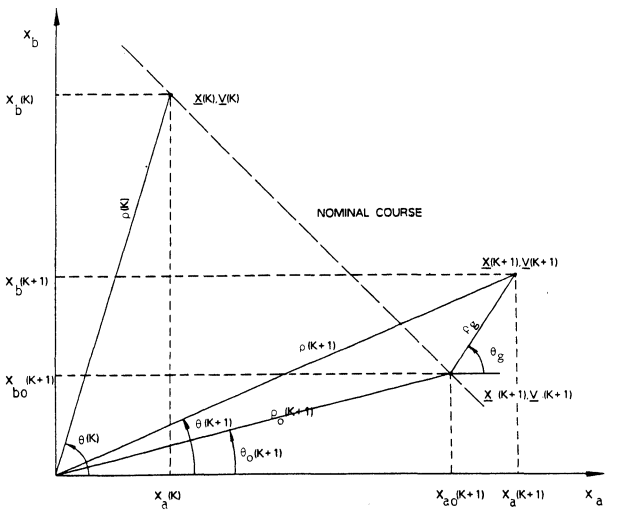


Fig. 2. Effects of acceleration noise on a nominal course of an object.

To make the two filters as similar as possible, the filter equations in scalar form given in Table II have also been implemented on the general purpose machine; input range and bearing data simulating the motion of the object being tracked are generated by subroutine INPUT; these measurements are corrupted by noise produced by a system subroutine.

The equations describing the object trajectory can easily be derived with the help of Fig. 2. At the kth sampling instant, the object is at position $\rho(k)$, $\theta(k)$ with velocity $\dot{\rho}(k)$ and $\dot{\theta}(k)$ which have cartesian representations $\boldsymbol{x}(k)$ and velocity $\boldsymbol{v}(k)$, respectively. Assuming that the velocity is constant in magnitude and direction in the interval $kT \leq t < (k + 1)T$, at the

$(k + 1)$st instant the object would be in position

$$\boldsymbol{x}_0(k+1)=\boldsymbol{x}(k)+T\boldsymbol{v}(k). \tag{8}$$

Consider now the object being subjected to an acceleration noise; for example, an object moving on a conveyor belt is subject to vibration, or an airplane is subject to wind effects. Let the acceleration vector be $\boldsymbol{g}(k)$; then the actual position of the object will be

$$\boldsymbol{x}(k+1)=\boldsymbol{x}_0(k+1)+\frac{1}{2}\boldsymbol{g}T^2. \tag{9}$$

To determine the estimation error, given $\rho(k)$ and $\theta(k)$, we need $\rho(k + 1)$ and $\theta(k + 1)$ for comparison with $\hat{\rho}(k + 1)$ and $\hat{\theta}(k + 1)$ which are the outputs of the Kalman filter. Since (8) and (9) involve vector addition, the polar coordinates for the kth instant are converted into cartesian form and the noise-free position given by (8) is obtained; then, the impact of the acceleration vector $\boldsymbol{g}(k)$ is added and the actual position $\boldsymbol{x}(k + 1)$ is obtained, which is subsequently converted to $\rho(k + 1)$ and $\theta(k + 1)$. These transformations are straightforward with the help of Fig. 2.

Designating as $\theta_v(k)$ and $\rho_v(k)$ the direction and magnitude, respectively, of the velocity $\boldsymbol{v}(k)$ along the nominal course, the noise-free position of the object is given by

$$\begin{aligned} x_{b0}(k+1)&=\rho(k)\cos\theta(k)+T\rho_v\cos\theta_v(k)\\ x_{b0}(k+1)&=\rho(k)\sin\theta(k)+T\rho_v(k)\sin\theta_v(k). \end{aligned} \tag{10}$$

The actual position of the object is given by

$$\begin{aligned} x_a(k+1)&=x_{a0}(k+1)+\frac{T^2}{2}\rho_g(k)\cos\theta_g(k)\\ x_b(k+1)&=x_{b0}(k+1)+\frac{T^2}{2}\rho_g(k)\sin\theta_g(k) \end{aligned} \tag{11}$$

where $\rho_g(k)$ and $\theta_g(k)$ are the magnitude and direction, respectively, of $\boldsymbol{g}(k)$.

The magnitude and direction of the acceleration disturbance are uniformly distributed random variables in the ranges $\{0 - |g|_{\max}\}$ and the range and bearing at the $(k + 1)$st instant are found as

$$\begin{aligned} \rho(k+1)&=\sqrt{x_a^2(k+1)+x_b^2(k+1)}\\ \theta(k+1)&=\tan\frac{x_b(k+1)}{x_a(k+1)}. \end{aligned} \tag{12}$$

Equations (10), (11), and (12) are used in the simulation program to describe the trajectory of the object, and the estimation error is determined by comparing the output of the filter to the values obtained from (12).

V. Program Verification

The filter program is simulated using the XDS/320 Macro Assembler, Linker, and Simulator, which are the software support programs for the TMS320 products [5], [6]. The source program is compiled, linked, and loaded into the XDS/320 simulator; input and output files may be attached to I/O ports to simulate peripherals connected to the processor.

Verification of the filter operation involves 1) implementation of the filter equations on a 32-bit machine using floating point arithmetic, 2) generation of actual object trajectory, with a state vector $\boldsymbol{x}_{32}$, on a 32-bit machine using the model described in Section IV, 3) generation of an estimated trajectory with a state vector $\hat{\boldsymbol{x}}_{32}$, and 4) generation of the estimated state vector $\hat{\boldsymbol{x}}_{16}$ by the filter implemented on the TMS320 using integer arithmetic.

The tracking properties of the reference filter are determined by computing the estimation error $\tilde{\boldsymbol{x}}_{32} = \boldsymbol{x}_{32} - \hat{\boldsymbol{x}}_{32}$ for various object trajectories. For testing purposes, eight different trajectories have been generated, four along straight lines and four containing midcourse changes in direction. The noise variances used are listed in Table I. Four of those trajectories are shown in Figs. 3–6; each trajectory contains 50 sampling points; course changes occur at the 26th sample. The estimation error $\tilde{\boldsymbol{x}}_{32}$ is a measure of the optimal performance of the reference filter; the equations of this filter are identical to those implemented on the TMS320.

The effects of the 16-bit integer arithmetic on the filter performance are determined by comparing the estimated state vector $\boldsymbol{x}_{16}$ from the TMS320 simulator to the state estimate $\hat{\boldsymbol{x}}_{32}$ from the reference filter; this difference forms the quantization error $\tilde{\boldsymbol{x}}_Q = \hat{\boldsymbol{x}}_{32} - \hat{\boldsymbol{x}}_{16}$.

In Figs. 3–6, the actual position is given by the solid line which contains the effects of noise on the object trajectory; the crosses indicate the measured position determined from noisy measurement, and the squares indicate the estimated positions. The measurement system is located at the origin of the coordinate system which is calibrated in unspecified distance units. It should be noted that as the distance of the object from the origin increases, both the measurement and estimation error increase; this is due to the bearing measuring system which is subject to a given noise power. At points close to the origin the effects of the bearing measurement noise are small; as the range increases, the effects become very significant thus affecting the filter accuracy.

The performance of the filter during the entire tracking period can be determined by considering the mean estimation error for every sample point over a large number of different trajectories, or

$$E[\tilde{\boldsymbol{x}}(k)\ \forall k=1,\cdots,50$$

over a set of sample vectors $\tilde{\boldsymbol{x}}(j, k))$, $j = 1, \cdots, 8$, where $\tilde{\boldsymbol{x}}(j, k) = \boldsymbol{x}_{32}(j, k) - \tilde{\boldsymbol{x}}_{16}(j, k)$ is the estimation error at the kth sample of jth trajectory. The means of the range and bearing estimation errors, $\bar{\rho}(j, k)$ and $\bar{\theta}(j, k)$, respectively, for the eight sample trajectories are shown in Fig. 7. From these plots it is evident that the performance of the system is consistent throughout the entire tracking period.

The quantization effect are similarly determined by considering the mean and variance of the range and bearing quantization errors for each sample over the set of sample trajectories. Fig. 8 shows $E[\rho_{32}(k) - \rho_{16}(k)]$ and var $[\rho_{32}(k)$

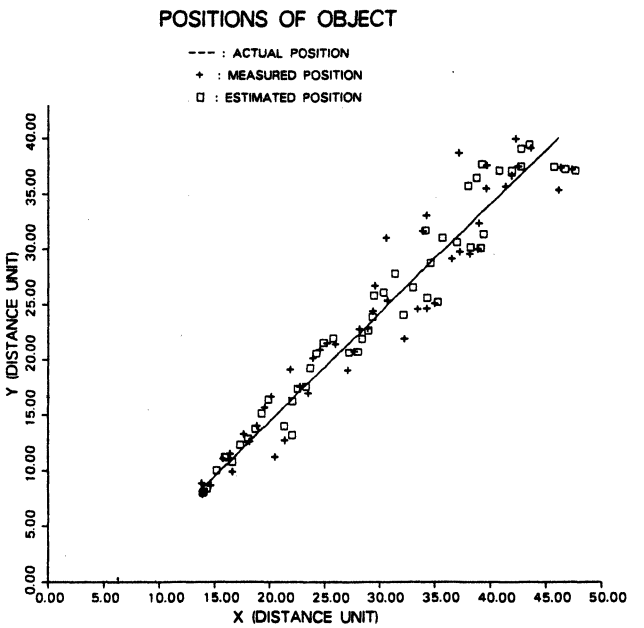


Fig. 3. Filter performance for a nominal straight line trajectory subject to acceleration noise with initial position close to the origin.

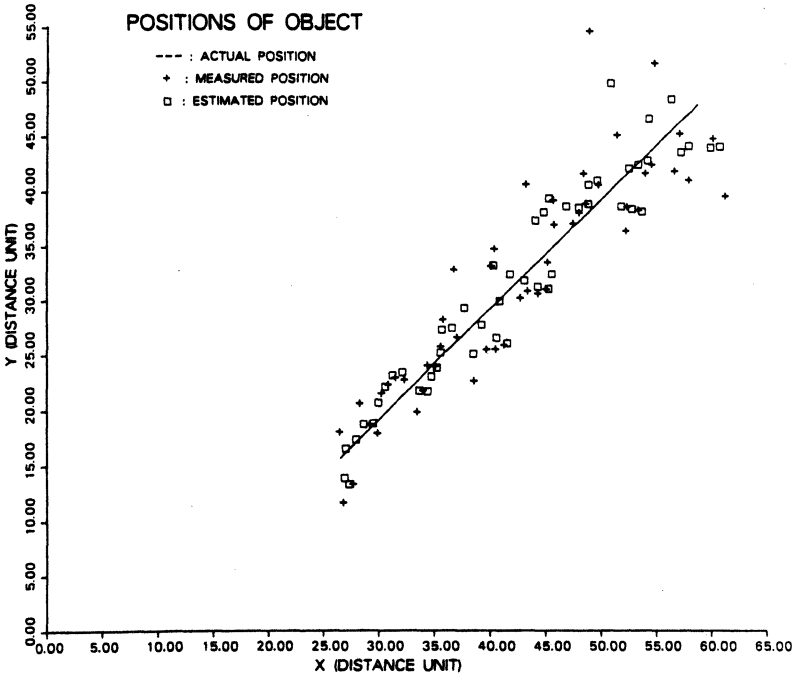


Fig. 4. Filter performance for a nominal straight line trajectory at some distance from the origin and subject to acceleration noise.

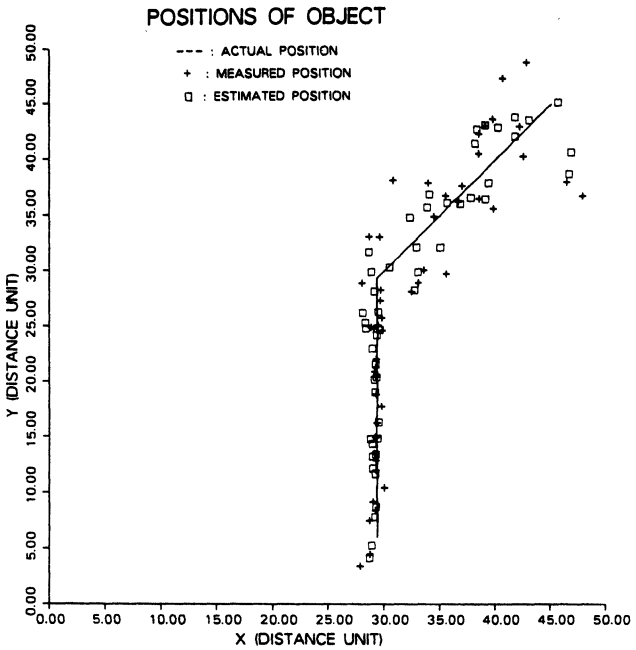


Fig. 5. Filter performance for a trajectory involving a 45° change from the initial course.

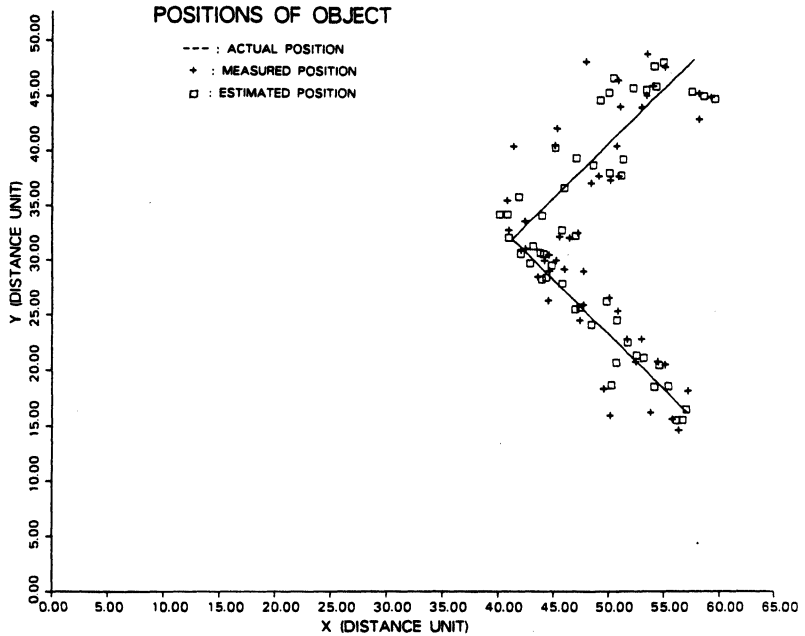


Fig. 6. Filter performance for a trajectory involving 90° change from initial course.

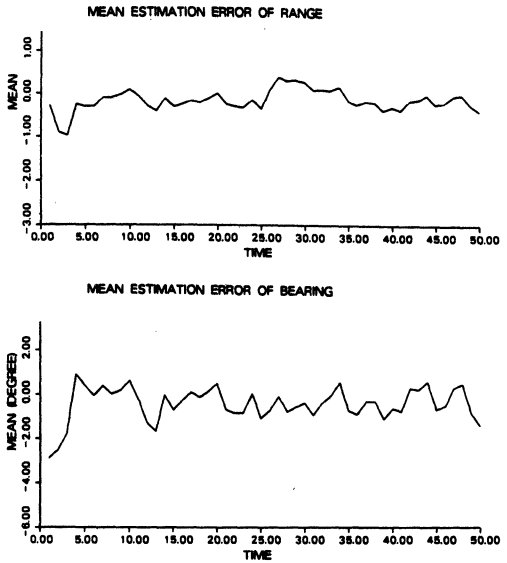


Fig. 7. Mean estimation error for each sampling point over a set of eight sample trajectories.

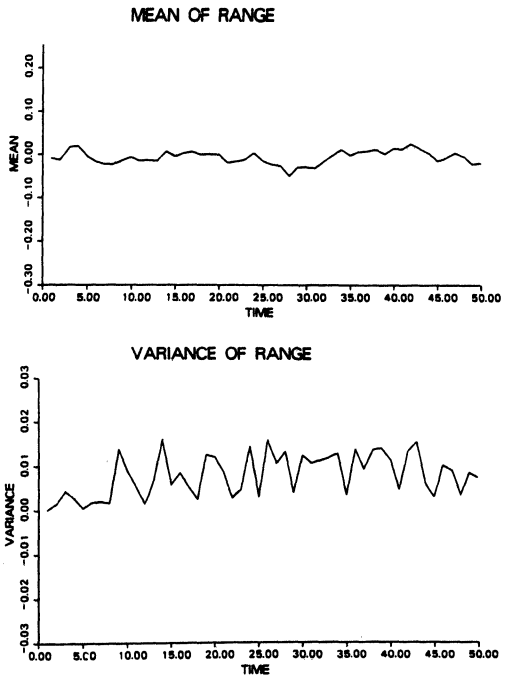


Fig. 8. Quantization effects on range estimates for a sixteen bit implementation compared to a thirty-two bit implementation over a set of eight sample trajectories.

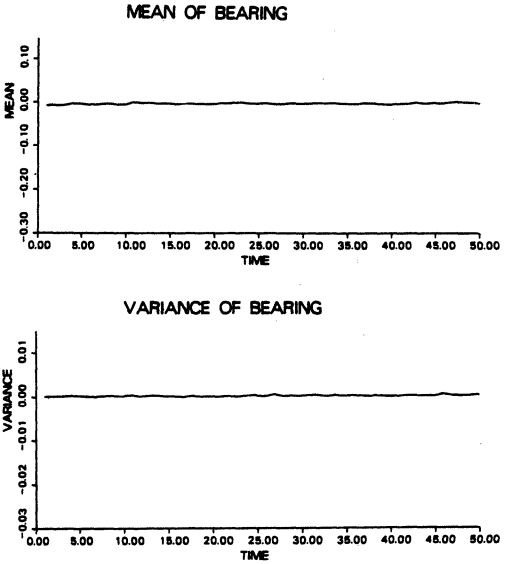


Fig. 9. Quantization effects on bearing estimates for a sixteen bit implementation compared to a thirty-two bit implementation over a set of eight sample trajectories.

$- \rho_{16}(k)] \forall k = 1, \cdots, 50$ over a sample trajectory range $j = 1, \cdots, 8$. Fig. 9 shows similar information about bearing quantization errors. From these figures it is evident that the filter performance is not degraded due to quantization effects.

Another aspect of the filter performance is the maximum system bandwidth under which the filter can operate in real time. The filter implementation presented in this paper requires 1488 instruction cycles to generate one sample estimate. With an instruction cycle time of 0.2 μs, the filter can operate in a system having a maximum sampling frequency of 3.36 kHz or a system bandwidth of 1.68 kHz. This bandwidth is more than sufficient to allow the Kalman filter implemented on the TMS32010 to be used in any application where mechanical motion is involved. The latest signal processing chips having instruction cycle times on the order of 0.1 μs, and can be used in real time systems with bandwidths up to 3.36 kHz. For three-dimensional motion the corresponding systems bandwidths are 1.12 kHz and 2.24 kHz, respectively. These bandwidths are derived by considering that the filter estimates a two-dimensional state vector for each coordinate. The 1488 instruction cycles involve the calculation of two state vector each of dimension 2 as shown in Fig. 1; thus for the three-dimensional case, the total number of instruction cycles required will be approximately equal to two-thirds of these required for the two-dimensional system.

VI. Conclusion

This paper presents a detailed implementation of a tracking Kalman filter on a special purpose digital signal processor. Implementation of such a filter on a real time basis allows for

the design of distributed real time control systems for applications involving multiple sensor tracking of moving objects. Although the design of the filter is based on a specific signal processor, the principles involved, especially in the modeling of the system noise effects, are general enough to be used for implementing the filter on any other processor.

References

[1] F. R. Castella, "An adaptive two-dimensional Kalman tracking filter," *IEEE Trans. Aerosp. Electron. Syst.*, vol. AES-16, pp.822–829, Nov. 1980.

[2] B. Friedland, "Optimum steady-state position and velocity estimation using noisy sampled position data," *IEEE Trans. Aerosp. Electron. Syst.*, vol. AES-9, pp. 906–911, Nov. 1973.

[3] B. D. O. Anderson and J. B. Moore, *Optimal Filtering*. Englewood Cliffs, NJ: Prentice-Hall, 1979.

[4] A. Gelb, *Applied Optimal Estimation*. Cambridge, MA: MIT Press, 1974.

[5] K. G. McDonough, and S. S. Magar, "A single chip microcomputer architecture optimized for signal processing," in *Proc. ICASSP'82*, Paris, May 3–5, 1982.

[6] Texas Instruments Inc., TMS320 User's Guide, 1983.

[7] Texas Instruments Inc., TMS320 Assembly Language Programmer's Guide, 1983.

A STAND-ALONE DIGITAL PROTECTIVE RELAY FOR POWER TRANSFORMERS

Ivi Hermanto, Student Member IEEE, Y.V.V.S. Murty, Member IEEE, M.A. Rahman, Fellow IEEE

Faculty of Engineering & Applied Science
Memorial University of Newfoundland
St. John's, NFLD. A1B 3X5

ABSTRACT

This paper deals with the complete design of a stand-alone prototype digital protective relay for three-phase power transformers. The major emphasis of the paper will be on the detailed description of the hardware and software of the prototype relay. The protection functions implemented include: a percentage differential protection with a second-harmonic restraint for magnetizing inrush and a fifth-harmonic restraint for overexcitation conditions, and a separate protection for high impedance primary and secondary ground faults. The present relay design is tested with the Fourier algorithm and any other relay algorithm can be used by replacing only one subroutine. The relay hardware consists of a data acquisition board and a digital processing board which is based on the TMS320E15 processor. Sample real-time test cases are included in the paper. The results show that the relay never misoperated and correctly identified all the faults that are applied.

