Implementation of PID and Deadbeat Controllers with the TMS320 Family

APPLICATION REPORT: SPRA083

Irfan Ahmed
Digital Signal Processor Products
Semiconductor Group
Texas Instruments

Digital Signal Processing Solutions

IMPORTANT NOTICE

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

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale 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. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain application using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any 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.

TRADEMARKS

TI is a trademark of Texas Instruments Incorporated.

Other brands and names are the property of their respective owners.

CONTACT INFORMATION

US TMS320 HOTLINE	(281) 274-2320
US TMS320 FAX	(281) 274-2324
US TMS320 BBS	(281) 274-2323
US TMS320 email	dsph@ti.com

Implementation of PID and Deadbeat Controllers with the TMS320 Family

Abstract

This report discusses implementation of the PID and deadbeat controllers with the TMS320 family of DSPs. These application-specific processors are designed to process signals, including control signals, very efficiently.

The report covers the following topics:

- Control Systems
  - Analog Controllers
  - Digital Controllers
  - Analog versus Digital Controllers
- Processor Selection Issues
  - DSP Architectures
- Design of Digital Control Systems
  - Discretization of Analog Systems
  - Plant Modeling
  - Digital Controller design
  - Design and Implementation of PID Controllers
  - Deadbeat

- Implementing Digital Controllers
  - Finite Wordlength Effects
  - Fixed-Point versus Floating-Point Arithmetic Processors
  - Sampling Rate Selection
  - Controller Design Tools
  - Hardware Design
- Applications
  - Computer Peripherals
  - Power Electronics
  - Automotive
- Summary and References

The report also includes the following appendixes:

- Appendix A Plant Modeling
- Appendix B PID Controller
- Appendix C Deadbeat Controller
- Appendix D PC-Matlab Design and Display Programs
- Appendix E TMS320C15 Assembly Code

Product Support

World Wide Web

Our World Wide Web site at www.ti.com contains the most up to date product information, revisions, and additions. New users must register with TI&ME before they can access the data sheet archive. TI&ME allows users to build custom information pages and receive new product updates automatically via email.

Email

For technical issues or clarification on switching products, please send a detailed email to dsph@ti.com. Questions receive prompt attention and are usually answered within one business day.

Introduction

Control systems are a necessary part of modern manufacturing, industrial processes, and our daily lives. They range from simple controls like those on our air conditioning to more intricate controls like those for a missile guidance system. Control mechanisms have evolved from mechanical, pneumatic, and electromechanical systems to electronic control systems. Electronic controls have been implemented with analog components like resistors, capacitors and op-amps (operational amplifiers). However, with the availability of microprocessors, control systems are being implemented in digital form. The use of microprocessors in digital control systems has created not only some new opportunities due to the powerful processing capabilities of microprocessors, but also a need for a new body of knowledge that utilizes some of these processing capabilities.

This report discusses implementation of PID (proportional integral derivative) and deadbeat digital controllers with the TMS320 family of digital signal processors. These digital signal processors are application-specific processors designed to process signals, including control signals, very efficiently. In numerically intensive applications, digital signal processors are at least an order of magnitude higher in performance than conventional processors and minimize the numerical problems of processing signals digitally.

This report is arranged in the following order:

- *Control Systems* – Provides an introduction to digital controllers and discusses selection of processors for a digital controller.
- *Design of Digital Control Systems* – Discusses the design of digital controllers.
- *Implementing Digital Controllers* – Discusses implementation of digital controllers.
- *Applications* – Describes applications of digital controllers using the TMS320 digital signal processors.
- *Appendices A* through *C* – Lists the mathematical procedures needed to design the controllers.
- *Appendix D* – Lists the PC-Matlab programs used for simulation of these controllers.
- *Appendix E* – Lists the TMS320C15 assembly code required to implement the controller algorithms.

Control Systems

A control system is a system that commands or regulates a process in order to achieve a desired output from the process. A control system consists of three main components:

- Sensors
- Actuators
- A controller

Sensors measure the behavior of the system or the process and provide feedback information. Typical sensors used in control systems are resolvers, shaft encoders, and potentiometers that provide position information; tachometers that provide speed information, and current sensors that provide current information. Actuators supply the driving and corrective forces to achieve a desired output. Typical actuators are DC and AC motors in electromechnical systems, and valves in hydraulic or pneumatic systems.

The controller generates actuator commands in response to the commands received from the system controller and in response to feedback information provided by the sensors. The controller consists of some computation elements that process the command and feedback 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, 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 can be implemented with analog or digital components.

Analog Controllers

Control systems have traditionally been implemented using analog components like operational amplifiers, resistors, and capacitors. The combination of these elements implements filter-like structures that modify the frequency response of the system. Although more powerful analog processing elements like multipliers are available, they are generally not used because of their high cost. In spite of their simpler processing elements, analog controllers can be used to implement high-performance systems.

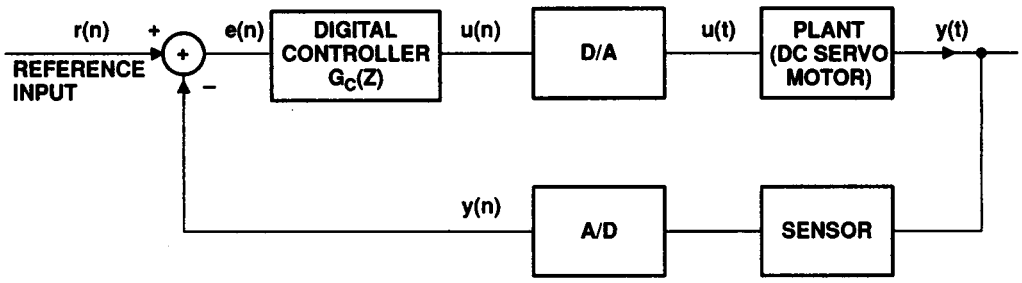
Digital Control Systems

With the performance and reliability inherent to microprocessors and microcontrollers, digital controllers are taking over many applications from analog controllers. In a digital control system, the controller is implemented with a microprocessor or a microcontroller, which is responsible for processing the signal. However, the actuator commands from the controller are digital and may have to be converted into analog signals by a D/A (digital-to-analog) converter. Similarly, the measurements from the sensor may be analog and will have to be converted into a digital signal by an A/D (analog-to-digital) converter.

Figure 1 shows the block diagram of a digital control system. The D/A converter converts the digital output of the microcomputer, u(n), into an analog signal, u(t). The output, u(t), of the D/A needs power amplification and drives the motor to the desired or reference state, r(n). The out-

put of the motor, y(t), is measured by a sensor and converted into a digital signal, y(n), by the A/D. The feedback signal is subtracted from the reference signal r(n) to create an error signal e(n). The error signal, e(n), is used by the controller to issue the corresponding control action u(n).

Figure 1. Digital Control System



Analog Versus Digital Controllers

Tradeoffs have to be made in the selection of controllers for a system. Analog controllers provide continous processing of the signal and can be used for very high bandwidth systems. They also give almost infinite resolution of the signal they are measuring, thus providing precise control. Analog controllers have been around for a long time; their behavior is well understood, and this makes them easy to design. They can be implemented with relatively inexpensive components and therefore are sometimes cheaper.

On the negative side, analog controllers suffer from component aging and temperature drifts. Thus, a perfectly designed controller will start to exhibit undesired characteristics after a while. Analog controllers are hardwired solutions; this makes modifications or upgrades in the design difficult. Analog controllers are also limited to simpler algorithms from classical control theory like PID and compensation techniques.