Key words:

Digital Relay, Power Transformer, Signal Processing.

INTRODUCTION

The digital protection of electrical power apparatus has been an active area of research for the past twenty years. These research results are being utilized in some of the digital relay designs in recent years [1]. Considerable amount of research has been done on digital protection of power transformers. A number of relay algorithms have been developed [2,3]. The digital protection of power transformers requires complex calculation and logic, hence the use of a digital processor seems natural and attractive.

Transformers are usually protected by means of a differential scheme. Unlike in the bus differential scheme, the transformer differential relay should be designed such that it does not misoperate during magnetizing inrush [4] and overexcitation conditions which fools differential relay operation. Fortunately these non-linear conditions are characterised by a heavy harmonic content in their current signals which can be used to prevent the misoperation of the differential relay.

This paper describes the complete design details of the hardware and software of the prototype digital protective relay for 3-phase power transformers. The following sections of the paper describe the percentage differential and ground fault protection, digital relay hardware and software design and real-time test results.

89 SM 733-7 PWRD A paper recommended and approved by the IEEE Power System Relaying Committee of the IEEE Power Engineering Society for presentation at the IEEE/PES 1989 Summer Meeting, Long Beach, California, July 9 - 14, 1989. Manuscript submitted February 1, 1989; made available for printing May 19, 1989.

PERCENTAGE DIFFERENTIAL AND GROUND FAULT PROTECTION

Principles of differential protection of transformers is well documented [5]. A typical percentage differential characteristic (PDC) which is used for power transformer protection is shown in Figure 1. The threshold C_0 should be selected based on the magnitude of the magnetizing current, and the differential current resulting from the on-load tap-changing during normal loading conditions of the transformer. During overexcitation conditions the threshold C_0 should be increased to C_0' in order to prevent relay misoperation. The slope (C_1^2) of the PDC should be adjusted to make the differential relay insensitive to transformer tap-changing, C.T. saturation and ratio errors during through fault conditions. In addition the relay should also be equipped with a second-harmonic restraint for inrush currents [6,7] and a fifth-harmonic restraint for overexcitation condition.

The sensitivity of the differential protection is somewhat limited for ground faults, due to the magnitude of ground fault impedance. Sensitive protection for ground faults can be obtained by providing a separate primary and secondary ground fault protection.

In order to design the digital relay with the above features, it is required to calculate the differential, through and ground fault currents. From Figure 2, the differential currents are:

$$\left.\begin{aligned} i_{da}(t) &= i_{1a}(t) - [i_{2a}(t) - i_{2c}(t)]n_2/n_1 \\ i_{db}(t) &= i_{1b}(t) - [i_{2b}(t) - i_{2a}(t)]n_2/n_1 \\ i_{dc}(t) &= i_{1c}(t) - [i_{2c}(t) - i_{2b}(t)]n_2/n_1 \end{aligned}\right\} \quad (1)$$

the through currents are:

$$\left.\begin{aligned} i_{ta}(t) &= \{i_{1a}(t) + [i_{2a}(t) - i_{2c}(t)]n_2/n_1\}/2 \\ i_{tb}(t) &= \{i_{1b}(t) + [i_{2b}(t) - i_{2a}(t)]n_2/n_1\}/2 \\ i_{tc}(t) &= \{i_{1c}(t) + [i_{2c}(t) - i_{2b}(t)]n_2/n_1\}/2 \end{aligned}\right\} \quad (2)$$

and the ground fault currents are:

$$\left.\begin{aligned} i_{1gf}(t) &= i_{1a}(t) + i_{1b}(t) + i_{1c}(t) \\ i_{2gf}(t) &= i_{2a}(t) + i_{2b}(t) + i_{2c}(t) + i_{2g}(t) \end{aligned}\right\} \quad (3)$$

where n_2/n_1 is the turns ratio of the transformer.

The main signal processing required for a digital transformer protective relay is the calculation of fundamental and harmonic components of the above current signals. There are many digital relay algorithms available for this purpose [2,3]. In this work an algorithm based on the discrete Fourier trans-

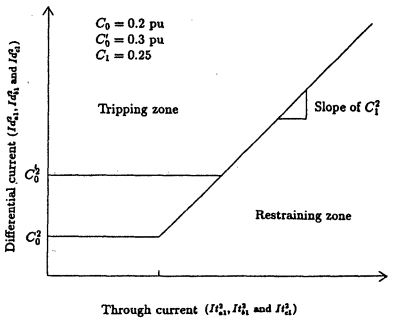


Figure 1. Percentage differential characteristic.

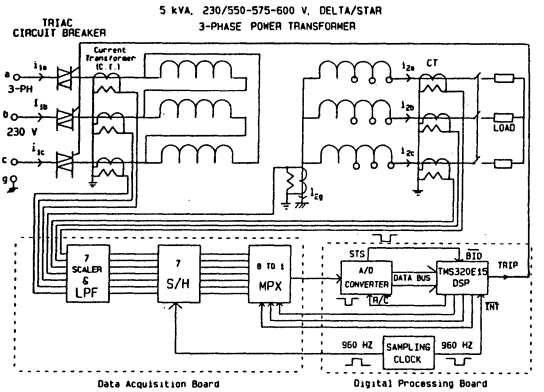


Figure 2. Laboratory test set-up.

form (DFT) is used [8,9]. When the recursive version of the DFT [9] is used, the algorithm worked well in the digital simulation, but when implemented, it showed convergence problems during very low signal magnitudes (overexcitation and high impedance faults) due to quantization errors. Hence, the direct implementation of the DFT is used and it is briefly described here. Consider a sampled signal (sampling period T), $z(k)$, at k_{th} instant (time $t = kT$), the Fourier sine and cosine components of an n_{th} harmonic component are given as follows:

$$\left.\begin{aligned} FS_n(k) &= \frac{2}{N}\sum_{r=0}^{N-1} z(k-r)\sin 2\pi r/N \\ FC_n(k) &= \frac{2}{N}\sum_{r=0}^{N-1} z(k-r)\cos 2\pi r/N \end{aligned}\right\} \quad (4)$$

and the squared magnitude of the n_{th} harmonic component at any time instant is given by:

$$h_n^2 = FS_n^2 + FC_n^2 \quad (5)$$

where h_n is the n_{th} harmonic component and N is the number of samples in one fundamental cycle ($N = 16$ is used in the present design).

In order to provide a secure harmonic restraint function for inrush and overexcitation conditions, the harmonic components of all the three phases (only the differential currents) are combined as follows:

$$ID_n^2 = (Id_{an}^2 + Id_{bn}^2 + Id_{cn}^2), \quad n = 1,2 \text{ and } 5 \quad (6)$$

where Id_{an}, Id_{bn} and Id_{cn} are the n_{th} harmonic components of the three differential currents and ID_n is the n_{th} harmonic component of the combined differential current. The relay software, given in a subsequent section, fully describes the actual implementation of the percentage differential and the ground fault protection.

DIGITAL RELAY HARDWARE DESIGN

The overall laboratory test set-up is shown in Figure 2. A 3-phase transformer with Δ/Y connection is chosen (this transformer and the C.T.s are obtained from the Newfoundland and Labrador Hydro). Seven C.T.s are used, three for the primary currents, three for the secondary currents and one for the secondary ground current. A triac-controlled circuit breaker (C.B.) is used for tripping. The circuit breaker has a built-in optical isolation between power and control circuits which provides complete isolation, and the breaker can operate within one half cycle. The digital relay hardware consists of two different boards, namely, the data acquisition board (DAB) and the digital processing board (DPB). The DAB consists of seven identical circuits each having a scaling circuit, a sixth order Chebyshev anti-aliasing filter (LPF) and a sample-and-hold (S/H) circuit. These seven analog signals are then multiplexed (MPX) and the output of the MPX is connected to the A/D converter which is on the DPB. The DPB consists of a TMS320E15 digital signal processor, an A/D converter, digital I/O ports and a sampling clock generator. The details of the above are given as follows:

Data Acquisition Board

The detailed circuit diagram of the data acquisition board is shown in Figure 3. Since the board has seven identical circuits, only two of them are shown in detail.

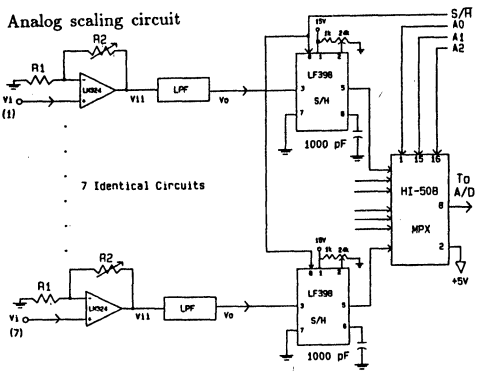


Figure 3. Circuit diagram of the data acquisition board (DAB).

Analog Scaling Circuit: Analog scaling circuit is used to scale the C.T.'s output signal to be compatible with the A/D converter input voltage and also to trim any gain errors between channels caused by either the C.T. shunt resistor mismatch or the gain mismatch of the LPFs. Figure 3 shows the analog scaling circuit which utilizes one operational amplifier (LM324AN) and is configured in non-inverting mode. The gain of the amplifier is given as follows:

$$\frac{V_{ii}}{V_i} = 1 + \frac{R_2}{R_1} \tag{7}$$

The voltage gain of the amplifier can be varied by adjusting R_2.

Anti-Aliasing Filter: If the analog signal cannot be sampled at a rate higher than the Nyquist sampling frequency (twice the highest frequency component in the signal), an error termed as aliasing will occur. Aliasing error occurs due to the fact that the sampled signal may contain low frequency components that are not present in the signal. The aliasing problem can be minimized by using an analog low-pass prefilter; the prefilter should reject all frequency components beyond $f_s/2$, where f_s is the sampling frequency. Since the relay logic utilizes up to fifth-harmonic component of the current signals, the prefilter should pass all frequency components up to fifth-harmonic (300 Hz). Based on the time required for computations and other considerations, a sampling frequency of 960 Hz is chosen. To meet the above-mentioned criteria a sixth-order low-pass Chebyshev filter is designed[10],and it is shown in Figure 4. The filter consists of three cascaded biquadratic sections. Each filter section is a low-pass filter whose general transfer function is:

$$G(s) = \frac{1}{s^2(C_1C_2R^2) + s(2RC_2) + 1}. \tag{8}$$

The values of resistors and capacitors are obtained for the required frequency response and they are given in Figure 4. The frequency response of the filter is shown in Figure 5. The amplitude gain is almost unity from 0 to 360 Hz and the frequency components above 480 Hz are sufficiently attenuated. The filter input and output signals are shown in Figure 6, which indicates a delay of approximately 2 msec between the input and output signals.

Sample-and-Hold Circuit: To achieve simultaneous sampling of all the seven current signals, seven S/Hs (LF398) are used and they are shown in Figure 3. The LF398 holds the analog input signal constant during the A/D conversion to avoid any conversion errors due to rapid fluctuation in the input signal. The hold capacitor is 1000 pF, polystyrene type, which provides fast acquisition time. The dc offset of the S/H can be adjusted by a voltage divider circuit connected to the offset

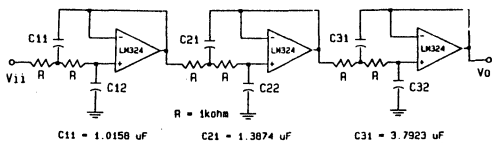


Figure 4. Circuit diagram of the anti-aliasing filter (LPF).

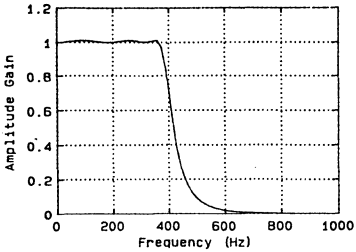


Figure 5. Frequency response of the anti-aliasing filter.

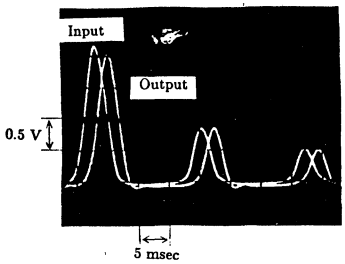


Figure 6. Input and output signals of the LPF: Inrush condition.

pin of the LF398. The acquisition time of the S/H is around 4 μsec. The S/$\overline{H}$ pin of the LF398 is connected to the output of the sampling clock generator which generates a 50% duty cycle square wave of period 1.04166 msec (960 Hz). During the 'high' state of the sampling clock, the S/H is put in the 'sample' mode and when the clock changes to 'low' state, the S/H holds the sampled data. Since the S/$\overline{H}$ pins of all the seven S/Hs are tied to the sampling clock, simultaneous sampling of all the seven signals is achieved. Figure 7 shows the input and output signals of one of the sample-and-hold.

Analog Multiplexer: The connection of the analog multiplexer (MPX) is shown in Figure 3. The HI-508 is an eight channel single-ended CMOS analog multiplexer. It has a fast access time of 250 nsec, fast settling time of 600 nsec and the break-before-make switching feature eliminates the chance of channel corruption. The three digital control lines A_0, A_1, and A_2 are software controlled and they are interfaced directly to the digital output port of the DPB (Fig. 8).

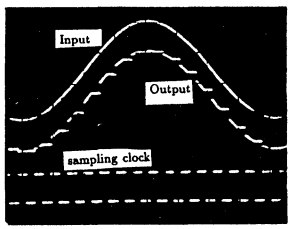


Figure 7. Input and output signals of the sample-and-hold. (Input and output signals are level shifted.)

Digital Processing Board

The detailed circuit diagram of the digital processing board (DPB) is given in Figure 8. The heart of the DPB is a single-chip digital signal processor (TMS320E15 [11]). TMS320 family of signal processors utilizes a modified Harvard architecture for speed and flexibility. Pipelined multiply, accumulate and data shift operations can be executed in two instruction cycles (400 nsec) which makes it very attractive for digital signal processing applications. If a general purpose microprocessor is chosen, it would require a complex hardware with multiple microprocessors to implement the transformer protective relay. The TMS320E15 has an on-chip program memory (EPROM) of 4k words and a data memory (RAM) of 256 words, which is quite sufficient to implement the transformer protective relay; no external memory interfacing is used. The TMS320E15 is interfaced with a digital output port (U_2). D_1 to D_3 bits of U_2 provide the channel address for the analog multiplexer, bit-D_5 is used to clear the interrupt generating latch (U_5), bit-D_6 is used to control the A/D converter operation (R/$\overline{C}$), bit-D_8 is used to send the trip signal to the circuit breaker, and bits D_4 and D_7 are available for any future use. The DPB is also interfaced with a 12-bit A/D converter (U_7). The A/D converter control logic is not fast enough to be directly interfaced with the TMS320, hence it is connected through a tri-state buffer (U_3, U_4). The A/D converter output is in offset binary form and the TMS320 works with 2's complement numbers, hence the bit-D_{12} of the A/D converter is complemented to obtain the output in 2's complement representation. Since the A/D converter has only 12-bit resolution, the remaining four bits (B_1 to B_4 of U_3) can be used as a digital input port for any future expansion. The chip select ($\overline{G}$) for the input port (U_3, U_4) is tied to the $\overline{DEN}$ and the chip select $\overline{DS_1}$ for the output port (U_2) is tied to the $\overline{WE}$. This does not create any conflict because only one input port and one output port are used in the design. The clock generator circuit generates a square wave of frequency 15.36 kHz and it is divided by 16 using a 4-bit counter (U_6) to obtain a square wave of frequency 960 Hz (sampling clock).

DIGITAL RELAY SOFTWARE

The overall flowchart of the relay software is given in Figure 9. The software starts by initializing all the variables. The circuit breaker is closed by sending a logic 'high' through the bit-D_8 of the output port (U_2). At the falling edge of the sampling clock the sampled current signals are held and at the same time the $\overline{INT}$ pin of U_1 goes 'low' which inturn interrupts the processor. One of the MPX channel is selected by sending an appropriate address through the digital output port. Until now the A/D converter is in read mode, and a logic 'low' is sent to the R/$\overline{C}$ pin of the A/D converter to initiate conversion of the selected channel. At the end of conversion (around 22 μsec), the STS line of the A/D converter which is connected to the $\overline{BIO}$ line of the processor goes 'low' and the processor then reads the converted data. When all the seven signals

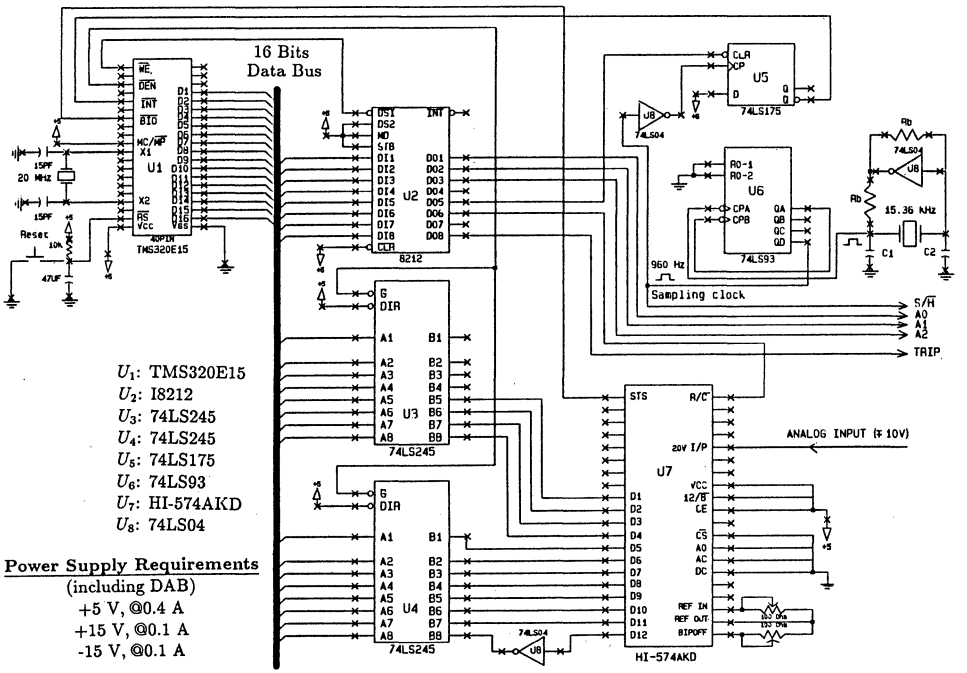


Figure 8. Circuit diagram of the digital processing board (DPB).

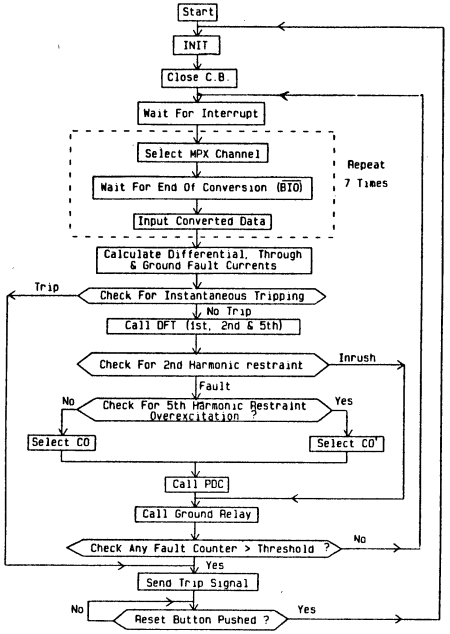


Figure 9. Overall flowchart of the relay software.

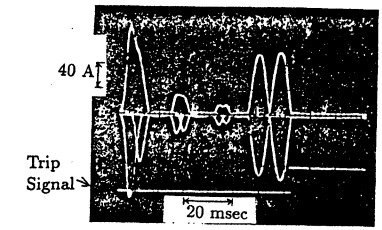
have been read, the differential, through and ground fault currents (eqn.s (1) to (3)) are calculated. Then, the instantaneous threshold is checked as follows: if any one of the differential currents exceeds an instantaneous threshold, C_{it} (C_{it}=10 pu is used), and stays for two consequtive samples then a trip signal is sent, else, the program proceeds. A subroutine which is based on the DFT is then called to compute the fundamental-(Id^2_{a1}, Id^2_{b1} and Id^2_{c1}) second- (Id^2_{a2}, Id^2_{b2} and Id^2_{c2}) and fifth-harmonic (Id^2_{a5}, Id^2_{b5} and Id^2_{c5}) components of the three differential currents, fundamental components (It^2_{a1}, It^2_{b1} and It^2_{c1}) of the three through currents and fundamental (I^2_{1g1}, I^2_{2g1}) and second-harmonic (I^2_{1g2}, I^2_{2g2}) components of each ground fault current. Then the combined harmonic components (ID^2_1, ID^2_2, and ID^2_5) are calculated using eqn. (6).