Digital controllers sample the signal at discrete time intervals. This limits the bandwidth (bandwidth is 1/6 to 1/10 sampling rate) that can be handled by the controller. The processing of the signal takes a finite amount of time, adding to phase delay in the system. In addition, the resolution of the signal is limited by the resolution or wordlength of the processor. Digital controllers also require additional components like A/D and D/A; although newer processors include these components on the same chip. Digital controllers are relatively new, and their behavior is not very well understood, thus making design of digital controllers relatively difficult in comparison to analog controllers.

However, digital controllers have some advantages also. They are not affected by component aging and temperature drift, and they provide stable performance. When the design is done in the z-domain, the behavior of digital controllers can be more precisely controlled. They can also be used to implement more sophisticated techniques from modern control theory, such as state controllers, optimal control, and adaptive control. Digital controllers are programmable, thus making them easy to upgrade and maintaining design investment. They can also be time shared to implement different functions in the system, like notch filters and system control, thus reducing system

cost. If digital controllers are designed properly, their advantages outweigh their disadvantages. Table 1 compares analog and digital controllers.

Table 1. Comparison of Analog versus Digital Controllers

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

Processor Selection Issues

The choice of processor is critical in determing the performance and behavior of the digital controller. The usual choices are microcontrollers, general purpose microprocessors, and digital signal processors (DSP). RISC processors and bit-slice processors can also be used; however it is not practical to use them in most cases because of their cost.

Digital controllers monitor signals at discrete time intervals or finite sampling rates. If the signal is not sampled fast enough, some of the information may be lost. The processing of the signal takes a finite amount of time. The processing has to be completed before the arrival of the next sample, or preferably as soon as possible. Too much delay in the output can cause loss of information or excessive phase delay in the system, leading to instability. These conditions impose certain minimum performance requirements on the processor. Most of the processors currently used to implement controllers are usually not fast enough to process signals in real time; they rely upon lookup tables with precomputed results.

Digital controllers use discrete steps to represent a signal, which is limited to the wordlength of the processor. Coefficients or gain constants also have to be represented in the limited wordlength. This discretization or loss of resolution is referred to as quantization error. In addition, results of mathematical operations have to fit in a limited wordlength and may lose part of the result due to this limitation. This is referred to as truncation error. Both of these errors cause oscillations or limit cycles and can lead to instability.

Another problem that occurs in digital controllers is overflow of registers. Successive mathematical operations can cause registers to overflow. Registers in most processors wrap around, causing the result of a calculation to go from most positive to least negative, in turn causing the output to reverse directions. Most of these problems occur in microcontrollers and microprocessors because their architectures are not designed for signal processing.

Early microprocessors (μP) and microcontrollers (μC) were designed to replace hardwired logic, TTL gates, etc. Newer microprocessors and microcontrollers have retained most of the old

architectures. These processors are adequate for simple applications that require little or no signal processing. In a control system, the controller is responsible for processing the command and feedback signal. Thus, applications such as control systems, speech, and telecommunications require intensive numeric processing of analog signals. μPs and μCs are usually unable to do the processing correctly, so using them can cause significant numerical problems.

Most of these problems occur in processors that have 8/16-bit ALUs and registers. This 8/16-bit architecture limits the accuracy of intermediate and final results and generates truncation/quantization errors. Lack of scaling shifters to maintain the required significant bits can cause additional quantization/truncation errors. Most processors also lack the performance to perform real time processing, so they rely upon lookup tables, thus limiting precision to low-performance or low-bandwidth systems. Lack of processing capability also limits these processors to simpler control techniques. They are unable to take advantage of sophisticated techniques from modern control theory. If used in higher-performance systems, they can cause excessive loop delays, leading to instability. Most of the problems discussed can be eliminated with the use of digital signal processors as controllers.

DSP Architectures

DSP architectures like those in the TMS320 family have been designed for signal processing systems. The TMS320 family not only has an architecture that minimizes numerical problems in signal processing, but also has the performance to meet the bandwidth requirements of high performance systems using sophisticated techniques.