All the computed harmonic components are stored in the data memory and the second harmonic restraint is checked as follows: if $C_2^2 \times ID_2^2$ exceeds ID_1^2 (C_2 = 0.1767, 17.67% threshold) an inrush condition is declared, then the program branches to the ground relay, else, the overexcitation condition is checked. The presence of a fifth-harmonic component in the differential current, which indicates overexcitation, is checked as follows: if $C_5^2 \times ID_5^2$ exceeds ID_1^2 (C_5 = 0.125, 12.5% threshold), then an overexcitation condition is declared and the upper pick-up value (C_0^2) is selected, else, the lower value (C_0^2) is selected. Then, using the fundamental components of the differential and through currents the PDC (shown in Figure 1) is checked. The PDC is checked three times, once for each phase. If tripping is declared, then the fault counter of that particular phase is incremented, else, it is reset. The program then proceeds to check for the presence of any primary or secondary ground fault. In addition to the level restraint a second-harmonic restraint (8.8% threshold) is also used for both the ground relays. The reason for using the harmonic restraint is that the ground relay is found to operate when the C.T.s saturate during inrush and through fault conditions. During a through fault, a large second and higher order harmonics are present in the ground fault current, whereas during ground fault of either primary or secondary, the second and higher order harmonics are very low. Hence with this harmonic restraint, the ground relay is able to differentiate between a through fault and a ground fault. The sensitivity of the ground relay, then, can be adjusted as desired by varying the pick-up value C_{gt} (C_{gt}=0.1 pu is used). If a ground fault is declared, then the program increments the corresponding fault counter, else, it is reset. Finally the program checks all the fault counters (differential a,b,c phases, and the primary and secondary ground fault). If any one of the fault counter exceeds its threshold value, T_d (T_d=1 for the differential relay and T_d=5 for the ground relay), then a trip signal is sent to the circuit breaker, else, the program returns and waits for the next interrupt. If a trip signal is sent, then the program waits in a loop until the reset button is pressed by the user to restart the relay software. The entire software occupied around 1k words of program memory and 220 words of data memory. The worst case execution time of the software (including data acquisition time of 200 μsec) is around 750 μsec which is well within one sampling period (T=1.04166 msec).

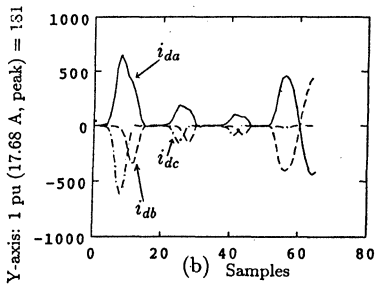
REAL-TIME TEST RESULTS

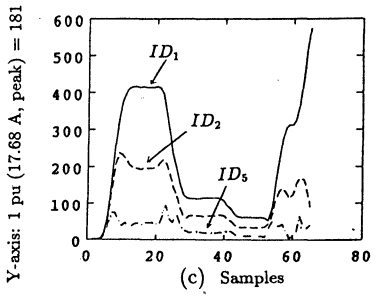
Various types of tests are conducted on the prototype digital relay in the laboratory over a three-month period. The relay correctly identified all the faults that are applied and never misoperated. The TMS320E15 does not have enough memory (RAM) to store a large number of real-time data samples and test results. Hence, for the purpose of plotting the results at each sample, the DPB is disconnected from the relay hardware and the TMS320EVM [12] and AIB [13] hardware is used instead. At each sample the differential current signals, ground fault current signals, calculated harmonic components are stored in the program memory of the TMS32010 (EVM). These results are up-loaded to an IBM-PC and they are shown in the following figures.

Figures 10 to 13 show the performance of the relay during real-time testing. Figure 10 shows the performance of the relay during an inrush condition followed by an internal fault. The relay does not operate during the inrush and it operates within a cycle after the initiation of the fault. Figure 11 shows the performance of the relay during switching on a high impedance internal fault condition. In this case, the tripping decision is delayed due to the presence of a heavy second harmonic content resulting from the inrush currents. Figure 12 gives the performance of the ground relay during a high impedance ground fault. The relay operation time is slightly more than one cycle due to the use of 5 sample delay (T_d=5 for ground relay). Figure 13 gives the performance of the ground relay during a through fault. In this case, the primary ground fault current exceeded its threshold value (C_{gt}=0.1 pu), but the relay did not misoperate due to the second-harmonic restraint.



(a)





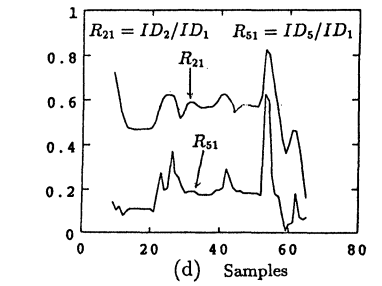
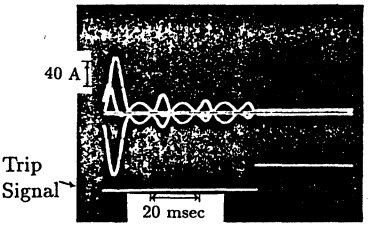
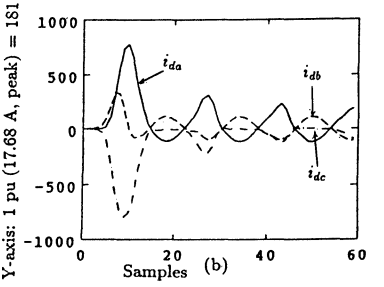


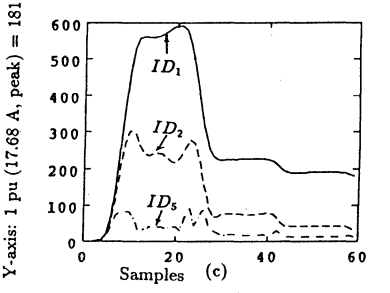
Figure 10. Inrush followed by an internal fault: phase a-b fault on primary side.

(a) Actual differential currents recorded on the oscilloscope
(b) Calculated values of differential currents
(c) Calculated combined harmonic components
(d) Ratios of combined harmonic components



(a)





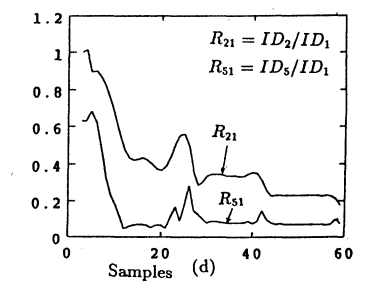


Figure 11. Switching on a high impedance internal fault: phase a-b fault on primary side.

(a) Actual differential currents recorded on the oscilloscope
(b) Calculated values of differential currents
(c) Calculated combined harmonic components
(d) Ratios of combined harmonic components

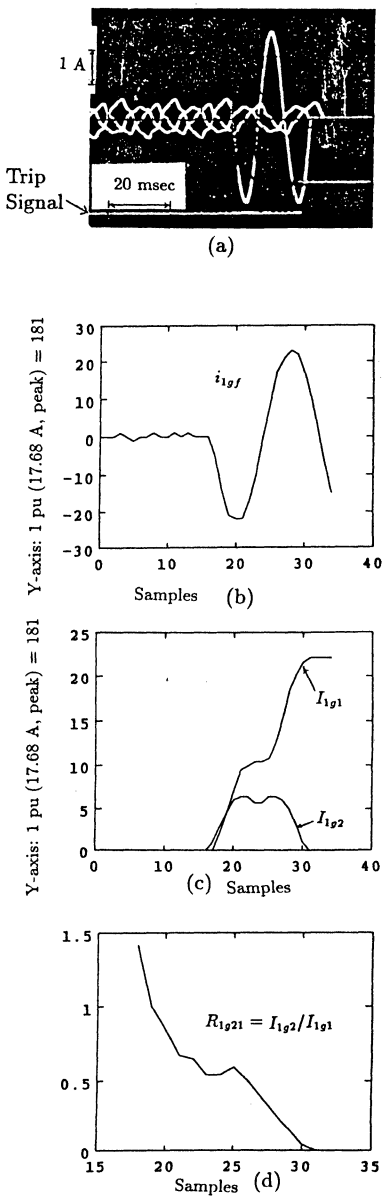


Figure 12. High impedance ground fault: phase a-g on primary.

(a) Actual differential currents recorded on the oscilloscope
(b) Calculated value of primary ground fault current
(c) Calculated harmonic components
(d) Ratio of harmonic components

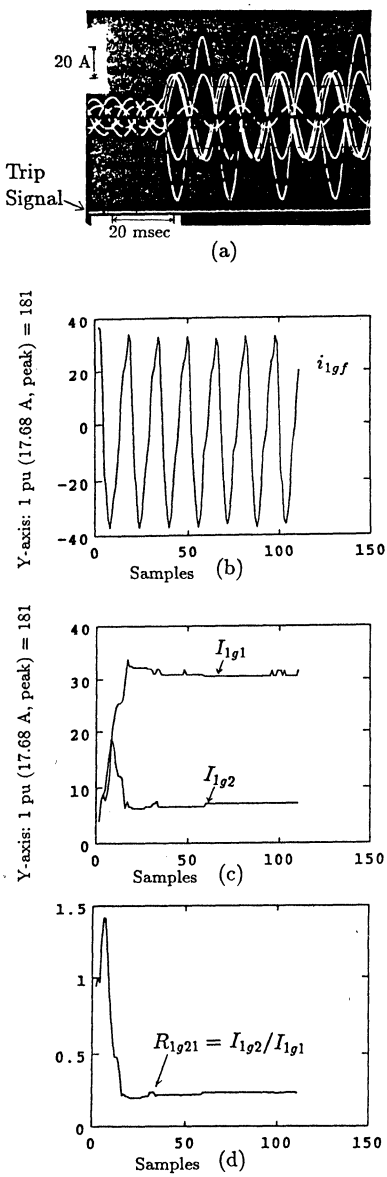


Figure 13. Through fault condition: performance of the primary ground relay.

(a) Actual primary and secondary currents recorded on the oscilloscope
(b) Calculated value of primary ground fault current
(c) Calculated harmonic components
(d) Ratio of harmonic components

Several other tests conducted on the relay in the laboratory include: faults between the primary and secondary windings, faults between transformer tappings, and operation of the transformer at different tap positions during normal loading, overexcitation and through fault conditions. In all these cases the relay performance was as expected.

CONCLUSION

The design of a stand-alone prototype digital protective relay for power transformers is described. The major emphasis of this paper has been the detailed description of the hardware and software development of the relay. The relaying functions implemented include: a percentage differential protection with the second-harmonic restraint for inrush currents and a fifth-harmonic restraint for overexcitation conditions, and primary and secondary ground fault protection. The ground relay is also equipped with a second-harmonic restraint to prevent tripping during inrush and through fault conditions with C.T. saturation.

The detailed circuit diagrams of the relay hardware which is based on the TMS320E15 are included in the paper. The relay had gone through an extensive real-time testing in the laboratory and the results of the sample test cases are reported in the paper. The relay is superior to its electromechanical counterparts in terms of its performance and cost. Currently plans are underway to install the developed relay at one of the Newfoundland and Labrador Hydro substations for in-situ testing and evaluation.

REFERENCES

1. Microprocessor Relays and Protection Systems, *IEEE Tutorial Course Text,* 88EH0269-1-PWR, Feb 1988.
2. M.A. Rahman and B. Jeyasurya, "A state-0f-the-Art Review of Transformer Protection Algorithms", *IEEE Transactions on Power Delivery,* Vol 3, No.2,1988 PP. 534-544
3. Y.V.V.S. Murty and W.J. Smolinski, "A Kalman Filter based Digital Percentage Differential and Ground Fault Relay for a 3-phase Power Transformer", *IEEE PES Winter Meeting,* Paper No. 88 WM 121, 1988, PP. 1-8.
4. M.A.Rahman and A. Gangopadhyay, "Digital Simulation of magnetizing inrush currents in three phase transformers", IEEE Transactions on Power Delivery, Vol. PWRD-1, No. 4, October 1986, PP. 235-242.
5. Guide for Protective Relay Applications to Power Transformers, C37.91, ANSI IEEE Standard, 1985.
6. H.J. Li,"Power Transformer Characteristics and Their Efffect on Protective Relays", *Westinghouse Silent Sentinels Publication RPL 76-1,* April 1984.
7. K. Winick, W. McNutt, "Transformer Magnetizing Inrush Currents and Harmonic Restrained Differential Relays", *General Electric Relaying the News,* RN letter No.86, Nov 26, 1975
8. O.P. Malik, P.K. Dash and G.S. Hope, "Digital Protection of Power Transformer", *IEEE PES Winter Meeting,* Paper No. A76 191-7, New York, Jan 1976, PP. 1-7.
9. J.S. Thorp and A.G. Phadke, "A Microprocessor based Three-phase Transformer Differential Relay", *IEEE Transaction on Power Apparatus and Systems,* Vol. PAS-101, No.2, 1982, PP. 426-432.
10. C.J. Savant, M.S. Roden and G.L. Carpenter, Electronic Circuit Design: An Engineering Approach, *the Benjamin Publishing Company, Inc.,* Menlo Park, California, 1987, PP. 634-644.
11. First-Generation TMS320 User's Guide, Digital Signal Processor Products, *Texas Instruments,* May 1987.
12. TMS32010 Evaluation Module User's Guide, Digital Signal Processor Products, *Texas Instruments,* March 1985.
13. TMS32010 Analog Interface Board User's Guide, Digital Signal Processor Products, *Texas Instruments,* 1984.

AUTHORS

Ivi Hermanto was born in Lampung, Indonesia, on November 27, 1962. He received the B. Engg. degree in electrical engineering from the Memorial University of Newfoundland, Canada in 1987. Currently he is working towards the Master of Engg. degree at Memorial University of Newfoundland. His research work involves the design, development and testing of a prototype digital relay for power transformer protection. He has worked with several companies on a work-term basis during his undergraduate career. His interests are in Computer Applications in Power Systems and Digital Signal Processing.

Yalla, V.V.S. Murty (S'84, M'89) was born in Bendamurlanka, AP, India. He received the Diploma from Andhra Polytechnic in 1976, the B. Tech degree from Jawaharlal Nehru Technological University, Kakinada, in 1981, the M. Tech degree from the Indian Institute of Technology (IIT) Kanpur, in 1983 and the Ph. D. degree from the University of New Brunswick (UNB), Canada in 1988, all in electrical engineering. From February 1983 to December 1983, he was with the Department of Electrical Engineering, IIT, Kanpur working as a Research Engineer in the Microprocessor Lab. He was a Post-Doctoral Fellow at UNB from December 1987 to March 1988. Presently, he is a Post-Doctoral Fellow with the Faculty of Engineering at Memorial University of Newfoundland, Canada, where he is involved in teaching of undergraduate courses in electrical engineering and conducting research in Digital Protection of Power Apparatus.

M. Azizur Rahman (S'66, M'68, SM'73, F'88) was born in Santahar, Bangladesh, on January 9, 1941. He received the B. Sc. Engg. degree in electrical engineering from the Bangladesh University of Engineering and Technology, Dhaka, the M.A.Sc. degree from the University of Toronto, and the Ph. D. degree from the Carleton University, Ottawa, Canada in 1962, 1965, 1968, respectively. Currently, he is a full professor at Memorial University of Newfoundland, Canada. He was the chairman of Newfoundland and Labrador IEEE section. He is the chairman of Electrical Machines Committee, and a member of Industrial Drives committee of the Industry Applications Society, and a member of the Rotating Machinery committee and Induction Machines subcommittee of the Power Engineering Society. He is a Fellow of the Institute of Engineers, Bangladesh. His current interests are in Machines, Power Systems, Digital Protection and Power Electronics.

Discussion

E. A. Baumgartner, (Baumgartner & Associates, Beaumont, TX): The authors point out in the paper that the sensitivity of the digital protective scheme is somewhat limited for ground faults. It would be of interest to know if the digital relay is sensitive enough to detect a turn to turn winding fault in the secondary of a power transformer. A typical percentage differential relay in common use to protect large power transformers is somewhat insensitive to this type of winding failure, especially if the fault current source from the secondary side is low in magnitude.

It would also be of interest to know what type of test equipment will be needed in the field to check out the relay for operational testing after it is installed in one of the Newfoundland and Labrador Hydro substations for in-situ testing and evaluation as proposed by the authors in the paper.

P. K. Dash, J. K. Satpathy, (Regional Engineering College, Rourkela, India): The authors are highly commended for writing an excellent paper on transformer differential protection. The description of the practical details regarding the hardware and software development of the relay is very noteworthy. Rarely such details are found in many relaying application papers. The following points, however, need some clarifications:

1. The digital relay algorithm uses a DFT technique for computing the magnitudes of the restraining and operating signals. It has been shown earlier in the relaying literature that the DFT results in 10 to 15% of error in the calculation of fundamental and harmonic component magnitudes and its accuracy is very much prone to noise magnitudes in the differential current signal.
2. The sampling frequency for this application is 960 Hz, although 720 Hz sampling frequency could have been adequate for this application. Earlier Butterworth filters were used for signal conditioning and it will be interesting to get some comparison regarding delay introduced in case of these two types of filters.
3. The basis for the choice of c2 and c5 for computing the restraining quantities for transformer protection is not very clear. In cases where the inrush current contains substantial components of load current (when the transformer secondary is loaded) these quantities (c2 and c5) need to be altered for providing restrain during inrush and overexcitation conditions.

The prototype building of the relay along with real-time test results is very interesting. It will be interesting to note the effect of a fault during intialization period of this relay. The discusser has also noted with interest the results for the high impedance ground fault on the performance of this relay.

Once again the discusser commends the efforts of the authors for an excellent, well written paper on digital protection.

B. Jeyasurya (Indian Institute of Technology, Bombay): The authors have presented a detailed paper on a digital relay for three-phase power transformers. The test results presented in the paper indicate that the relay operation time is above one cycle. The Fourier algorithm, as implemented by the authors (equation 4) use a data window of one cycle. It is possible to obtain a faster estimate of the fundamental and harmonic components of the input signals using a sub-cycle data window. The sensitiveness of this method to the decaying dc components in the current signals can be minimised by modifying the reference sine/cosine waveforms [A].

The authors use a sixth order low-pass Chebyshev filter to avoid aliasing errors. Figure 2 indicates that this filter has introduced a delay of about 5 msec. A third order Butterworth filter could have provided a maximally flat response with significantly less delay between input and output signals.

For the differential relay, the authors have used a fault counter threshold value, Td = 1. How reliable is the tripping decision based on a single count?

The authors must be commended for a well-written paper. The discusser looks forward to reports of field experience with this digital relay.

Reference

[A]. A. Wiszniewski, "How to Reduce Errors of Distance Fault Locating Algorithms", Trans. IEEE, Vol. PAS-100, No. 12, December 1981, pp. 4815-4820.

A. GANGOPADHYAY (Federal Pioneer Ltd., Toronto, Ontario,Canada): The authors are to be congratulated for real time implementation of a 3-phase digital differential and ground fault relay for power transformers. The algorithm and the basic equations for differential protection are well-known and I do not find any discussion is necessary in that aspect of the paper. However, I would like to place few suggestions to the authors regarding hardware part of the relay.

1. The author has used LM324AN op-amp for analog scaling and low pass filter circuits. LM324 has an input offset voltage of maximum ±7mv. Since the output of scaling and each stage of LPF are cascaded together, the final offset voltage may be predominant and will be reflected at the input of the S/H. Using LM124, which has much lower input offset voltage, or using any other op-amp with lower offset voltage will be an improvement.

2. It would be preferable to have LPF before scaling circuit. There will be transient noises coming from power surge, switching and other electrical disturbances in the input circuit. The voltage gain of the scaling circuit will amplify those noises. It is always better to attenuate a low magnitude noise than an amplified one. Besides, any unwanted noise should be arrested at the very input for a better electronic design.

3. TMS320E15 is a powerful machine for DFT calculation. However, I do not agree with the authors that using other micropocessors will make the design more complex. There are quite a few general purpose micro-controllers available in the market which are much cheaper in price than TMS320E15. With the proper selection of crystal frequency it is possible to compute the algorithms shown in the paper with 16 samples/cycle.

4. The authors could have avoided an external sampling clock. Instead of, a precise sampling pulse could have been generated from the microprocessor by software control in either of the following manners.

a) Software Timing Loop, if time is still available after computation.

b) Internal timer of microprocessor.

The advantage is that the sampling interval is fully under control of the designer by above method. The same relay could be used in 50Hz system by mere changing the software, to modify the sampling time at no extra cost of additional hardware.

5. It is shown in the Fig. 9 of the paper that the relay waits for the Reset button to be pressed after issuing a trip signal. This is not desired from practical viewpoint. What will happen if the circuit breaker fails to trip or the relay is protecting a transformer in an unmanned substation? The relay should be self reset in this case.

6. An additional feature can be supplemented to the design by adding a LED or LCD display. After tripping the breaker at a fault, the relay can calculate the RMS value of the fault current from one cycle information that the relay has already collected. It can keep the fault magnitude in it's memory and can be displayed to the user at any time. This will give an idea to the user about the extent of the fault current, i.e. whether it is an interturn fault or it involves quite a few number of turns.

7. Nothing much has been mentioned about the design of the power supply circuit. If it is intended to be taken from the station battery supply, two dc to dc conversion circuits will be needed. If it is to be taken from UPS, proper consideration should be given to design two different voltage levels. If resolution for harmonic computation can be sacrificed to certain extent, there are standard techniques

which can be employed to handle the bipolar signals by unipolar A/D converters.

8. One would like to see an analog backup circuit in the event of failure of main processor or it's other electronic accessories. Also, external noise suppression and overvoltage protection due to open CT secondaries will be required to meet several ANSI and IEEE standards before the relay can be used for practical purposes.

Manuscript received July 20, 1989.

IVI HERMANTO, YALLA, V.V.S. MURTY and M.A. RAHMAN: The authors thank the discussers for their interest and thoughtful comments on our paper.

Our response to the questions raised by Mr. Baumgartner is as follows. The first question deals with the performance of the relay for a turn-to-turn fault on the secondary winding. Since the test transformer did not have individual turn taps brought out to create a turn-to-turn fault, it was not possible to test for this condition. However, the tests conducted on the transformer include a short circuit between the taps on the secondary side. Here, an additional test case which represents a short circuit between the 575 V and 600 V taps on the phase-"a" secondary is provided in Figure F1. This represents a case when 4.166% of the phase-"a" secondary winding is short circuited. The relay successfully operated within one cycle. It can be seen from Figure F1 (c) that the value of ID1 reaches about 0.95 pu. With the threshold (CO) set at 0.2 pu, this shows that the relay will operate even if a smaller percentage of the winding, perhaps less than 1%, is involved in the fault.

The second question deals with the type of test equipment that will be required for operational testing in the field. An IBM PC will be required in the substation which will be connected to the relay through a serial link. The PC will continuously monitor the relay operation and acquire the data during various operating conditions. In order to completely isolate the digital relay from the existing relay equipment of the transformer, additional current transformers will be used.

Our reply to the questions raised by Professor Dash and Mr. Satpathy is as follows. The first question deals with the choice of the relay algorithm. As mentioned in the paper, any relay algorithm can be used by simply modifying the appropriate subroutine. There are various algorithms available with their own advantages and disadvantages, and the user can select the algorithm suitable for any particular application. Since the implementation of the discrete Fourier transform (DFT) algorithm is considered to be computationally more complex than most of the other algorithms, it will be an easy task to replace it with any other algorithm.

The second question deals with the choice of the sampling frequency and the antialiasing filter design. The execution time of the relay software is around 750 microseconds and the use of 960 Hz sampling frequency leaves about 290 microseconds for any other tasks. Ofcourse, one can use 720 Hz sampling frequency but it requires a very good antialiasing filter whose frequency response has a very fast roll-off near its cut-off frequency. The authors used Butterworth filters in an earlier design and the delay introduced by the Chebyshev filter is nearly the same as the Butterworth filter if both filters are of the same order and have the same cut-off frequency.

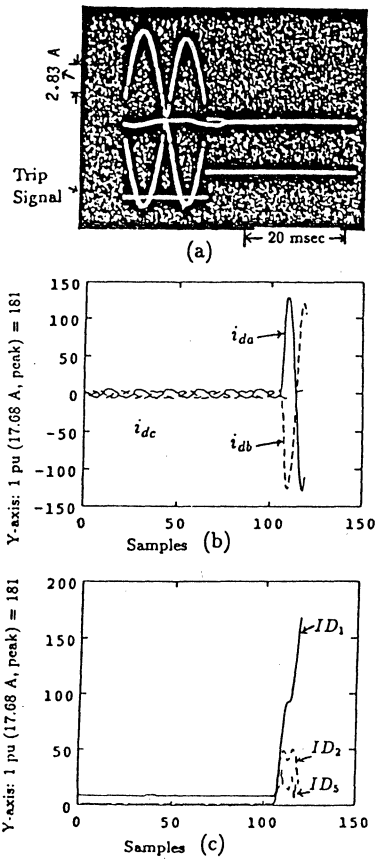


Figure F1. Internal fault condition: fault between 575-600 V taps on phase-a.

a) Experimental differential currents, b) Calculated differential currents, c) Calculated harmonic components

The third question deals with the choice of relay settings C2 and C5. The values selected are for the test transformer and these can be changed to other values depending on the application. The initialization time for the relay software is around 250 microsecond, and hence it does not introduce any major delay.

Our response to the questions of Mr. Gangopadhyay is as follows:

1. We agree with the discusser that the operational amplifier used in the design has a high offset voltage. However, the offset voltage is not a major concern, since the DFT algorithm effectively filters out any dc offset present.

2. The gain of the analog scaling circuit used in the design is very close to unity. Hence, it does not

amplify the noise. It is mainly used to trim any gain errors between channels. Also, the analog scaling circuit provides very high input impedance.

3. Considering the equation (4) of the paper, each harmonic calculation requires 32 multiplications. A total of 16 harmonic calculations (fundamental, second and fifth harmonic components of three differential currents, fundamental components and second harmonic components of two ground fault currents) were performed in each sampling interval. This gives a total of 512 multiplications plus several additions and other operations are required. In the opinion of the authors it is difficult to perform these calculations on any presently available low cost microcontroller within a reasonable sampling interval.

4. The software generation of sampling interval is not suitable for this type of application. The authors agree that the sampling interval could have been generated by a hardware timer which would give flexibility in changing the sampling clock for 50/60 Hz systems. In fact one could use TMS 320E17 processor which is software compatible with TMS320E15 and also has an internal 16-bit timer.

5. The authors agree that in certain applications the relay should be of self reset type. The software can be easily modified to achieve this.

6-8. The authors agree that more work needs to be done in developing the user interface, power supply and other related hardwares compatible to ANSI/IEEE standards.

Our response to the questions raised by Dr. B. Jeyasurya is as follows:

The laboratory test shows that the relay operating time was within one cycle of 60 Hz for various internal faults except in the case of switching on a high impedance faults. In the case of high impedance faults the relay operating time was above one cycle due to the presence of strong second harmonic component. One cycle data window of the Fourier algorithm was used in our design. However, any other algorithms can be easily implemented.

From Figure 6, it is quite clear that the delay between the input and output signals of the anti-aliasing filter is only 2 milliseconds not 5 milliseconds as mentioned by the discusser. Both Chebyshev and Butterworth filters have nearly the same delay for a given order.

The authors agree with this discusser that a tripping decision based on a single fault counter threshold is not reliable. However, this fault counter can be increased easily to any other safe threshold value.

Manuscript received November 13, 1989.

A Real-Time Digital Simulation of Synchronous Machines: Stability Considerations and Implementation

JONATHAN P. PRATT AND SHELDON GRUBER, SENIOR MEMBER, IEEE

Abstract—**The study of the transient behavior of a large power system has been difficult and time consuming even on mainframe computers. One way to obtain real-time studies is to configure digital simulation modules in a parallel processing network that corresponds to the physical system. The focus of this work is on the creation of a generator module that is compatible with such a digital simulation network. To approach operation in real time, a fast and accurate state equation integrator is required. Investigation has revealed that the load imposed on the simulated generator plays a major role in the stability of the integration routines. The linearized stability limits of forward difference, modified Euler, fourth-order Runge-Kutta and Adams-Bashforth-Moulton integration methods were calculated for an impedance terminated generator. These were found to agree closely with the corresponding experimentally determined nonlinear limits. The TMS32010 digital signal processor was chosen as the heart of the generator simulator module, and fixed-point arithmetic routines were developed to make it a high-speed state equation integrator. Operation in real time was achieved for an infinite bus-type termination, but an impedance load led to a somewhat slower simulation.**

I. INTRODUCTION

ELECTRIC POWER systems are particularly well suited for simulation, as the information gleaned in their study would usually be sought in advance of their construction or expansion. The expense of power system operation as well as customer expectations require that reliability and efficiency be assured under a variety of stressful conditions. Studies on existing equipment are necessarily limited by the need to maintain service, thus simulation becomes a realistic tool [1].

A primary concern in a power system's assembly is that it will maintain stable operation for a reasonably wide range of operating conditions. Two important types of instability receive most of the attention in literature: dynamic and transient [2]. The variables of power system stability are rotor speeds are relative positions, and generator loads. Dynamic stability is concerned with the usual small-speed variations within a system which can become oscillatory and growing in nature. Transient instability refers to the system response to a major fault. In either case loss of synchronism may occur, an event that tends to break up a system. Such considerations play an important role in power system planning, even more in the last 20 years than before, due to the "very extensive interconnection of power systems with greater dependence on firm power flow over ties" [2]–[4].

Manuscript received September 11, 1985; revised January 20, 1987.

The authors are with the Electrical Engineering and Applied Physics Department, Case Institute of Technology, Case Western Reserve University, Cleveland, OH 44106.

IEEE Log Number 8716649.

Any study not using actual equipment [5] requires a mathematical model, and much work has been done to find the least complicated models for power system components that will still provide accurate results [2]–[7]. Once a model is derived, implementing the simulation is a matter of choosing between several methods, each with its own advantages and disadvantages. Use of a mainframe computer has been the dominant method. Flexibility and low cost per user make mainframes attractive. Unfortunately, if the power system of interest contains many elements—generators, exciters, loads, etc.—the serial solution of the system equations is cumbersome and slow. Speed is increased at the expense of the model, and beyond a certain point the model becomes too simple to be useful. Alternatively, a parallel solution is possible. This exploits the system parallelism so that many smaller, inexpensive units take the place of one fast computer. The choices for this method are analog [8], digital, or a combination of the two. A good discussion of the difficulties of an analog simulation is found in [9]. Among these are achieving correct scaling and avoiding a lack of flexibility. The usefulness of the analog method lies in its ability operate in real time. Until recently, reasonably priced digital hardware with sufficient speed did not exist. The analog–digital hybrid approach of [9] is an example of a useful intermediate step. Reasonable real-time results were obtained with a MC68000 microprocessor interfaced with appropriate conversion hardware to an analog bus.

The simulation method chosen here parallels the construction of the power system which is divided into blocks of turbines, governors, generators, exciters, and loads. The scheme is one in which all power system components are replaced by digital hardware modules. These modules employ the models of the components they replace to create an accurate simulation. During each computation cycle the generator modules calculate and supply the rotating frame currents to a set of hardware matrix multipliers [10]. Following the relation $V = ZI$ the voltages of the power network are obtained. These in turn are used by the generator modules to compute the currents of the next cycle. Additionally, the exciter modules employ both the voltages and currents of their respective generators to apply appropriate regulation through the field voltages. Turbines, having time constants longer than those of interest, are simply represented as constant mechanical powers within the generator modules. It should be noted that load representation by constant impedance is certainly not

ideal [1], [11], [12]. Although variation of line frequency is usually small enough to render corresponding impedance variations negligible, the same does not hold true for line voltage changes. In Kent *et al.* [1] typical loads are reported to be distributed so that one-half are constant impedance and one-half constant MVA. The latter is obviously not compatible with the matrix multiplication scheme. The influence of load representation on stability studies is not clear, but it can be significant [12]. In spite of the necessary simplifications, the speed and flexibility of the completed simulation network will undoubtedly render it a useful tool in the analysis of power system behavior. The machine parameters used in the remainder of this paper are typical of realistic machines. Those quantities used by the generator simulation program are found in Anderson and Fouad [6, ch. 4].

The next section is devoted to the stability of the required integration routines, as this was found to be a significant limitation in the quest for real-time operation. In particular, the stability limits of forward difference, modified Euler, fourth-order Runge–Kutta, and fourth-order Adams–Bashforth–Moulton predictor–corrector (ABM-4) integration methods were investigated. Choice of hardware, program development, and results are discussed in Section III. Module simulation data are compared to mainframe results. Suggested improvements are also included. Appendixes I and II refer to interface hardware added to the TMS32010EVM board to make it function within the simulation network, and design for simplified generator modules employing the TMS32010 processor with the host processor interface that allows access to the new modules.

TABLE I
SUMMARY OF GENERATOR STATE EQUATIONS

$$\underline{L_d} = \begin{vmatrix} L_d & kM_F & kM_D \\ kM_F & L_F & M_R \\ kM_D & M_R & L_D \end{vmatrix} \qquad \underline{L_q} = \begin{vmatrix} L_q & kM_Q \\ kM_Q & L_q \end{vmatrix}$$

$$V_d = \begin{vmatrix} r\cdot i_d + \omega\cdot(L_q\cdot i_q + kM_Q\cdot i_Q) + v_d \\ r_F\cdot i_F \\ r_D\cdot i_D \end{vmatrix}$$

$$V_q = \begin{vmatrix} -\omega\cdot(L_d\cdot i_d + kM_D\cdot(i_F + i_D)) + r\cdot i_q + v_q \\ r_Q\cdot i_Q \end{vmatrix}$$

$$I_d = \frac{d}{dt}\begin{vmatrix} i_d \\ i_F \\ i_D \end{vmatrix} = -\underline{L_d}^{-1}\cdot V_d$$

$$I_q = \frac{d}{dt}\begin{vmatrix} i_q \\ i_Q \end{vmatrix} = -\underline{L_q}^{-1}\cdot V_q$$

$$Pe = r\cdot(i_d^2 + i_q^2) + r_D\cdot i_D^2 + r_Q\cdot i_Q^2 + v_d\cdot i_d + v_q\cdot i_q$$

$$\frac{d\omega}{dt} = (Pm - Pe)/(6\cdot H\cdot\omega_R\cdot\omega)$$

$$\frac{d\delta}{dt} = \omega - 1$$

II. Integrator Stability

A. General Considerations

Table I is a summary of the seven state equations that model the behavior of a generator. The inherent nonlinearity of these equations makes step-by-step numerical integration the only viable method of obtaining a solution apart from analog modeling. The transients to be studied by the simulation are large enough to render continuous time linear models inadequate. In this section a brief survey is taken of some commonly used integration methods.

The foremost consideration in the simulation is a faithful reproduction of generator's nonlinear behavior when it is connected to a load. The load may not be linear and consists of the transmission network and active sources which are also nonlinear. That this simulation proceed in real time is a requirement that makes compromises in the design necessary.

The transient behavior of the system will not be adequately modeled if the sampling rate is lower than the Nyquist rate. An estimate of the latter can be obtained by examination of the eigenvalues of the linearized system. The magnitude of the largest eigenvalue is typically that due to currents injected into the rotor circuit needed to balance the constant stator flux and is approximately 60 Hz [6]. Thus it is reasonable to use a time interval of less than 8 ms.

This is the upper limit in time step from the point of view of producing a model generator which must interact faithfully with the rest of the power system. There remains the question of stability of the integration method using this time step. A recent paper [13] presented a design technique which allows for a simulation step size to be chosen independently of the eigenvalues while maintaining stability of the integration. Unfortunately, application of the results of that paper to this problem is difficult in that the system at hand is nonlinear and would require constant examination for stability region violation as the integration proceeds and the instantaneous eigenvalues change. One additional factor in not utilizing the stability region approach is that the generator simulator must operate into unknown and nonlinear loads which form part of its system of equations in an indirect manner. That is, the model requires the terminal voltage of the generator from which it calculates the current vector. These currents combined with those of the other generators in the simulated power system produce the terminal voltage for calculating the next time step. Thus the stability region idea is difficult to apply because of the parallel operation of many simulators.

In keeping with the ultimate goal of this project, all effort was to minimize the computation time per step, maintain reasonable accuracy, and insure stability.

B. Parameter Representation

The parameters used in the remainder of the paper are typical of realistic machines. Those quantities used are to be found in Anderson and Fouad [6, ch. 4]. To improve machine speed in the module, these parameters were rounded off to 16 bits and the resulting numbers were deemed "exact." This is contrast to the parameters that are derived from the above values, namely, the inverse inductance matrix whose elements

are only as accurate as the number of bits used to represent them. The time increment, T, of the module state equation integration is also treated as a 16-bit "exact" constant. Here it should be noted that time in the equations shown in Table I is normalized as are all the parameters in these equations. The normalization entails multiplication of true time by the nominal radian frequency, in this case 377 rad/s. This being the case, T is also measured in radians at 60 Hz.

C. Integration Methods

If the equations involved were linear, simple backwards difference or trapezoidal integration methods could be applied. The advantages of these are discussed in the next section. But for a nonlinear system, the equivalent integration routines are forward difference and modified Euler. Assuming state equations of the form

$$dY/dt = f(Y)$$

where Y is the vector of variables, then the methods compare as follows:

Backward Difference
$$Y_{n+1} = Y_n + T * f(Y_{n+1})$$

Forward Difference
$$Y_{n+1} = Y_n + T * f(Y_n) \tag{2.1}$$

Trapezoidal
$$Y_{n+1} = Y_n + T * (f(Y_n) + f(Y_{n+1}))$$

Modified Euler
$$Y_{n+1}, p = Y_n + T/2 * f(Y_n)$$
$$Y_{n+1} = Y_n + T/2 * (f(Y_n) + f(Y_{n+1}, p)). \tag{2.2}$$

Forward difference integration may be termed first order, and it has a global error term corresponding on the order of T, $0(T)$. Modified Euler is second order and has a global error of $0(T * T)$. The price of the improved accuracy is an additional evaluation of the derivative functions. Model error and integrator error should be approximately equal to obtain maximum efficiency. One of the most commonly used integration routines is fourth-order Runge–Kutta. Its global error is $0(T ** 4)$, and in this paper is was used as a benchmark to provide the "definitive" answer. The particular version used was taken from [14]:

$$Y_{n+1} = Y_n + (k_1 + 2k_2 + 2k_3 + k_4)/6$$
$$k_1 = T * \cdot f(Y_n)$$
$$k_2 = T * f(Y_n + k_1/2)$$
$$k_3 = T * f(Y_n + k_2/2)$$
$$k_4 = T * f(Y_n + k_3). \tag{2.3}$$

The independent variable, time, is left out because in the generator model the state equations have no explicit time dependence. Note that four derivative calculations are required per cycle. As evaluating the derivatives is the most time consuming operation, it was worth investigating another integration method that yields similar accuracy for only two evaluations per cycle. In particular, the fourth order Adams–Bashforth–Moulton predictor–corrector was explored. Known as a multistep method, ABM-4 requires that the past three derivatives be saved for use in a weighted average. Also from [14]

Predictor:

$$Y_{n+1} = Y_n + T * (55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3})/24$$

Corrector:

$$Y_{n+1} = Y_n + T * (9f_{n+1} + 19f_n - 5f_{n-1} + f_{n-2})/24. \tag{2.4}$$

D. Load Considerations

With the state equations set up on terms of currents, the rotating frame voltages, V_d and V_q, must be supplied to the generator integration routines. For a mainframe simulation, there are two obvious possibilities. The first is to have the generator supply power to an infinite bus. The second is to have the generator connected to a constant impedance. The impedance can be chosen to maintain a steady state at the initial angle—that is, to make all of the derivatives in the state equations vanish. In either case, infinite bus or impedance, the voltage calculations are placed directly within the derivative evaluation routines. When the benchmark integration was performed on the infinite bus configuration, anticipated results were obtained. However, the impedance configuration proved to be much less stable, and could not be integrated except at much higher sample rates. From the system standpoint, this is a cause of considerable concern. The infinite bus approximation is only valid when there is a great deal of generating capacity on line, thus it is expected that the impedance results could reflect the behavior of a small multi-generator system. The desired integration sample rate determines the minimum number of generators that create a stable solution. Because a higher sample rate jeopardizes the ability of the simulation to run in real time, the stability phenomenon was explored in detail. It was discovered that there are fixed limits to the sample rate, under which the integration variables grow exponentially and invalidate the run. Even in cases where no initial perturbation was applied, roundoff changes in the least significant digits were enough to start the explosion. This was evidence that the nonlinear aspects of the state equations are not responsible, and indicated that it might be possible to apply linear analysis to the problem. To support this assertion, the moment of inertia of the generator was made infinite, effectively removing the mechanical state equations. The unstable behavior persisted. To find the cause of instability from a qualitative standpoint, the linear state equation matrices for infinite bus and impedance configurations were compared. The state equation may be written as

$$dI/dt = f(I) = M * I. \tag{2.5}$$

The most significant difference between the two generator loads is that the self-feedback terms for i_d and i_q in the matrix,

M, become three orders of magnitude layer in the impedance load case. Also, these new coefficients are of the same order as the largest elements in their respective rows.

E. Discussion of Stability of Linear Systems

A brief discussion of stability considerations for linear systems should help in understanding the behavior of integration methods in the nonlinear case at hand.

Most integrations of linear state equations based on a discrete-time system can be described by a mapping from the complex s-plane of the Laplace transform to the complex z-plane of the z-transform. To demonstrate this, consider the continuous time system $dY/dt = M * I$. The Laplace transform of this results in the characteristic equation

$$sI - M = 0$$

where I is the identity matrix of the same dimension as M.

All of the single step integration methods of the previous section produce a sequence of the form $Y_{n+1} = N * Y_n$, which is a discrete time description of the original system. Applying the z-transform results in the characteristic equation

$$zI - N = 0.$$

Because the matrix N is a function of M, an explicit mapping exists between the z and s planes. In the case of multistep integration methods, the predictor sequence includes more previous value terms, e.g., Y_{n-1}, Y_{n-2}, Y_{n-3}, $\cdots$. Consequently there are higher powers of z in the z-transform relation and hence in the mapping. The s and z domains have the advantage of easily predicting the stability of systems. A stable continuous-time system has all its roots or eigenvalues in the left-half of the s-plane, while a stable discrete-time system has all its roots inside the unit circle centered at the origin of the z-plane [16]. A system with roots outside of these boundaries is unstable. It is now apparent that a necessary feature of an integration method is its ability to map a stable continuous-time system into a stable discrete-time system. That is, roots that were in the left half of the s-plane must map into the z-plane unit circle. The backward difference maps the left half of the s-plane into a subcircle within the z-plane unit circle and the trapezoidal method maps the left-half plane into the interior of the unit circle. Both satisfy the stable-in stable-out requirement. Unfortunately neither can be used for nonlinear equations, except with approximations as in [15]. This is because the derivative at the $n + 1$ step is now known exactly. Practical integration routines reply on predictions of the $n + 1$ derivative and function. In general, a smaller step size helps to squeeze the z-plane roots into the unit circle so that stability can be achieved. As will be shown, parameters of the function being integrated also play a role in determining stability.