DSP architectures are optimized to give the highest possible performance. To achieve high processing speed, the TMS320 DSPs perform all functions in internally hardwired logic. Thus, it takes a single clock cycle to execute most functions. Other processors perform the same functions in software or microcode, thus taking a large number of cycles for execution. To enhance the performance even further, the TMS320 architecture employs a multiple-bus internal architecture. This allows simultaneous fetch of instructions and data operands. Most instructions on the TMS320, including arithmetic operations, are executed in a single clock cycle.

In digital signal processing, most algorithms, including control algorithms, can be represented as difference equations consisting of multiply accumulates. The TMS320 DSPs contain a hardware multiplier that performs a 16×16 multiply in a single instruction cycle. This high speed allows the TMS320 to execute most control algorithms in real time. The fast processing speed minimizes the computation delay time for generating the output from the controller and also allows very fast sampling rates to be implemented for high bandwidth systems. Additional features in the TMS320 architecture include an instruction set that is optimized for data sampled systems. The DMOV instruction implements the z^{-1} operator. The MACD instruction implements four operations simultaneously:

- A multiply
- Data move
- Accumulate previous result
- Load T register

For greater precision, the TMS320 has 32-bit registers for storing intermediate results. In addition, the TMS320 has multiple hardware parallel shifters to allow scaling and prevent overflows. These shifters enable shifting to take place simultaneously with other operations without additional overhead or execution time. The 32-bit registers and shifters minimize quantization and truncation errors because a very high precision can be maintained both for intermediate and final results. The TMS320 also contains an overflow mode, which, in case of overflow, 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. For fast context saves, the TMS320 contains an on-chip hardware stack, reducing interrupt response time and minimizing stack pointer manipulations. Since the TMS320 is a family of microcontrollers, it also minimizes system costs with features such as on-chip RAM, on-chip ROM/EPROM, and on-chip peripherals like serial ports, timers, and parallel I/O. The high degree of on-chip functionality, flexible instruction set, pipelined architecture, and high performance make the TMS320 the preferred choice in most control and signal processing applications. Table 2 lists a comparison between the TMS320C14, TMS320C25, and several μCs and μPs.

Table 2. Features Comparison

Feature	'320C14	'320C25	80C196	68000	68020
Instruction cycle time-ns	160	100	333	400	120
Frequency – Mhz	25	40	12	10	24
Multiply 16×16 → 32 (μs)	0.16	0.1	2.2	7.0	1.0
PID loop (μs)	2.2	1.3	27.0	25.0	4.8
Matrix multiply (3×3, 1×3 — μs)	4.3	2.7	24.3	65.2	9.7

TMS320 Digital Signal Processors

The TMS320 digital signals processors can be divided into two families: a fixed-point arithmetic family and a floating-point arithmetic family. Each family is further divided into different generations and offers different performance ranges between generations. Within each generation, members are object code and, in some cases, pin compatible.

TMS320 Family of Fixed-Point Arithmetic DSPs

The fixed-point arithmetic family is made up of three generations, TMS320C1x, TMS320C2x, and TMS320C5x. All members of the fixed-point arithmetic family have a 16-bit architecture with 32-bit ALU and accumulator. They are based on a Harvard architecture with separate buses for program and data, allowing both instructions and operands to be fetched simultaneously. They also feature a 16 × 16=32 hardware multiplier for a single-cycle multiply, and a hardware stack for fast context saves. An overflow saturation mode is included to prevent wraparound. All instructions (except branches) are executed in a single clock cycle. Performance ranges from 5 MIPS (million of instructions per second) to 28.5 MIPS among the three generations.