F. Some Results

In Fig. 1 the linear continuous-time stability of the generator equations is demonstrated by a presentation of the system roots closest to the border of instability. As is done throughout, the system parameters varied are generator angle and generator open circuit voltage. Note that the roots move farther into stable territory as the angle increases. Beginning with forward difference, the linear stability of last section's four integration methods is explored in detail. Combining (2.1) and (2.5) yields $Y_{n+1} = Y_n + T * M * Y_n$, which, after application of the z-transform and use of $Z(Y_n)J = JY_n = I$. Y_n reveals the characteristic equation

$$(z-1)I - T * M = 0.$$

A computer program was written to generate and solve the characteristic polynomial in z. In this case there are terms up to the fifth power of z. For given values of the angle and open circuit voltage the stability limit was calculated in terms of T. Thus in Fig. 2, integrator stability is achieved for values of T below the appropriate V_{oc} curve. Limits have been calculated for five values of V_{oc}. It is seen that at small angles an extremely large sample rate is necessary to avoid instability. Next the modified Euler method of integration is considered. Unlike forward difference, where it was possible to calculate directly the value of T that puts the z-plane roots on the unit circle stability boundary, modified Euler equations (and the other methods discussed) necessitate the use of a half-interval search method. That is, a high and a low T are selected initially, and the calculated stability of the average kt determines the new search interval. In this fashion it is possible to converge on the correct limit to arbitrary accuracy. The stability limits of the modified Euler system are graphed in Fig. 3. Compared to forward difference this system is more stable at every point, and the low angle problem is no longer as severe. Fourth-order Runge–Kutta concludes the single step methods. Fig. 4 reveals that the stability limits of the Runge–Kutta method are the most favorable of all. The lines of constant open-circuit voltage V_{oc}, level out at small angles and prevent the required sample rate from becoming excessively large. Linearizing the ABM-4 integration method is somewhat more complicated, but the idea is the same. The fact that this polynomial is of 20th degree and has roots of greatly different magnitudes makes it difficult to compute the stability limits as a smooth curve, even with double precision arithmetic. Fig. 5 shows the results of this effort. The limit lines, though somewhat ragged, are very similar to those obtained with the single step methods. In this instance it is likely that the Jury test [16] would have been a better method of determining stability than calculating the roots directly. To test the validity of the calculated stability limits, values were determined experimentally for several cases. This was accomplished by applying a small perturbation to the various integration routines, and observing the response as a function of the time increment. Since the generator equations were set up for steady-state operation, exponential growth of the current variables was taken to imply an unstable system. The stability limits are quite distinctive, and as Fig. 6 shows, they are in good agreement with the calculated values. The irregular notches in the calculated limits are a product of the root-finding algorithm and do not appear in the tested limits. Now it is possible to make some observations about the results. One is the correlation between stability and open-circuit voltage. In

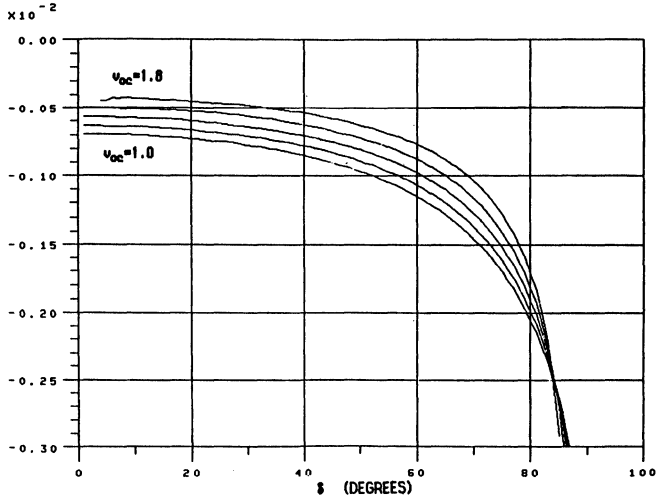


Fig. 1. Largest real roots of the continuous time system versus machine angle. Open-circuit voltage, V_{oc}, is the parameter.

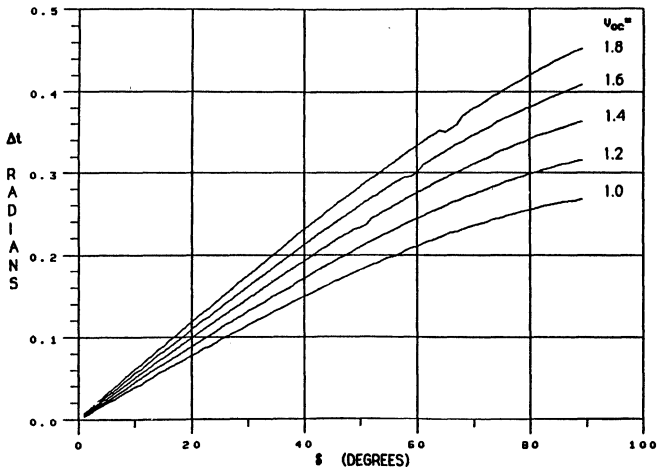


Fig. 2. Stability limits of the forward difference integration method. Normalized time step versus machine angle. Again, open-circuit voltage is the parameter.

all cases, a larger open-circuit voltage corresponds with more favorable stability conditions. Another feature of the results is the inverse relation between stability and angle. This is a sharp contrast to the stability behavior of real generators. The latter are more stable at small angles, whereas the integration routines are more stable at large angles. The relative merits of the integration routines is displayed in Fig. 6 as well for an open-circuit voltage of 1.8 per unit. Fourth-order Runge-Kutta is the most stable, followed by modified Euler, forward difference, and ABM-4 taking a disappointing last place. To pick the best method in terms of speed, it is necessary to balance the additional computations of the fourth-order Runge-Kutta method against the higher cycle rate required by the other methods. If the number of derivative evaluations is the judge of computational requirements, then for the sake of example it may be written that:

$$T_{\min} \text{ (4th order RK)} = 1.00 \text{ ms}$$

$$T_{\min} \text{ (Mod. Euler)} = 0.50 \text{ ms}$$

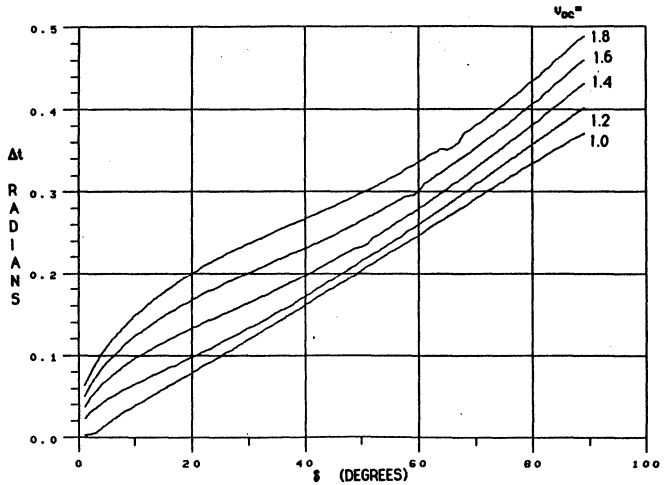


Fig. 3. Stability limits for the Euler method. SWame parameter.

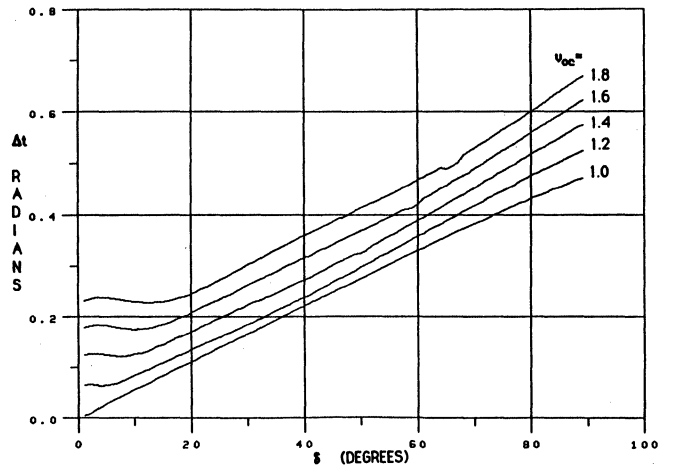


Fig. 4. Stability limits for the fourth-order Runge–Kutta method. Again normalized time step is shown versus rotor angle with open-circuit voltage as the parameter.

$$T_{\min}\ (\text{ABM}-4) = 0.50\ \text{ms}$$

$$T_{\min}\ (\text{Forward Diff}) = 0.25\ \text{ms}$$

where $T_{\min}$ is the time it takes an arbitrary computer to calculate the $n + 1$ set of variables. The ratio of $T_{\max}$ (stability limit) to $T_{\min}$ (computation time) for each method can be taken as a figure of merit. Unity and greater implies a real-time simulation is possible. The problem with this gauge of efficiency is demonstrated by its application to Fig. 6. For all angles greater than 20 degrees forward difference is picked as the best method. Not taken into account is the difference in accuracy between the integration techniques. Stable integration is no guarantee that meaningful results are being produced. In conclusion, it must be stated that a variety of factors influence the choice of an integration routine. It is difficult to obtain all the desired properties in one method. Some time was spend searching for more stable integration routines, but most were too complex to be considered for use in a real-time operation.

III. Generator Implementation and Results

A. Mainframe Simulation

To provide a source of comparison data, and to test the various methods of integrating the generator state equations, a

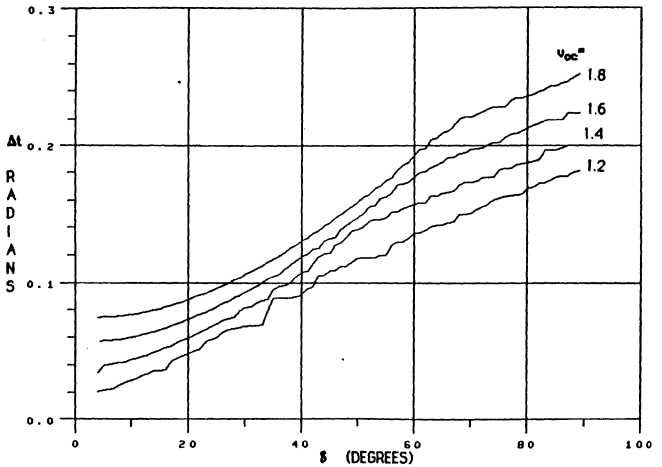


Fig. 5. Stability limits for the fourth order Adams–Bashforth–Moulton with predictor corrector.

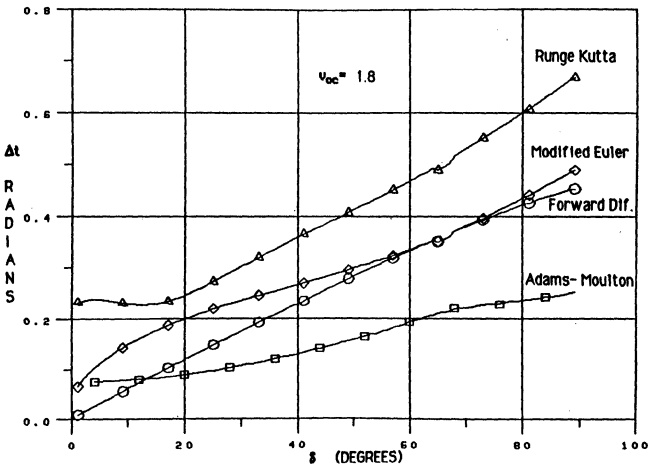


Fig. 6. Theoretical stability limits for all four methods of integration considered with experimental results shown as points. The per unit open-circuit voltage is 1.8.

Fortran program was developed on a DEC VAX 11/782. Any one of the four integration methods in Section II may be employed: fourth order Runge–Kutta, modified Euler, ABM-4, and a forward difference. The derivative evaluation routine may be set to have the simulated generator supply power to either a constant impedance load or an infinite bus. A power transient of any duration and constant magnitude may be applied to the simulated generator or a bus-voltage transient of any duration and constant magnitude may be applied to the simulated generator. This does not apply to the case of an impedance load.

B. Choice of Processor

With the generator state equations established, it was determined that a second order integration routine such as modified Euler would make at least 88 multiplications and 64 additions/subtractions per cycle. Since real-time operation is desired, a cycle time of 1 ms was targeted. All arithmetic and bookkeeping must fit into this interval. Besides the speed requirement, there are two other important considerations in the choice of a processor: Namely cost and availability. The simulation network is designed to handle as many as 100

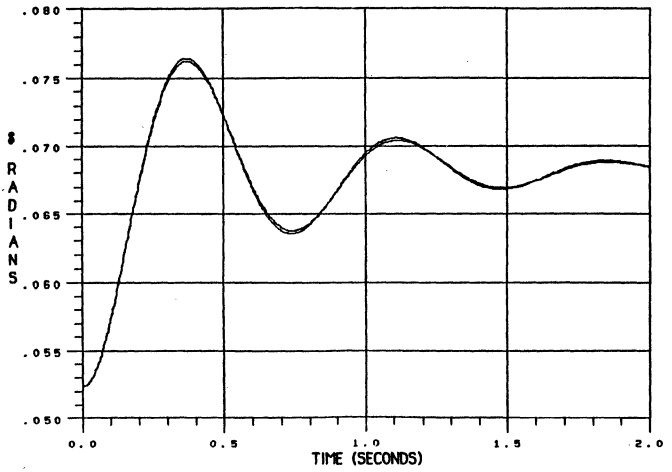


Fig. 7. TMS32010 generator simulation as compared to a benchmark case calculated using double-precision on a VAX 11/782. The power transient is +30 percent from a machine angle of 3 degrees and open-circuit voltage of 1.5 into an infinite bus.

generator modules, and at this quantity expense is certain to become a limitation. This makes the use of a mass-produced commercially available board desirable. One early prospect dismissed due to cost in time and money was having a microprocessor such as the Intel 8086 control an arbitrary number of Intel 8232 floating-point units. The parallelism of the state equations is such that several calculations could be done simultaneously. However, the 8232 takes nearly 15 μs to add or subtract and about 50 μs to multiply, implying that at least three would be required to even approach the target cycle time of 1 ms [17]. A stronger contender was a board that employed the 8086 microprocessor and the 8087 floating-point coprocessor. The times boasted by this combination are 14/18 μs for addition/subtraction and 19 μs for multiplication [18]. Although too slow, this looked like the best choice until the TMS32010 digital signal processor was considered. This processor can be programmed to do floating point arithmetic at a speed comparable to the 8087, and has the additional advantage of being able to multiply two 16-bit integers in 200 ns [19]. The hardwired multiplier makes the 32010 very efficient at fixed-point arithmetic. MacMinn [9] showed that with proper care, fixed-point arithmetic can be used successfully in a generator simulation. Texas Instruments offers a full 32010 development system for the relatively modest cost of about $700. This and its ready availability lead to the choice of the TMS32010EVM board as the generator module. This board uses a TMS9995 processor in a master-slave relationship with the 32010, a luxury that is not needed once the generator module is successfully meshed in the simulation network.

C. Implementation on the TMS32010

The first step of developing a generator simulation program for the TMS32010 involved obtaining 32010 assembler and simulation packages for the VAX. Programming in the VAX environment was found to be both convenient and efficient. The TMS32010 simulator allows I/O through arbitrary files,so that a Fortran program can be used to supply initial parameters to the simulator, and output simulation data is readily obtainable for use by a plotting package. The first version of the generator simulation program employed fourth-order Runge–Kutta and used only 16-bit variables. Unfortunately, the precision was insufficient. Many of the state equation variables had magnitudes large enough to push any increments out of the 16-bit range. Even when some of the key summations (those involving the subtraction of large numbers to yield small parameters) were done in double precision, proper operation could not be obtained. The program was rewritten to do most arithmetic in double precision (30 bits + sign bit, also fixed point). Only constants that could be defined as exact were left in 16-bit format (Section II-B). To compensate for the loss in execution speed, second-order Runge–Kutta integration was used. This method is essentially the same as modified Euler, except that a different set of constants is used. The linear stability of the two methods is the same. Figs. 7 and 8 compare TMS32010 simulation data to the benchmark fourth-order Runge–Kutta integration for power transients of plus and minus ten percent respectively. As this is for stand-alone operation, the generator program computes its own infinite bus voltage. The same results were obtained when the TMS32010 generator code was executed on one of the TMS32010EVM boards. With an integration cycle time of 0.53 ms, the TMS32010 program can, simulate a generator on an infinite bus at a sample rate of nearly 2 kHz. The above results were reproduced at a sample rate of about a kilohertz, implying the simulation was taking place at nearly double the real-time rate. To see what is possible with an impedance load, the 0.53 ms. is converted to 0.2 rad of simulation time. With

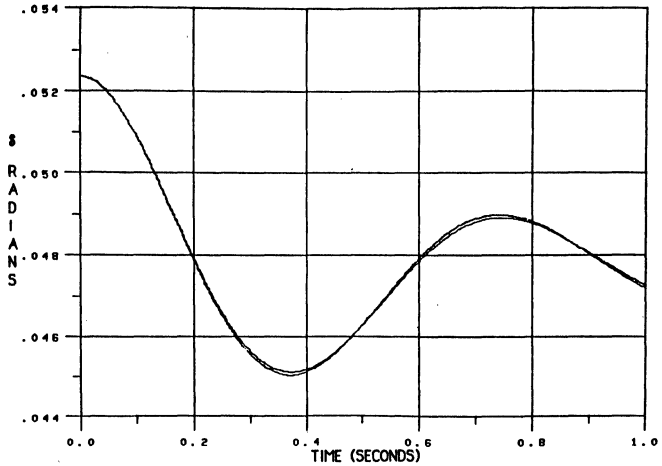


Fig. 8. Same as Fig. 7 but with a power transient of − 10 percent.

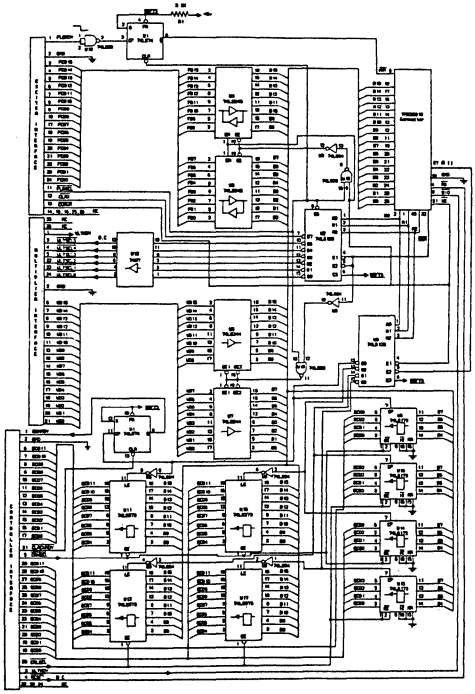


Fig. 9. Generator module to network interface.

reference to the stability limits of Section II, Fig. 3, it is seen that real-time simulation is possible to generator angles greater than about 9 degrees ($V_{oc} = 1.5$).

IV. Conclusion

This paper has examined the design of a real-time digital simulator of a synchronous generator which is to operate in a network of such generators (parallel processors), a transmission system and nonlinear loads. It was determined that the nonlinear load that the generator faces places a significant constraint on the simulation time step. The design must use a conservative estimate of time step in order for the simulation to be stable in the presence of the *a priori* unknown loads. A 16-bit generator simulator has been proposed and tested which uses the TMS32010 digital signal processing chip.

Appendix A

Interface Hardware

To work within the digital simulation network, the generator module must have a means of communicating with other elements of the network. Specifically, the generator module must transmit its currents to the exciter unit and to the voltage determining matrix multiplier. In return, the generator receives its field voltage from the exciter, and its terminal voltages from the multiplier.

Fig. 9 is a schematic of an interface designed to connect either the TMS32010EVM or the proposed generator module of Appendix B to the network. This circuit features a 16-bit bidirectional port between the generator and exciter, a 16-bit input port from the multiplier, and a 24-bit output port to a master controller. The controller is needed to orchestrate the transfer of all the generator modules' currents to each multiplier. Timing works as follows:

1) TMS32010 saves generator angle to memory with Table Write (TBLW) instruction.

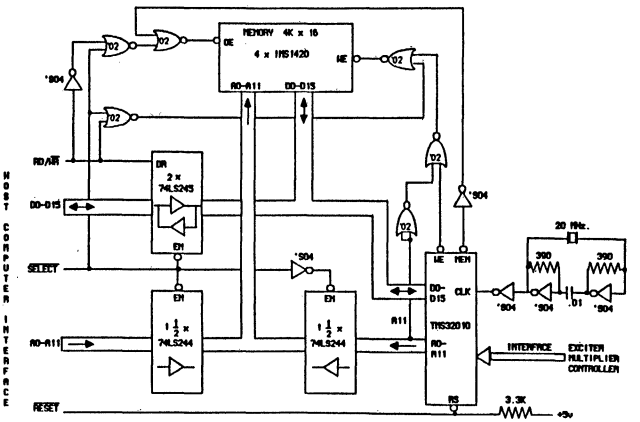


Fig. 10. Proposed generator module using the TMS32010 chip.