TMS320C1x

The TMS320C1x generation is based on the TMS32010, the first DSP, introduced in 1982. It comes with 144 words of on-chip RAM and 4K-words of address space. Instruction cycle time goes down to 160 ns. Other members of the first generation include the TMS320C15 and its

EPROM version (TMS320E15), TMS320C17/E17, and TMS320C14/E14. All of these devices have an expanded memory of 256 words of on-chip RAM and 4K-words of on-chip ROM/ EPROM. The TMS320C14/E14 is optimized for digital control applications and has 16 pins of bit I/O, four timers (including a watchdog timer), a USART, 6/4 channels of pulse width modulation (PWM), and 2/4 capture inputs for optical encoder interface and PWM[1,2].

TMS320C2x

The TMS320C2x generation is based on the TMS320C25. It comes with 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 is considerably enhanced from the C1x generation. Instruction cycle time is reduced to 80 ns. Other members include the TMS320E25, TMS32020, and TMS320C26[3].

TMS320C5x

The TMS320C5x generation is based on the TMS320C50. It features 8K-words of on-chip RAM and 2K-words of on-chip ROM. The instruction set is considerably enhanced from the TMS320C2x generation. Some of the new features include a separate PLU, shadow registers for fast context save, JTAG serial scan emulation, and software wait states. Instruction cycle time is 35 ns.

TMS320 Family of Floating-Point Arithmetic DSPs

The floating-point arithmetic DSPs consist of two generations: the TMS320C3x and the TMS320C4x. All members of the floating-point arithmetic family have a 32-bit architecture with 40-bit extended precision registers. The floating-point arithmetic family is based on Von Neuman architecture. However, multiple buses are included to give even faster throughput than traditional Harvard architectures. Features include hardware floating-point multiplier and a floating-point ALU.

TMS320C3x

The TMS320C3x family is based on the TMS320C30. It features 2K × 32 words of on-chip RAM, 4K × 32 of on-chip ROM, and 64 words instruction cache. Other features include a separate DMA, two serial ports and two timers. The TMS320C30 features two external 32-bit data buses and a 16M-word address space. Instruction cycle time is 60 ns, and performance is up to 33 MFLOPS (million floating-point operations/second)[4].

Design of Digital Control Systems

Design of a control system involves two major steps:

1) The process or plant has to be put into mathematical form so that its behavior can be analyzed and evaluated (i.e., a plant model has to be derived).
2) An appropriate controller must be designed so that the plant gives the desired response under influence of the control system.

Designing a digital control system requires additional steps that convert the system into a discrete form. This conversion can be done in two different ways.

1) The design of the controller can be carried out entirely in the analog form in s-domain, and then converted into discrete form at the final stage for implementation.
2) The design of the controller may also be carried out entirely in discrete form. In this case, the plant model has to converted into discrete form or z-domain.

This section describes how to

- Discretize analog systems
- Reduce a plant into a mathematical form
- Design the controller.

Discretization of Analog Systems

There are several ways to convert existing continuous or analog control systems into discrete or digital systems. However, the conversion from the s-domain to the z-domain causes some distortion in the location of the poles in the z-domain, and must therefore be taken into account.

Zero-Order Hold (ZOH)

This technique assumes that the controller 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:

$$H(z) = (1 - z^{-1})Z\,[L^{-1}(H(s)/s)] \quad (1)$$

It is assumed that the Laplace transform will be split up using partial fractions and z-transform tables will be used.

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.

Bilinear Transformation

This technique, also called the Tustin or trapezoidal approximation, uses the relationship

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

to transform an s-domain function into the z-domain. The left half of the s-plane is mapped into the unit circle in the z-plane. However, this method warps the frequency response at the critical frequencies of the system. To overcome this problem, the critical frequencies of the original s-domain are prewarped so that the critical frequencies of the z-domain system end up where they belong. The critical frequency W_0 is prewarped to another frequency by the transformation

$$W^* = \frac{2}{T}\tan\left(W_o\frac{T}{2}\right) \tag{3}$$

where T is the sampling period.

Before these techniques can be used, an appropriate plant transfer function must be derived, or a controller may have to be designed. The next section describes derivation of a plant transfer function and the controller design.

Plant Modeling

The first part of designing any control system is to describe the plant in a mathematical form or identify the plant's parameters. This section describes the derivation of a mathematical model for a plant.

A DC servo motor is used in the example, 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 the z-domain for use with a digital controller. The ZOH method is used to transform the model into a discrete form.

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

$$L\frac{di}{dt} + Ri = V - emf \tag{4}$$

where

L = inductance of motor
R = resistance
V = applied voltage
i = current
emf = back emf = $K_e \times \theta$

The mechanical characteristics are given by

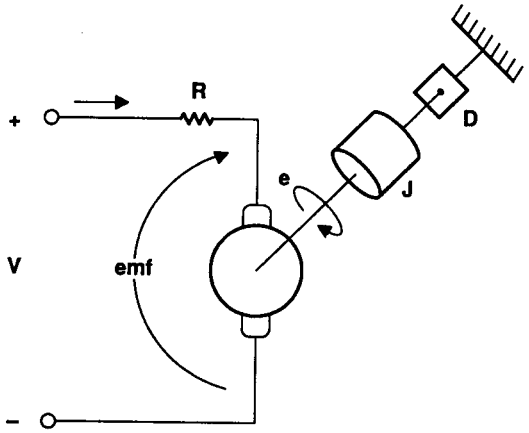
$$J_M\frac{d^2\theta}{d_t^2} + B\frac{d\theta}{d_t} + K\theta = T_L - J_L\frac{d^2\theta}{dt^2} \tag{5}$$

where

J_M = motor inertia
θ = angular displacement
K = stiffness constant
B = damping constant
J_L = load inertia
T_L = load torque = $K_t \times i$
K_t = torque constant

Figure 2 shows a representation of the equivalent electrical and mechanical model of a DC servo motor.

Figure 2. Model of DC Motor



The motor selected for this report is a Pittman model 9412G316. It has the following parameters:

R = 6.4 Ω
J = 1.54×10^{-6} kg × m^2
K_t = 0.0207 (N × m)/A
K_e = 0.0206 volt/(rad/s)

The electrical and mechanical characteristics of the motor given by (4) and (5) can be combined. After the values of the motor parameters are substituted, the complete transfer function for the mathematical model of the motor as derived in Appendix A can be stated as

$$G_m(s) = \frac{53.906}{s\ (s + 1.116)} \tag{6}$$

This transfer function is then transformed into the z-domain using the zero-hold approximation as shown in Appendix A. The transfer function or the model in the discrete form is stated as

$$G_m(z) = \frac{.2694z^{-1} + .2693z^{-2}}{1 - 1.999z^{-1} + .999z^{-2}} \times K_m \tag{7}$$

where

K_m = the numerator gain constant.