2) TMS32010 clears exciter input registers by a dummy read from port 7.
3) In 12-bit increments the TMS32010 writes the 24 most significant bits of id and iq to ports 0–3. Besides making the currents available to the controller, this operation also sends the currents to the exciter registers via a second bus. Access to port 3 activates the GENRDY signal.
4) TMS32010 disables the interrupt line in order to give the controller free access to the exciter bus.
5) When all generators signal that they are ready, the controller routes their currents to the multipliers. The controller deactivates the GENRDY signals, and when the voltages have been calculated, it notifies the generators via the VLTRDY line.
6) TMS32010 waits until the VLTRDY signal is activated, and then enables the interrupt line. Next, the two terminal voltages are read from the particular multiplier assigned to this generator. This is accomplished by reading two sets of 16, 16, and 6 bits from ports 0–5.
7) TMS32010 calculates machine variables for the next time step, then repeats the process from the top.
8) When the exciter has calculated a new field voltage for the generator, it activates the interrupt line of the TMS32010. The interrupt service routine causes the TMS32010 to read the 16-bit field voltage from port 6. Reading from port 6 also clears the interrupt holding flip-flop. Should the interrupt line be disabled when the exciter signals, then the TMS32010 will not respond until after the voltages have been calculated.

Appendix B

Proposed Generator Module and Host Interface

Fig. 10 is a proposed design for future generator modules. It is intended to yield a low cost and compact system. The total chip count of this and the interface hardware of Appendix A is 31, and the total cost is probably under $200.

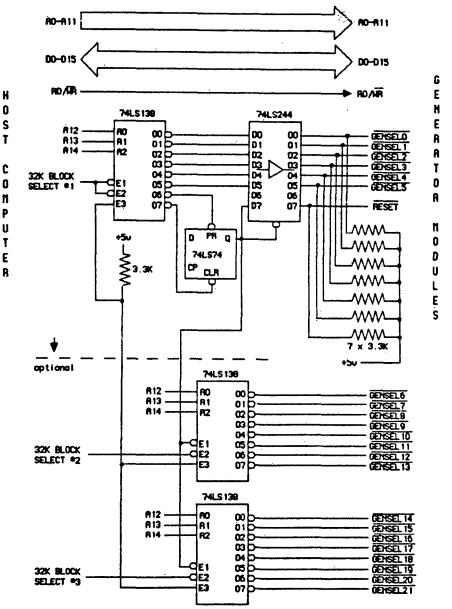


Fig. 11. Proposed host computer interface.

The only expensive components are the TMS32010 processor and the accompanying fast memory. The system features easy direct memory access by a host computer. This facilitates the downloading of programs and uploading of results. One the host computer has pulled the reset line low it may treat the generator memory just like any 4K block of its own memory. The generator simulation program resides in the lower 2K bytes of the external ram memory and calculated generator angles are stored in the upper 2K bytes. The TMS32010 is not allowed to write to the program portion of memory in order to prevent erasure of the first 8 bytes of memory by the OUT

instruction. (The external signals of the OUT and TBLW instructions are indistinguishable.) Fig. 11 is an example of a host computer interface that allows many generator modules to be accessed. The reset lines of the generators are under software control. By addressing one memory location the host can reset all the generator modules and open up their memories to full speed access. Expansion is simple, as the addition of one 74LS138 decoder chip allows another eight generators to be mapped into a 32K block of host address space.

References

[1] M. H. Kent, W. R. Schumus, and F. A. McCrackin, "Dynamic modeling of loads in stability studies," *IEEE Trans. Power App. Syst.*, vol. PAS-88, May 1969.

[2] R. T. Byerly and E. W. Kimbard, *Stability of Large Electric Power Systems.* New York: IEEE Press, 1974.

[3] P. L. Dandeno and R. L. Hauth, "Effects of synchronous machine modeling in large scale system studies," *IEEE Trans. Power App. Syst.*, vol. PAS-92, Mar./Apr. 1973.

[4] D. W. Olive, "Digital simulation of synchronous machine transients," *IEEE Trans. Power App. Syst.*, vol. PAS-87, Aug. 1968.

[5] Yoa-nan Yu, Jack H. Sawada, and M. D. Wvong, "A dynamic power system model for teaching and research," *IEEE Trans. Power App., Syst.*, vol. PAS-95, July/Aug. 1976.

[6] P. M. Anderson and A. A. Fouad, *Power System Control and Stability.* Ames, IA: Iowa State Univ., 1977.

[7] P. L. Dandeno, P. Kindur, and R. P. Schulz, "Recent trends and progress in synchronous machine modeling in the electric utility industry," *Proc. IEEE,* vol. 62, July 1974.

[8] R. Joetten, T. Weiss, J. Wolters, H. Ring, and B. Bjorensson, "A new real-time simulator for power system studies," presented at the PES Winter Meeting, 1985.

[9] S. R. MacMinn, "A digital model generator for real-time simulation," Ph.D. dissertation, Cornell Univ., Ithaca, NY, Aug. 1984.

[10] S. Wiryaman, "Modular implementation of a fast matrix-vector multiplier," M.S. thesis, Case Western Reserve Univ., Jan. 1985.

[11] F. Iliceto and A. Ceyhan, "Behavior of loads during voltage dips encountered in stability studies. Field and laboratory tests," *IEEE Trans. Power App., Syst.*, vol. 91, Nov./Dec. 1972.

[12] Computer Analysis of Power Systems Working Group of the Computer and Analytical Methods Subcommittee-Power System Engineering Committee, "System Load Dynamics—Simulation Effects and Determination of Load Constants," *IEEE Trans. Power App. Syst.*, vol. PAS-92, Mar./Apr. 1973.

[13] Tom T. Hartley and Guy O. Beale, "Integration operation design for real-time digital simulation," *IEEE Trans. Ind. Electron.*, vol. IE-32, Nov. 1985.

[14] Curtis F. Gerald, *Applied Numerical Analysis.* Reading, MA: Addison-Wesley, May 1980.

[15] H. W. Dommel and N. Sato, "Fast transient stability solutions," *IEEE Trans. Power App. Syst.*, vol. PAS-91, July/Aug. 1972.

[16] Charles L. Phillips and H. Troy Nagle, Jr., *Digital Control Systems Analysis and Design.* Englewood Cliffs, NJ: Prentice-Hall.

[17] Intel Corp., Component Data Catalog, Jan. 1981.

[18] Intel Corp., iAPX 286 Programmer's Reference Manual, 1984.

[19] Texas Instruments, TMS32010 User's Guide, 1983.

[20] Texas Instruments, TMS32010 Evaluation Module User's Guide, 1984.

Real-Time Dynamic Control of an Industrial Manipulator Using a Neural-Network-Based Learning Controller

W. THOMAS MILLER, III, MEMBER, IEEE, ROBERT P. HEWES, STUDENT MEMBER, IEEE, FILSON H. GLANZ, MEMBER, IEEE, AND L. GORDON KRAFT, III

Abstract—**The overall complexity of many robotic control problems, and the ideal of a truly general robotic system, have led to much discussion of the use of neural networks in robot control. A learning control technique is discussed which uses an extension of the CMAC network developed by Albus, and results of real-time control experiments are presented which involved learning the dynamics of a five-axis industrial robot (General Electric P-5) during high-speed movements. During each control cycle, a training scheme was used to adjust the weights in the network in order to form an approximate dynamic model of the robot in appropriate regions of the control space. Simultaneously, the network was used during each control cycle to predict the actuator drives required to follow a desired trajectory, and these drives were used as feedforward terms in parallel to a fixed gain linear feedback controller. Trajectory tracking errors were found to converge to low values within a few training trials, and to be relatively insensitive to the choice of control system gains. The effects of network memory size and trajectory characteristics on learning system performance were investigated.**

I. INTRODUCTION

NUMEROUS manipulator control schemes have been studied during the past decade. One approach involves using a dynamic model of the robot to calculate the joint drive torques for the specified trajectory (computed torque controllers) [1]–[5]. Recent work in this area has focused on efficient techniques for implementing robot dynamic models [6]–[16], custom parallel computer architectures suitable for high-speed implementation of robot dynamic models [17]–[20], and techniques for estimating dynamic model parameters [21]. While computed torque techniques are capable of providing excellent results if the complete dynamic model is known, they are generally inflexible in that the detailed model is highly specific to a particular robot and payload.

Considerable work has also been reported concerning the application of adaptive control techniques to the robotic control problem [22]–[35]. These adaptive control schemes have the advantage that in general they require no *a priori* knowledge of the robot dynamics. A general drawback to adaptive controllers is that the computational requirements for real-time parameter identification, and the sensitivities to numerical precision and observation noise, tend to grow undesirably as the number of system state variables increases [36].

Several investigators have presented learning control schemes for improving the performance in trajectory following tasks over successive attempts at following the same trajectory [37]–[39]. Typically, control torques for each time instant in the trajectory are adjusted iteratively based on observed trajectory errors at similar times during previous attempts. In the results presented by these investigators, the trajectories followed consistently converged on the ideal trajectories over several repetitions. A drawback to such control techniques is that they are only applicable to operations which are repetitive.

Recently there has been considerable interest in learning in the form of simple models of networks of neurons. The overall complexity of many robotic control problems, and the ideal of a truly general robotic system, have led to much discussion of the use of neural networks in robot control [40]–[50]. The basic theme of all such discussions is that of using the network to learn the characteristics of the robot/sensor system, rather than having to specify explicit robot system models. While there seems to be widespread interest in this problem within the neural network and robotics communities, relatively little has been reported in the nature of actual robot control experiments. This is due, at least in part, to the computational speed and stability problems encountered when using typical neural models in networks of sufficient complexity to be useful for realistic robot control problems.

Albus [51]–[54] proposed a unique control scheme developed from models of human memory and neuromuscular control. The control scheme was based on a neural model called CMAC (Cerebellar Model Arithmetic Computer) which, in a table look-up fashion, produced a vector output in response to a state vector input. In the controller, the state vector input was composed of position and velocity feedback from the robot joints, as well as additional state variables which provided a command input to the system. The output vector was the drive signal to the robot actuators. Assuming that the values in the table were adjusted correctly, the robot would automatically follow the correct trajectory if put in the correct initial state and given the correct command state (the

Manuscript received February 10, 1988; revised December 30, 1988. This work was partially supported by the National Science Foundation, Division of Information, Robotics and Intelligent Systems, under Grant IRI-8813225. Part of the material in this paper was presented at the Intelligent Robots and Computer Vision Conference, Cambridge, MA, November 7–11, 1988.

W. T. Miller, III, F. H. Glanz, and L. G. Kraft, III are with the Department of Electrical and Computer Engineering, Kingsbury Hall, University of New Hampshire, Durham, NH 03824.

R. P. Hewes is with Harvard University, Perkins Hall, Cambridge, MA 02138.

IEEE Log Number 8929804.

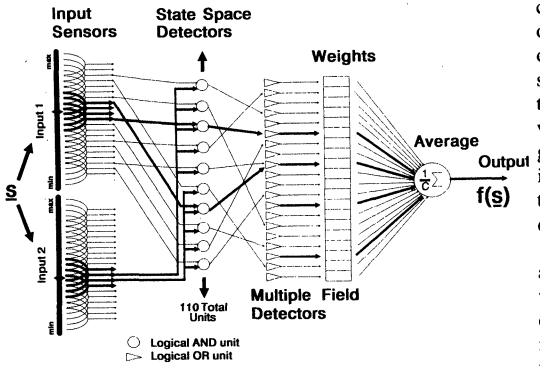


Fig. 1. A simple example of the network architecture with two inputs, one output, and four active units per input ($C = 4$). Note that only a subset of the state-space detectors is shown.

current input state would result in a set of actuator drives which would cause the arm to move, generating a new input state which would result in new actuator drives, and so on). While the system was capable of generating such "learned responses" once the memory was trained, training techniques which would make the control approach suitable for use in industrial robotics were not proposed.

During the past three years, we have been investigating a learning technique for the control of robotic manipulators [55]–[59] which utilizes a CMAC neural network similar to that developed by Albus. However, the control scheme is quite different from that proposed by Albus. The controller is similar to the computed torque controllers discussed above, with the robot dynamic model replaced by the neural network model. A training scheme is used to adjust the weights in the CMAC network on-line based on observations of the robot input/output relationships, in order to form an approximate dynamic model of the robot in appropriate regions of the state space. The CMAC network is used to predict the actuator drives required to follow a desired trajectory, and these drives are used as feedforward terms in parallel to a fixed-gain linear feedback controller.

The learning control technique developed in our laboratory has been previously evaluated in a simulation study involving learning the dynamics of a two-axis robot arm [55], and in real-time control studies which successfully demonstrate the ability to learn the kinematics of a robot/video camera system interacting with randomly oriented objects on a moving conveyor, during both repetitive and nonrepetitive operations [56]–[59]. This paper presents the results of real-time experiments which involved learning the dynamics of a five-axis industrial robot (General Electric P-5), during high-speed movements simulating industrial tasks.

II. Methods

A. The CMAC Network

Fig. 1 shows a simple example of the CMAC network as implemented in our laboratory, where s is a multidimensional, continuous-valued input vector and $f(s)$ is the network scalar output. Each component of the input vector s is fed to a series of input sensors with overlapping receptive fields. Each input sensor has a binary valued output, indicating whether or not the associated input value falls within its receptive field. The width of the receptive field of each sensor produces input generalization, while the offset of the adjacent field produces input quantization (similar to well-known coarse coding techniques). Each component of the input vector s excites exactly C input sensors ($C = 4$ in Fig. 1).

The outputs (on or off) of the input sensors are combined in a series of threshold logic units (called state-space detectors) with thresholds adjusted to produce logical AND functions (the output is on only if all inputs are on). Each of these units receives one input from the group of sensors for each input variable, and thus its input receptive field is the interior of a hypercube in the input hyperspace. If the input sensors were fully interconnected, a very large number of state-space detectors would be excited for each possible input. The input sensors are interconnected in a sparse and regular fashion, however, so that each input vector excites exactly C state-space detectors. The details of this input mapping are discussed elsewhere [52], [55].

The outputs of the state-space detectors are connected randomly to a smaller set of threshold logic units (called multiple-field detectors) with thresholds adjusted such that the output will be on if any input is on (a logical OR function). The receptive field of each of these units is thus the union of the fields of many of the state-space detectors. Since exactly C state-space detectors are excited by any input, at most C multiple-field detectors will be excited by any input. The converging connections between the large set of state-space detectors and the smaller set of multiple-field detectors are referred to as "collisions."

Finally, the output of each multiple-field detector is connected, through an adjustable weight, to an output averaging unit. The output for a given input is thus the average of the weights selected by the excited multiple-field detectors.

For a practical control problem, the total number of state-space detectors needed is large. However, since these units perform logical AND functions, and their interconnections with the input sensors are geometrically regular, they can be implemented as virtual units, and it is only necessary to consider a predictable set of C units for each input vector (C is typically less that 100). If the random interconnections between the state-space detectors and multiple-field detectors can be presented using a hashing function, it is possible to predict directly which weights are excited by a particular multidimensional input via a simple algorithm, and without analyzing all of the units in the network via a complete connectivity table. Software implementation of the network is thus very efficient, even for complex problems.

The CMAC network will produce an output $f(s)$ for any state vector s in the input space S, regardless of the number of adjustable weights in the memory. However, since the memory is typically much smaller than the total number of possible discrete input states, it is unlikely that a set of weights can be found which will produce the correct output for every

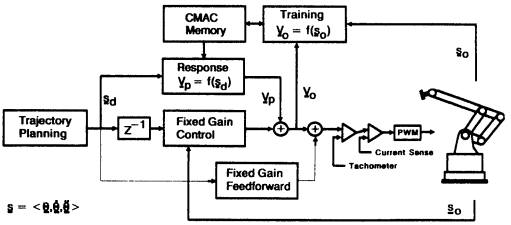


Fig. 2. A block diagram of the learning controller.

possible input state. On the other hand, it is also unlikely that every possible system state will be encountered in solving a particular control problem (even one that is nonrepetitive). Learning involves finding values for the weights which will result in correct network output for input states in the regions of interest.

B. Manipulator Control

Consider a multi-axis manipulator with drive electronics to be an electromechanical system represented by the general equation

$$V = m(\theta, \dot{\theta}, \ddot{\theta}) \tag{1}$$

where V is a vector of the applied actuator drives, θ, $\dot{\theta}$, and $\ddot{\theta}$, are vectors of the joint positions, velocities, and accelerations, respectively, and m represents a nonlinear vector function describing the inverse robot dynamics and actuator drive characteristics. If the function m is known, (1) can be used to calculate the joint drives required to follow a desired trajectory, and these estimated drives can be used as feedforward terms in parallel with a feedback controller. For typical manipulators, however, the function m is difficult to determine accurately and involves complex computations which are difficult to implement as part of a real-time controller.

The CMAC network can be applied to the manipulator control problem as follows (Fig. 2). Let the CMAC input state vector s be formed from the vectors θ, $\dot{\theta}$, and $\ddot{\theta}$, and let the CMAC function $f(s)$ correspond to the manipulator function $m(\theta, \dot{\theta}, \ddot{\theta})$ (the CMAC network can produce a vector rather than a scalar output if every weight in Fig. 1 is assumed to contain a vector value). The only assumption being made is that the drive signal for each axis is a function of the desired positions, velocities, and accelerations of all of the axes. No restrictions are placed on the forms of these functions, except that they be single-valued.

At each control cycle, the trajectory planner determines the desired state of the system s_d for the next control cycle (the desired positions, velocities, and accelerations of the actuators) based on the ideal trajectory. The desired next state s_d is sent to the CMAC network which produces $f(s_d)$. The resulting vector value is assumed to be an estimate of the actuator drives required to achieve the desired state s_d and is added to the output of the fixed gain error feedback controller to form the command vector V which is sent to the robot actuator drivers.

At the end of each control cycle a training step is executed. The observed state of the system s_0 during the previous control cycle is used as input to the CMAC network which produces $f(s_0)$. The difference between the predicted drive value $f(s_0)$ and the actual applied command vector V_0 during the previous control cycle is used to compute the weight vector adjustment as follows:

$$dw = \beta^*(V_0 - f(s_0)) \tag{2}$$

where β is a training gain between 0 and 1. This correction vector is added to each of the weight vectors excited by the input state s_0. Note that this training procedure is similar to the well-known Widrow–Hoff training procedure for linear adaptive elements [60], [61]. The nonlinear characteristics of the CMAC neural network are embodied in the interconnections of the input sensors, state-space detectors, and multiple-field detectors, which perform a fixed nonlinear mapping of the continuous-valued input vector s to a many-dimensional binary-valued vector (the set of outputs from all of the multiple-field detectors). The training is linear in this many-dimensional space and the convergence theorems for linear adaptive elements apply [61].

When the system is initialized, the weights contain all zeros such that $f(s_d)$ is the null vector for any desired state s_d and the command vector set to the robot is equal to the output of the fixed gain controller alone. As the CMAC network is continually trained following successive control cycles, the CMAC function $f(s)$ forms an approximation of the system inverse dynamic transfer function $m(\theta, \dot{\theta}, \ddot{\theta})$ over particular regions of the state space. If the future desired states are in regions of the state space similar to previous observed states, the CMAC network output will be similar to the actual actuator drives required. As a result, the state errors will be small and the CMAC network will take over from the fixed-gain controller. The more experience the controller obtains, the more closely the CMAC output $f(s)$ approximates the actual system tranfer function in the appropriate regions of the state space. Note that while a repetitive trajectory may be the easiest to learn, the technique is applicable to nonrepetitive operations. The trained information is in the form of the system transfer characteristics at individual points in the state space, and is not explicitly related to the overall trajectory.

C. The Experimental Model

For this study, the learning control system was implemented using a VAX-11/730 minicomputer with a TMS32010 auxiliary processor. The basic control architecture is shown in Fig. 2. The robot was a General Electric P-5 five-axis articulated robot. This robot was driven by five 100-V dc motors with pulsewidth-modulated motor drivers. Feedback was available to the digital controller in the form of a pulse train position encoder for each axis. Maximum speed varied for each axis but was on the order of 100°/s for each. The position encoder resolution was on the order of 0.01° for each axis. Note that the drive signal being learned was the input to analog motor driver circuits containing analog tachometer and current sense feedback loops.

A digital fixed-gain feedback controller was designed for each axis, including position error (encoder count units) and

velocity error (1 count/cycle units) terms. Gains were adjusted experimentally to give good performance. A fixed-gain velocity feedforward term was included in the controller. This term was added to the total drive after the training drive observation (Fig. 2). Thus the network was trained to represent the difference between the actual system and the approximate model (fixed feedforward). The "tuned" gain for the velocity feedforward term was obtained from a knowledge of the gain in the tachometer feedback loop for the corresponding axis.

In the learning module, the drive signal for each motor was assumed to be a function of the desired positions (800 count units), velocities (4 count/cycle units), and accelerations (2 count/cycle/cycle units) of all five axes. The neural network thus had fifteen discrete numeric inputs and five discrete numeric outputs, with approximately 1 000 000 virtual state-space detector units and 32 active units per input ($C = 32$). The number of multiple-field detector units and actual weight vectors in the memory varied from 32 768 to 8, as indicated in the results. The training gain was set to 0.05 in all experiments.

As discussed above, the ideal components for the input state vector s included the actuator positions at the beginning of a control cycle, actuator velocities at the beginning of a control cycle, and actuator accelerations during the control cycle. For the P-5 robot, however, only direct measurements of position were available to the digital controller. A symmetrical quadratic least squares estimator was used to estimate velocity from five sequential positions. A central difference estimator was used to estimate acceleration from two sequential estimated velocities. The actual input state vectors used were thus functions of six sequential positions (three past and three future) for each actuator. During control computations, the future desired positions were readily available since the entire desired trajectory was known in advance. During training, the weight adjustment for a given control cycle was delayed by three cycles such that "future" position measurements would be available for the training computations.

The control cycle time was 20 ms, accommodating both one learned feedforward computation (5.4 ms) and one training computation (7.7 ms). This control cycle time was marginal for the high-speed movements tested, but was possible because of the analog velocity feedback loops in series with the PWM motor drivers.

Two test trajectories were designed. The first (CIRCLES) involved tracing ten tangential circles in three orthogonal Cartesian planes, holding the wrist orientation constant relative to the upper arm. This smooth trajectory was considered difficult in that the positions, velocities, and accelerations of the five individual actuators (including the wrist axes) were constantly changing during the 24-s exercise, and the peak drive voltage required for each of the five actuators was at least 80% of the saturation drive. The second test trajectory (SEGMENTS) involved a series of long constant high actuator velocity moves (50% to 95% of maximum actuator velocity) separated by abrupt changes in velocity (brief intervals of high acceleration). The total trajectory duration was 6.6 s. This trajectory was difficult in that only limited information about

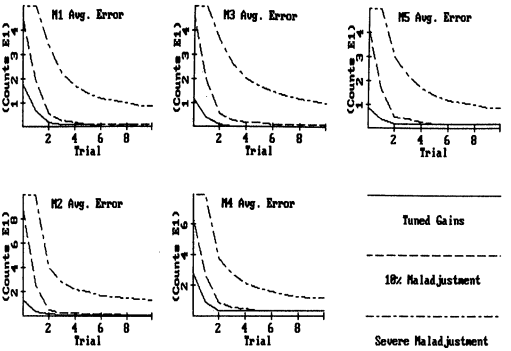


Fig. 3. The average position errors as a function of training trial for the 24-s exercise (CIRCLES) described in the text. Trial 0 corresponds to the fixed-gain controller without learning. Error curves are shown for each of the five motors during experiments using the tuned-velocity feedforward, slightly reduced velocity feedforward (10% maladjustment), and no velocity feedforward (severe maladjustment).

the system dynamics was available during the long constant actuator velocity intervals which dominated the trajectory in terms of total control cycles, and yet good system dynamics information was required in order to successfully achieve the desired abrupt velocity changes.

III. Results

Fig. 3 shows the average trajectory position error, in position encoder units, for CIRCLES during each of ten sequential trials. Error curves are shown for the optimal velocity feedforward ("tuned gains"), for a 10% reduction in feedforward gain ("10% maladjustment"), and for no velocity feedforward ("severe maladjustment"). In each case, the intercept with the vertical axis (trial 0) indicates the average trajectory error of the fixed gain controller without learning. The maximum position error for each axis followed the same trend during training as the average error. The position errors for the fourth axis were higher because that motor was driven by an 8-b D/A converter, while the other four motors were driven by 12-b D/A converters.

Control system performance improved significantly, even when using the tuned fixed-gain controller, converging within five training cycles. Detuning the fixed feedforward (approximate system model) by only 10% had a large effect on performance without learning, but had essentially no effect on performance after five training trials. Without velocity feedforward, the average control errors for the five motors were 261, 424, 247, 255, and 201 counts (about five times greater than the plot vertical scale in Fig. 3) when using the fixed gain controller without learning. Even for this severe detuning of the fixed gain controller, average trajectory position error converged to a low value within ten training trials. For all actuators, the final error was similar to or less than the trajectory error for the tuned controller without learning.

Fig. 4 shows the percentage of the total drive signal for each motor which was provided by the combined feedforward terms (fixed gain velocity feedforward and learned feedforward) as a function of training trial during the experiments depicted in

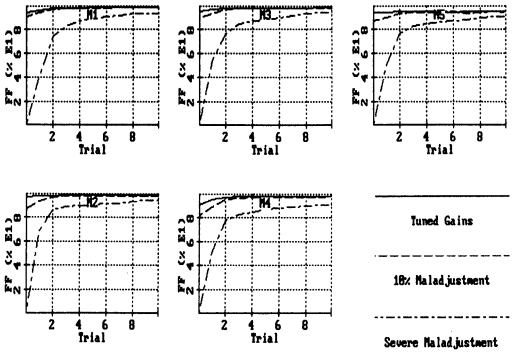


Fig. 4. The average feedforward drive magnitude as a percentage of the average total drive magnitude during the three experiments depicted in Fig. 3.

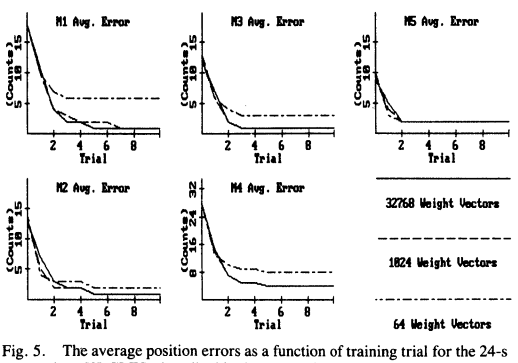


Fig. 5. The average position errors as a function of training trial for the 24-s exercise (CIRCLES) described in the text. Trial 0 corresponds to the fixed-gain controller without learning. Error curves are shown for each of the five motors during experiments using network memories containing 32 768 weight vectors, 1024 weight vectors, and 64 weight vectors.

Fig. 3. Percent drive was computed as follows:

$$FF\,\% = 100 * \frac{|FF|}{|FF| + |FB|} \tag{3}$$

where $|FF|$ represents the summed absolute drive signal from the combined feedforward terms and $|FB|$ represents the summed absolute drive signal from the combined error feedback term.

By this measure, the tuned velocity feedforward provided over 90% of the total drive signal, even without learning. This is consistent with both the nature of the mechanical system and the presence of the analog tachometer feedback loops in the motor drive electronics. Learning was able to increase this percentage to 98% or 99% for each motor (the values for motor five are lower due to the lower total average drive magnitude for this motor). The 10% maladjustment of the velocity feedforward reduced its contribution to the total drive signal, but had almost no effect on the total feedforward contribution after five training trials. With the velocity feedforward gains set to zero, the learned feedforward term still accounted for over 90% of the total drive signal after ten training trials.

In these trials, a network memory containing 32 768 weight vectors was available (10 bytes per vector). After ten 24-s training trials, each involving 10 circles in Carteisan planes, 2708 vectors had been accessed when using the tuned fixed gain controller. This implies that practical length tasks can be implemented using realistic amounts of memory. In order to test the effect of memory size on performance, the experiment was repeated using networks with memory sizes of 1024 weight vectors and 64 weight vectors. The results are shown in Fig. 5.

Learning controller performance was essentially the same for the 1024-vector memory as for the 32 768-vector memory, even though the 2708 vectors used for the same problem with the larger memory imply that many collisions were certain. This indicates the network's ability to resolve collisions in the connections between the very large set of state-space detectors and the much smaller set of multiple-field detectors. Performance was measurably impaired when using the network with only 64 weight vectors, but after five training trials was still significantly better than that obtained using the tuned fixed-gain controller alone.

It is informative to consider individual weights from the network memories. Given a sufficiently large memory, each individual weight is influenced primarily by training data corresponding to a particular region of the system state space. After several training trials, each network weight would reach a stable value, assuming that the system's properties are constant. In a smaller memory, each weight is influenced by training relative to multiple disjoint regions of the state space. Within reason, however, such network collisions should be resolved with sufficient training, resulting again in stable weight values. For very small network memories, each weight will be influenced by many or most regions of the state space, and the continuous training is likely to act as a low-pass filter, with each weight tracking the applied drive signal rather than reaching a stable value.

In order to confirm these assumptions, individual weights were monitored during training trials with networks including 32 768 weight vectors, 1024 weight vectors, 64 weight vectors, and 8 weight vectors. Typical results are shown in Fig. 6 for individual weights as functions of the time during the 16th training trial for each network. The weights shown in the figure were deliberately chosen as having similar magnitudes (in order to facilitate comparison) but bore no other relation to each other. As expected, the weights reached stable values for both the large network memory (with relatively few likely network collisions) and for the smaller memory (where many collisions were certain). Even for the very small memory (64 vectors), a stable average value with only a small deviation is clear. For the tiny network memory (8 weight vectors), the magnitude varies constantly during the trial, essentially tracking the applied drives as the result of the training algorithm. Note that although this weight appears unstable in the figure, the apparent oscillation was the direct result of the varying actuator drive required to track the sequence of circles, and did not reflect numerical training instability.

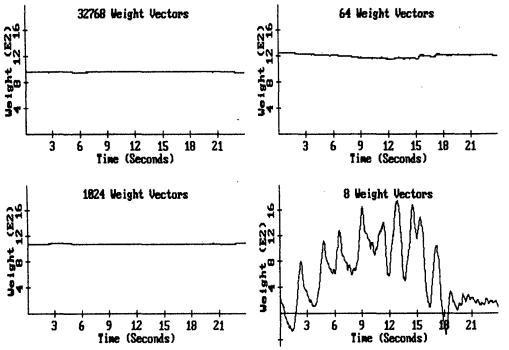


Fig. 6. Individual network weights as functions of time during the 16th training attempt for the CIRCLES trajectory in experiments using network memories 32 768 weight vectors, 1024 weight vectors, 64 weight vectors, and 8 weight vectors.

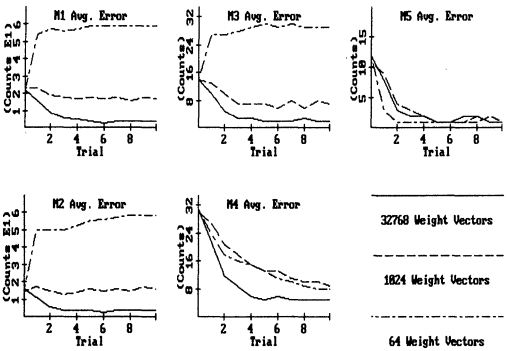


Fig. 7. The average position errors as a function of training trial for the 6.6-s exercise (SEGMENTS) described in the text. Trial 0 corresponds to the fixed-gain controller without learning. Error curves are shown for each of the five motors during experiments using network memories containing 32 768 weight vectors, 1024 weight vectors, and 64 weight vectors.

The same experiments were repeated for the SEGMENTS trajectory. Fig. 7 shows the trajectory position errors for the five actuators during a sequence of ten training trials for networks containing 32 768, 1024, and 64 weight vectors. For the larger network memory, performance converged rapidly to a low error during the first five trials, similar to the results for CIRCLES. Of the total available, only 1115 weight vectors were modified during the trials, which again was consistent with CIRCLES given the shorter duration of SEGMENTS. However, when the memory size was decreased to 1024 weight vectors, increasing the frequency of collisions, the performance degraded noticeably relative to the larger memory. When the network memory was further reduced to 64 weight vectors, the average trajectory position errors with training were actually greater than when using the tuned fixed-gain controller alone.

At first, these results seemed contradictory in that CIRCLES used three times more weight vectors in the large network than SEGMENTS, and yet CIRCLES mapped much more successfully into the smaller networks. This difference can be explained, however, by considering the nature of the two trajectories. CIRCLES involved constantly and smoothly varying accelerations for all five axes. As a result, the training data were relatively rich in information about the system dynamics, with substantial generalization between sequential control cycle observations. Learned information was reinforced both during successive control cycles during each trial, and from one trial to the next. The effects of the frequent collisions in the network connections in the smaller memories were thus able to be resolved.

On the other hand, SEGMENTS involved sequences of long movements with constant actuator velocity, separated by short intervals of high acceleration. The training data during the long constant velocity intervals contained relatively little dynamic information, and the short intervals of high acceleration were quite different from the nearby control cycles, providing little reinforcement of the necessary dynamic information during each trial. When using the large network there was sufficient reinforcement from one trial to the next, with little destruction of information due to collisions, so that the performance converged to low errors. When using the smaller memories with large numbers of collisions, however, the information about the trajectory corners was not reinforced sufficiently during each trial to offset the destruction of information by collisions.

While trajectories which require moving all five axes simultaneously are the best test of performance, it is difficult to evaluate the results other than as error statistics. For this reason, a simple trajectory demonstrating the advantage of the learned feedforward term was devised. The arm was retracted and then the base axis was rotated through 80° at approximately 70% of full speed. The desired acceleration was set to a constant magnitude of 11 counts/cycle/cycle at the beginning and end of the move. The arm was then extended, and an opposite rotation of the base performed. The desired velocity profile for the base axis was the same for the retracted and extended portions of the trajectory. However, the required drive signal was quite different as the result of the configuration-dependent inertial terms. Fixed gain velocity and acceleration feedforward terms could not possibly generate the correct drive signals for both rotations.

Fig. 8(a) shows the base-axis trajectory errors during the acceleration portions of the retracted and extended moves using two different controllers: error feedback with optimal velocity feedforward, and error feedback with learned feedforward only (after 15 attempts). The fixed gain velocity feedforward resulted in low error for the retracted case, but significant lag occurred for the extended arm during the acceleration phase. Addition of acceleration feedforward could have helped to reduce this lag, but at the cost of overdriving the retracted arm. The controller with learned feedforward showed small error for both the retracted and extended cases.

Fig. 8(b) shows plots of the corresponding base drive signals. The dashed lines correspond to the drive saturation level. The dotted lines indicate the drives that were (or would have been) predicted by the fixed-gain velocity feedforward. Note that for the retracted case, the velocity feedforward term

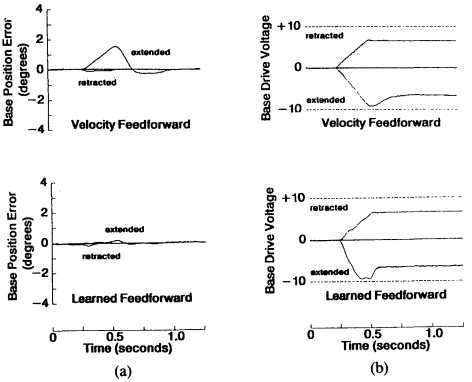


Fig. 8. Robot base axis position error (a) and drive voltage (b) as a function of time during base rotations with the arm retracted and extended. Corresponding data are shown for the controller with tuned-velocity feedforward and with learned feedforward only (after 15 trials).

predicted nearly the correct drive signal. A similar drive signal was learned by the learned feedforward term when the arm was retracted. For the extended case, the velocity feedforward term did not predict the large transient drive required to accelerate the arm, thus the arm lagged behind the desired trajectory until sufficient error drive was generated to accelerate the arm. On the other hand, the learning controller learned to apply a large drive signal immediately in order to accelerate the extended arm, and then to decrease the drive at the beginning of the fixed velocity phase.

IV. Discussion

Computation time for the 15-input, 5-output nonlinear learning problem posed was 5.5 ms for the feedforward computation and 7.7 ms for the training computation (feedforward plus weight adjustment) using the VAX-11/730–TMS32010 processor pair. These times were adversely affected by the relatively slow UNIBUS pathway between the two processors (a factor-of-four improvement in speed for the same size problem has been achieved in our laboratory using a closely coupled 68000–TMS32010 processor pair). For general use in the dynamic control of robotic manipulators, overall control cycle times on the order of 1 ms or less would be desirable. The natural parallel structure of the network makes it well suited to parallel implementation, using multiple RISC processors or special-purpose digital or analog hardware. We are currently developing an implementation of the CMAC neural network using standard cell arrays on a dual-height VME module. This hardware network implementation will include 1 048 576 8-b adjustable weights and will be able to perform control or training operations, similar in dimension to those discussed in this paper, in approximately 100 μs.

The results presented clearly indicate that with sufficient memory the learning controller converges to a low error within a few trials. This observation is consistent with the results of our previous simulation [55] and experimental studies [56]–[59]. While good performance was generally possible with the carefully adjusted fixed-gain controller, control system performance without learning was highly sensitive to controller maladjustment. In contrast, control system performance with learning was relatively insensitive to control parameter selection, resulting in control errors lower than or comparable to the tuned fixed-gain controller, even for severe parameter maladjustment.

While the learning controller was relatively insensitive to the gains chosen for the fixed-gain controller, there was an obvious symbiotic relationship between the learning system and the fixed-gain error feedback. This is evident from the fact that the low trajectory position errors achieved with learning were well below the resolution of the position variables used in the network input vector. After training, the learning system formed a discrete model of the nonlinear system properties, and the fixed gain error terms (which were implemented at the full measurement resolution) served to correct remaining differences between this discrete nonlinear model and the real system.

An obvious adjustment to achieve even better performance might be to use the position, velocity, and acceleration terms at their full resolutions in the network input vector. Increasing input variable resolutions, however, decreases network generalization, adversely affecting performance, unless the number of active units per input state (the parameter C) is correspondingly increased. In our current software implementation, computation time is slightly less than proportional to C, prohibiting the use of very large values. In parallel implementations, network response time could be made nearly independent of C (if the amount of parallel hardware was proportional to C), allowing large values and correspondingly increased input variable resolutions.

Many learning control schemes are applicable only to repetitive tasks [37]–[39]. The learning system being developed in our laboratory does not suffer from this restriction, since the trained information is in the form of the system transfer characteristics at individual points in the state space, and is not explicitly associated with the trained trajectories [57], [59]. Clearly, it would be difficult to train a system to generate correct control outputs for every possible control objective. However, it is also difficult to imagine a useful nonrepetitive task that truly involved making random motions spanning the entire control space of the mechanical system. The ability to learn to perform a variety of movements within a reasonable operating window should be sufficient for most useful nonrepetitive operations. This follows the concept of an "expert" robot as being one which is trained for a certain class of operations, rather than one which is trained for virtually all possible tools and applications.

The results obtained using the first trajectory demonstrate that tasks of reasonable complexity can be learned successfully using a network with relatively few weights. Thus it should be possible to train a system for a variety of operations using a network memory of practical size. The results obtained using the second trajectory tested, however, illustrate the pitfalls possible when trying to utilize a very small memory if important information is seen infrequently in the training data. The characteristics of the planned trajectories clearly have a direct impact on the ability to learn and to retain previously

learned information. This relationship has yet to be clearly defined.

Currently accepted approaches to handling the nonlinearities of robot dynamics during high-speed movements are generally related to the computed torque technique. In order to obtain high speeds and sufficiently high control rates with these techniques, computer code must be tailored to the specific robot to such an extent that transportability is impossible. Furthermore, attachment of a new tool requires code modification. Although many different adaptive and learning algorithms have been discussed recently in the literature as being of potential use in robotics, few have been tested in real time using an industrial manipulator. Some are too difficult to apply to a realistic manipulator with five or more axes (typically fifteen or more input variables). Others are too complex to implement in real time on typical hardware for suitable control cycle times. Investigation of these techniques has typically been limited to simulation studies using simplified models.

In contrast, the learning controller presented here is well suited for practical application to the control of industrial robotic manipulators. The learning algorithm structure is simple and is independent of the choice of learning system parameters (the number of state variables, the size of the weight vector memory, the number of weight vectors accessed by each state, and so on). This makes adaptation of the control software to accommodate system changes unnecessary or relatively easy, and makes it possible to transport large portions of the control software from one robot to another. In addition, the algorithm is time-efficient, highly parallel, and can be implemented in real time using current, low-cost technology. Finally, the learning control system appears to provide good dynamic performance relative to other adaptive or learning control schemes.

While the focus of this research was the dynamic control of industrial manipulators, the technique described is applicable to a wide range of robotics control problems which will be increasingly important in the future. For example, the use of low-mass materials in the construction of robots, for applications in space or on mobile platforms, will almost certainly require the use of high-performance learning controllers, since the control characteristics will be highly payload/task dependent. As another example, the problem of sensor data fusion (combining information from multiple dissimilar sensors to achieve a single control objective) makes it difficult to derive explicit control transformations, but can be systematically approached using learning techniques [56]–[58].

References

[1] R. P. Paul, "Modelling, trajectory calculation and servoing of a computer controlled arm," Stanford Artificial Intelligence Lab. Memo AM-177, Nov. 1972.

[2] J. Y. S. Luh, M. W. Walker, and R. P. Paul, "Resolved-acceleration control of mechanical manipulators," *IEEE Trans. Automat. Control.*, vol. AC-25, pp. 468–474, 1980.

[3] C. G. S. Lee, M. J. Chung, T. N. Mudge, and J. L. Turney, "On the control of mechanical manipulators," in *Proc. 6th IFAC Symp. on Identification and System Parameter Estimation* (Washington, DC, June, 1982), pp. 1454–1459.

[4] C. H. An, C. G. Atkeson, J. D. Griffiths, and J. M. Hollerbach, "Experimental evalution of feedforward and computer torque control," in *Proc. IEEE Int. Conf. on Robotics and Automation* (Raleigh, NC, Mar. 31–Apr. 3, 1987), pp. 165–168.

[5] C. P. Neuman and V. D. Tourassia, "Robust discrete nonlinear feedback control for robotic manipulators," *J. Robotic Syst.*, vol. 4, pp. 115–143, Feb. 1987.

[6] J. M. Hollerbach, "A recursive Lagrangian formulation of manipulator dynamics and a comparative study of dynamics formulation complexity," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-10, pp. 730–736, 1980.

[7] J. Y. S. Luh, M. W. Walker, and R. P. Paul, "On-line computational scheme for mechanical manipulators," *Trans. ASME, J. Dyn. Syst., Meas. Contr.*, vol. 120, pp. 69–76, 1980.

[8] M. W. Walker and D. E. Orin, "Efficient dynamic computer simulation of robotic mechanisms," in *Proc. 1981 Joint Automatic Control Conf.* (June 17–19, 1981), vol. 1, pp. WP-2B/1-9.

[9] M. Vukobratovic, L. Shi-Gang, and N. Kircanski, "An efficient procedure for generating dynamic manipulator models," *Robotica,* vol. 3, pp. 147–152, 1985.

[10] C. A. Balafoutis, P. Misra, and R. V. Patel, "Recursive evaluation of linearized dynamic robot models," *IEEE J. Robotics Automat.*, vol. RA-2, pp. 146–155, Sept. 1986.

[11] J. Koplik and M. C. Leu, "Computer generation of robot dynamic equations and related issues," *J. Robotic Syst.*, vol. 3, pp. 301–319, Fall, 1986.

[12] C. P. Neuman and J. J. Murray, "Customized computational dynamics," *J. Robotic Syst.*, vol. 4, pp. 503–526, Aug. 1987.

[13] ——, "Symbolically efficient formulation for computational robot dynamics," *J. Robotic Syst.*, vol. 4, pp. 743–769, Dec. 1987.

[14] ——, "The complete dynamic model and customized algorithms of the Puma robot," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-17, pp. 635–644, July/Aug. 1987.

[15] J. P. Wander and D. Tesar, "Piplined computation of manipulator modeling matrices," *IEEE J. Robotics Automat.*, vol. RA-3, pp. 556–566, Dec. 1987.

[16] G. Rodriguez, "Kalman filtering, smoothing, and recursive robot arm forward and inverse dynamics," *IEEE J. Robotics Automat.*, vol. RA-3, pp. 624–639, Dec. 1987.

[17] H. Kasahara, and S. Narita, "Parallel processing of robot-arm control computation on a multimicroprocessor system," *IEEE J. Robotics, Automat.*, vol. RA-1, pp. 104–113, June, 1985.

[18] R. Nigam and C. G. S. Lee, "A multiprocessor based controller for the control of mechanical manipulators," *IEEE J. Robotics Automat.*, vol. RA-1, pp. 173–182, Dec. 1985.

[19] C. G. S. Lee and P. L. Chang, "Efficient parallel algorithms for robot forward dynamic computation," in *Proc. IEEE Int. Conf. on Robotics and Automation* (Raleigh, NC, Mar. 31–Apr. 3, 1987), pp. 654–659.

[20] Y. Wang and S. E. Butner, "A new architecture for robot controls," in *Proc. IEEE Int. Conf. on Robotics and Automation* (Raleigh, NC, Mar. 31–Apr. 3, 1987), pp. 664–670.

[21] C. G. Atkeson, C. H. An, and J. M. Hollerbach, "Estimation of inertial parameters of manipulator loads and links," *Int. J. Robotics Res.*, vol. 5, pp. 101–119, Fall, 1986.

[22] S. Dubowsky and D. T. DesForges, "The application of model referenced adaptive control to robotic manipulators," *Trans. ASME, J. Dyn. Syst., Meas. Contr.*, vol. 101, pp. 193–200, 1979.

[23] M. Takegaki and A. Arimoto, "An adaptive trajectory control of manipulators," *Int. J. Contr.*, vol. 34, pp. 219–230, 1981.

[24] A. J. Koivo and T. H. Guo, "Adaptive linear controller for robotic manipulators," *IEEE Trans. Automat. Contr.*, vol. AC-28, pp. 162–171, 1983.

[25] G. G. Leininger, "Adaptive control of manipulators using self-tuning methods," in *First International Symposium on Robotics Research,* M. Brady and R. Paul, Eds. Cambridge, MA: MIT Press, 1984.

[26] C. G. S. Lee and M. J. Chung, "An adaptive control strategy for mechanical manipulators," *IEEE Trans. Automat. Contr.*, vol. AC-29, pp. 837–840, 1984.

[27] J. J. Craig, P. Hsu, and S. S. Sastry, "Adaptive control of mechanical manipulators," *Int. J. Robotics Res.*, vol. 6, pp. 16–28, Summer 1987.

[28] C. W. deSilva and J. Van Winssem, "Least squares adaptive control for trajectory following robotics," *Trans. ASME,* vol. 109, pp. 104–110, June 1987.

[29] J.-Y. Han, H. Hemani, and S. Yurkovich, "Nonlinear adaptive control of an N-link robot with unknown load," *Int. J. Robotics Res.*, vol. 6, pp. 71–86, Fall 1987.

[30] K. Y. Lim and M. Eslam, "Robust adaptive controller designs for

robot manipulator systems," *IEEE J. Robotics Automat.*, vol. RA-3, pp. 54–66, Feb. 1987.
[31] H. Seraji, "Direct adaptive control of manipulators in Cartesian space," *J. Robotic Syst.*, vol. 4, pp. 157–178, Feb. 1987.
[32] ——, "An approach to multivariable control of manipulators," *Trans. ASME,* vol. 109, pp. 146–154, June 1987.
[33] ——, "A new approach to adaptive control of manipulators," *Trans. ASME,* vol. 109, pp. 193–202, Sept. 1987.
[34] J. E. Slotine and W. Li, "Adaptive manipulator control. A case study," in *Proc. IEEE Int. Conf. on Robotics and Automation* (Raleigh, NC, Mar. 31–Apr. 3, 1987), pp. 1392–1400.
[35] ——, "On the adaptive control of robot manipulators," *Int. J. Robotics Res.*, vol. 6, pp. 45–59, Fall 1987.
[36] A. H. Levis, S. I. Marcus, W. R. Perkins, P. Kokotovic, M. Athans, R. W. Brockett, and A. S. Willsky, "Challenges to control: A collective view," *IEEE Trans. Automat. Contr.*, vol. AC-32, pp. 275–285, 1987.
[37] S. Arimoto, S. Kawamura, and F. Miyazaki, "Bettering operation of robots by learning," *J. Robotics Syst.*, vol. 1, pp. 123–140, 1984.
[38] C. G. Atkeson and J. McIntyre, "Robot trajectory learning through practice," in *Proc. IEEE Int. Conf. on Robotics and Automation* (San Francisco, CA, Apr. 7–10, 1986), pp. 1737–1742.
[39] M. Togai and O. Yamano, "Learning control and its optimality: Analysis and its application to controlling industrial robots," in *Proc. IEEE Int. Conf. on Robotics and Automation* (San Francisco, CA, Apr. 7–10, 1986), pp. 248–253.
[40] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-13, pp. 834–846, Sept./Oct. 1983.
[41] E. Ersu and H. Tolle, "A new concept for learning control inspired by brain theory," in *Proc. IFAC 9th World Congr.* (Budapest, Hungary, July 2–6, 1984).
[42] H. Ritter and K. Schulten, "Topology conserving mappings for learning motor tasks," in *AIP Conf. Proc.*, no. 151, *Neural Networks for Computing* (Snowbird, UT, Apr. 13–16, 1986), pp. 376–380.
[43] M. Kuperstein, "Adaptive visual-motor coordination in multijoint robots using parallel architecture," in *Proc. 1987 IEEE Int. Conf. on Robotics and Automation* (Raleigh, NC, Mar. 31–Apr. 3, 1987), pp. 1595–1602.
[44] A. Sideris, A. Yamamura, and D. Psaltis, "Dynamical neural networks and their application to robot control," in *IEEE Conf. on Neural Information Processing Systems-Natural and Synthetic* (Denver, CO, Nov. 8–12, 1987), p. 29.
[45] B. W. Mel, "MURPHY: A robot that learns by doing," in *AIP Proc. 1987 Neural Information Processing System Conf.* (Denver, CO, 1987), pp. 544–553.
[46] D. Bullock and S. Grossberg, "A neural network architecture for automatic trajectory formation and coordination of multiple effections during variable-speed arm movements," in *Proc. ICNN* (San Diego, CA, 1987), pp. 559–566.
[47] M. Kawato, K. Furukawa, and R. Suzuki, "A hierarchical neural-network model for control and learning of voluntary movement," *Biol. Cybern.*, vol. 57, pp. 169–185, 1987.
[48] A. Guez and J. Selinsky, "A trainable neuromorphic controller," *J. Robotic Syst.*, vol. 5, pp. 363–388, 1988.
[49] M. Kuperstein, "Neural model of adaptive hand-eye coordination for single postures," *Science,* vol. 239, pp. 1308–1311, 1988.
[50] H. Miyamoto, M. Kawato, T. Setoyama, and R. Suzuki, "Feedback-error learning neural network for trajectory control of a robotic manipulation," *Neural Networks,* vol. 1, pp. 251–265, 1988.
[51] J. S. Albus, "Theoretical and experimental aspects of a cerebellar model," Ph.D. dissertation, University of Maryland, College Park, Dec. 1972.
[52] ——, "A new approach to manipulator control: The cerebellar model articulation control (CMAC)," *Trans. ASME, J. Dyn. Syst., Meas. Contr.*, vol. 97, pp. 220–227, Sept. 1975.
[53] ——, "Mechanisms of planning and problem solving in the brain," *Math. Biosc.*, vol. 45, pp. 247–293, Aug. 1979.
[54] ——, *Brain, Behavior, and Robotics.* Peterborough, NH: BYTE Books, 1981, pp. 139–179.
[55] W. T. Miller, F. H. Glanz, and L. G. Kraft, "Application of a general learning algorithm to the control of robotic manipulators," *Int. J. Robotics Res.*, vol. 6.2, pp. 84–98, Summer, 1987.
[56] W. T. Miller, "A nonlinear learning controller for robotic manipulators," in *Proc. SPIE: Intell. Robots Comput. Vision*, vol. 726, pp. 416–423, Oct. 1986.
[57] ——, "A learning controller for nonrepetitive robotic operations," in *Proc. Workshop on Space Telerobotics* (Pasadena, CA, Jan. 19–22, 1987) (JPL Publ. 87-13), vol. II, pp. 273–281.
[58] ——, "Sensor based control of robotic manipulators using a general learning algorithm," *IEEE J. Robotics Automat.*, vol. RA-3, pp. 157–165, Apr. 1987.
[59] W. T. Miller and R. P. Hewes, "Real time experiments in neural network based learning control during high speed nonrepetitive robot operations," in *Proc. 3rd IEEE Int. Symp. on Intelligent Control* (Washington, DC, Aug. 1988), pp. 24–26.
[60] B. Widrow and M. E. Hoff, "Adaptive switching circuits," in *1960 Wescon Conv. Rec.*, pp. 96–104, 1960.
[61] B. Widrow and S. Stearns, *Adaptive Signal Processing.* Englewood Cliffs, NJ: Prentice-Hall, 1985.

W. Thomas Miller, III (M'79) received the B.S. degree in electrical engineering and the M.S. and Ph.D. degrees in bioengineering all from the Pennsylvania State University, University Park, in 1972, 1974, and 1977, respectively.

He served for two years as a Postdoctoral Fellow in biomedical engineering at the Duke Medical Center, and is currently an Associate Professor of Electrical and Computer Engineering at the University of New Hampshire, Durham. During his graduate and postgraduate studies he specialized in bioelectric phenomena, including computer modeling, real-time processing, and automatic interpretation of bioelectric signals. His current research is focused on the design of neural network architectures for problems in control, pattern recognition, and signal processing.

Robert P. Hewes (S'88) received the B.S. and M.S. degrees in electrical engineering from the University of New Hampshire, Durham, in 1986 and 1988, respectively. He is currently pursuing a doctoral degree in Engineering Sciences at Harvard University, Cambridge, MA.

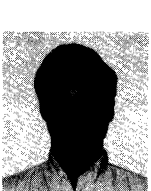
His research interests include control system theory and robotics.

Mr. Hewes is an E.I.T. and a member of Tau Beta Pi and Sigma Xi.

Filson H. Glanz (S'59–M'70) received the B.S. degree in mathematics, the M.S. degree in engineering mechanics, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 1956, 1957, and 1965, respectively.

He has been with the Department of Electrical and Computer Engineering at the University of New Hampshire, Durham, since 1965. He has worked for Librascope, Inc., Stanford Research Institute, and spent a year working on a project modeling the Swedish Coniferous Forests. His professional interests include digital signal processing, nonuniform sampling, robot control, and learning systems.

Dr. Glanz is a member of Sigma Xi and Phi Kappa Phi.



L. Gordon Kraft, III received the Ph.D. degree from the University of Connecticut, Storrs, in 1977, specializing in convergence properties of adaptive control systems.

He has industrial experience at United Technologies and MIT Lincoln Laboratories, as well as ten years of university level teaching of control system concepts from classical servomechanisms to modern optimal control/estimation theory. He is currently as Associate Professor of Electrical and Computer Engineering at the University of New Hampshire, Durham. His current research interests include comparisons of neural network based controllers to more traditional adaptive systems such as self-tuning regulators and Lapunov model reference control systems.

BIBLIOGRAPHY

TMS320 Bibliography

Since the TMS32010 was disclosed in 1982, the TMS320 family has received an ever-increasing amount of recognition. The number of outside parties contributing to the extensive development support offered by Texas Instruments is rapidly growing. Many technical articles are being written about TMS320 applications in the field of digital signal processing.

The following articles and papers have been published since 1982 regarding the Texas Instruments TMS320 Digital Signal Processors. They are divided into one of three application categories and, within each category, are listed in reverse chronological order (most recent first). Those having the same publication date are shown in alphabetical order by author's last name. The application categories are:

- Automotive
- Control
- Industrial

Readers who are interested in gaining further information about the TMS320 processors and applications may obtain copies of the articles/papers from their local or university library.

Automotive

1) K.E. Beck, M.M. Hahn, "A Real-Time Combustion Analysis Instrument," *SAE Technical Paper Series*, USA, February/March 1988.
2) M. Payne, "Do Not Disturb: Lotus in Action. (Lotus Racing Cars Use of an Active Suspension System)," *Electronics Weekly*, USA, i20 1394, page 12, January 1988.
3) C.M. Anastasia, G.W. Pestana, "A Cylinder Pressure Sensor for Closed Loop Engine Control," *SAE Technical Paper Series*, February 1987.

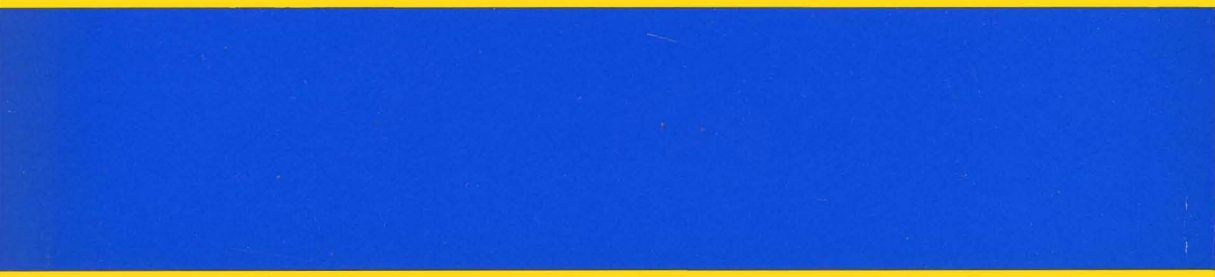
Control

1) I. Ahmed, "16-Bit DSP Microcontroller Fits Motion Control System Application," *PCIM*, October 1988.
2) D. Bursky, "Merged Resources Solve Control Headaches," *Electronic Design*, USA, pages 157–159, October 1988.
3) I. Ahmed, "Implementation of Self Tuning Regulators with TMS320 Family of Digital Signal Processors," *MOTORCON '88*, pages 248–262, September 1988.
4) D. Dunnion, M. Stropoli, "Design a Hard-Disk Controller with DSP Techniques," *Electronic Design*, USA, pages 117–121, September 1988.
5) S.W. Yates, R.D. Williams, "A Fault Tolerant Multiprocessor Controller For Magnetic Bearings," *IEEE Micro*, USA, Volume 8, Number 4, page 6, August 1988.
6) Y.V.V.S. Murty, W.J. Smolinski, S. Sivakumar, "Design of a Digital Protection Scheme For Power Transformers Using Optimal State Observers," *IEE Proc. C, Generation Transmission, Distribution*, Great Britain, Volume 135, Number 3, pages 224–230, May 1988.
7) R.D. Jackson, D.S. Wijesundera, "Direct Digital Control of Induction Motor Currents," *IEE Colloquim on Microcomputer Instrumentation and Control Systems in Power Electronics*, Great Britain, Digest Number 61, 1/1–3, April 1988.

8) A. Lovrich, G. Troullinos, R. Chirayil, "An All Digital Automatic Gain Control," *Proceedings of ICASSP '88*, USA, Volume D, page 1734, April 1988.
9) K. Bala, "Running on Imbedded Power," *Electronics Engineering Times*, USA, March 1988.
10) I. Ahmed, S. Meshkat, "Using DSPs in Control," *Control Engineering*, February 1988.
11) M. Babb (Editor), "Solving Control Problems with Specialized Processors," *Control Engineering*, February 1988.
12) S. Meshkat, "High-Level Motion Control Programming Using DSPs," *Control Engineering*, February 1988.
13) I. Ahmed, "DSP Architectures for Digital Control Systems," *SATECH 1988*, 1988.
14) S. Meskat, "Advanced Motion Control Systems," *Intertec Communications*, Ventura, CA., 1988.
15) I. Ahmed, S. Lundquist, "DSPs Tame Adaptive Control," *Machine Design*, USA, Volume 59, Number 28, pages 125–129, November 1987.
16) Y. Dote, M. Shinojima, R.G. Hoft, "Digital Signal Processor (DSP)-Based Novel Variable Structure Control For Robotic Manipulator," *Proceedings of IECON '87*, Volume 1, pages 175–179, November 1987.
17) I. Ahmed, "Deadbeat Controllers and Observers with the TMS320," *MOTORCON '87*, pages 22–33, September 1987.
18) R.D. Ciskowski, C.H. Liu, H.H. Ottesen, S.U. Rahman, "System Identification: An Experimental Verification," *IBM Journal of Research Developments*, Volume 31, Number 5, pages 571–584, September 1987.
19) E. Debourse, "Emergence of DSPs in Machine-Tool Axes Control Systems: Application of Distributed Interpolation Concepts," *Proceedings of the International Workshop on Industrial Automation*, February 1987.
20) C. Chen, "The Mathematical Model and Computer Simulation of an LCI Drive," *Electrical Machinery Power Systems*, USA, Volume 13, Number 3, pages 195–206, 1987.
21) M.C. Stich, "Digital Servo Algorithm For Disk Actuator Control," *Conference on Applied Motion Control*, pages 35–41, 1987.
22) T. Takeshita, K. Kameda, H. Ohashi, N. Matsui, "Digital Signal Processor Based High Speed Current Control of Brushless Motor," *Electronic Engineering*, Japan, USA, Volume 106, Number 6, pages 42–49, November/December 1986.
23) R. Alcantara, J. Prado, C Guegen, "Fixed-Point Implementation of the Fast Kalman Algorithm: Using the TMS32010 Microprocessor," *Proceedings of EUSIPCO '86*, Volume 2, pages 1335–1338, September 1986.
24) B. Nowrouzian, M.H. Hamza, "DC Motor Control Using a Switched-Capacitor Circuit," *Proceedings of the IASTED International Symposium on High Technology in the Power Industry*, pages 352–356, August 1986.
25) N. Matsui, T. Takeshita, "Digital Signal Processor-Based Controllers For Motors," *SICE*, July 1986.
26) R. Cushman, "Easy-to-Use DSP Converter ICs Simplify Industrial-Control Tasks," *Electronic Design*, USA, Volume 29, Number 17, pages 218–228, August 1984.
27) W. Loges, "Signal Processor as High-Speed Digital Controller," *Elektronik Industrie*, Germany, Volume 15, Number 5, pages 30–32, 1984.
28) W. Loges, "Higher-Order Control Systems with Signal Processor TMS320," *Elektronik Industrie*, Germany, Volume 32, Number 25, pages 53–55, December 1983.

Industrial

1) D.E. Luttrell, T.A. Dow, "Control of Precise Positioning System with Cascaded Colinear Actuators," *American Control Conference*, pages 121–126, June 1988.
2) M. Ruscio, M. Santoro, M. Adorni, A. Chiaravalloti, "Digital Control System For The Coordinated Boiler/Turbine Control in the ENEL Piombino Power Station," *Elettrotencia*, Italy, Volume 75, Number 3, pages 253–258, March 1988.
3) J.A. Taufiq, R.J. Chance, C.J. Goodman, "On-Line Implementation of Optimised PWM Schemes For Traction Inverter Drives," *International Conference of 'Electric Railway Systems For a New Century'*, Great Britain, Conference Publication Number 279, pages 63–67, September 1987.
4) Y. Dote, M. Shinojima, H. Hoshimura, "Microprocessor-Based Novel Variable Structure Control for Robot Manipulator," *Proceedings of the 10th IFAC World Congress*, July 1987.
5) H. Henrichfriese, W. Moritz, H. Siemensmeyer, "Control of a Light, Elastic Manipulation Device," *Conference on Applied Motion Control*, pages 57–66, 1987.
6) Y. Wang, M. Andrews, S. Butner, G. Beni, "Robot-Controller System," *15th Annual Symposium on Incremental Motion Control Systems & Devices*, pages 17–26, June 1986.
7) R. Cushman, "Easy-to-Use DSP Converter ICs Simplify Industrial-Control Tasks," *Electronic Design*, USA, Volume 29, Number 17, pages 218–228, August 1984.
8) P. Rojek and W. Wetzel, "Multiprocessor Concept for Industrial Robots: Multivariable Control with Signal Processors," *Elektronik Industrie*, Germany, Volume 33, Number 16, pages 109–113, August 1984.
9) G. Farber, "Microelectronics-Developmental Trends and Effects on Automation Techniques," *Regelungstechnik Praxis*, Germany, Volume 24, Number 10, pages 326–336, October 1982.



TEXAS INSTRUMENTS

Printed in U.S.A., May 1991

SPRA019