Digital Controller Design

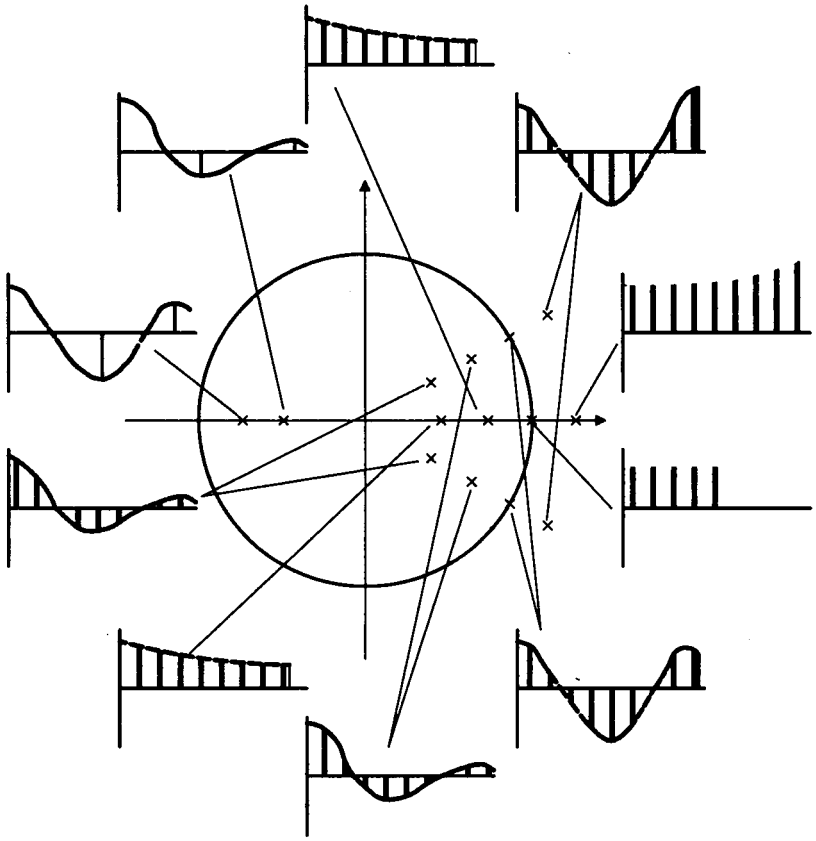
The next step in designing a digital control system is to design the controller. The controller may be designed in the continuous domain and then converted into discrete form. Alternatively,

the entire design may be carried out in the discrete domain. It is assumed that the design is carried out in the discrete domain. The next few sections give an overview of different types of control algorithms and discuss design of a PID controller and a deadbeat controller. However, before design of a digital controller is done, some discussion of behavior of poles in the z-domain is essential. The next section discusses behavior of poles in the z-domain.

Behavior of Poles in Z-domain

Use of the conversion techniques allows the conversion of an existing analog design into a digital design. However, to insure successful implementation of the control system design, some knowledge of the behavior of the poles in the z-domain is essential. 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 manifest oscillatory behavior. As poles move towards the origin, their response decays at a faster rate. Figure 3 shows the different pole locations and their corresponding responses.

Figure 3. Behavior of Poles in Z-Domain



Control Algorithms

The next 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. For a continuous control system, the controller is designed in the s-domain by using 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 (z- domain) by using a 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 a precise modification to system behavior. Implementation of compensation techniques is described in an application report, Control System Compensation and Implementation with the TMS32010[5].

PID

The PID is a 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[6, 7].

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[8].

State Models

In a state 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 where multiple variables or states need to be controlled. State controllers are sometimes impractical to implement because it may not be possible to measure all states. They are usually used in conjunction with observers. State controllers allow very precise control of system behavior[6, 9, 10].

Observer Models

Often, in control systems, some of the states of the system are not available for measurement. The measurement of known states in an observer model can be used to estimate unknown states in the control system. 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, where access to all state variables may not be available[6, 9, 10, 11].

Optimal Control

Optimal control synthesis is used when a specific performance or cost criteria (e.g., time or energy) must be minimized. The given criteria or cost function is used to derive an appropriate control law, which is then implemented with a controller or compensator[7, 12].

Kalman Filter

An observer model can be 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[7, 13, 14].

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 changes in the plant by redesigning the controller to give an optimum control system[6, 8, 16].

Design and Implementation of PID Controllers

This section describes design and implementation of a 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. Including the integral of the error term reduces the steady-state error to 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. This technique is also being used in discrete form with digital control systems.

Two different approaches are used for conversion of PID into discrete form: rectangular and trapezoidal approximations. For the rectangular approximation, the design is done in the analog domain and then converted into z-domain. For the trapezoidal approximation, the design is done directly in the z-domain, using pole placement techniques.

The analog PID algorithm is given by

$$u(t) = K_p e(t) + K_i \int d_t + K_d \frac{de}{dt} \tag{8}$$

where

K_p, K_i, and K_d = PID constants
$u(t)$ = output of controller
$e(t)$ = error signal

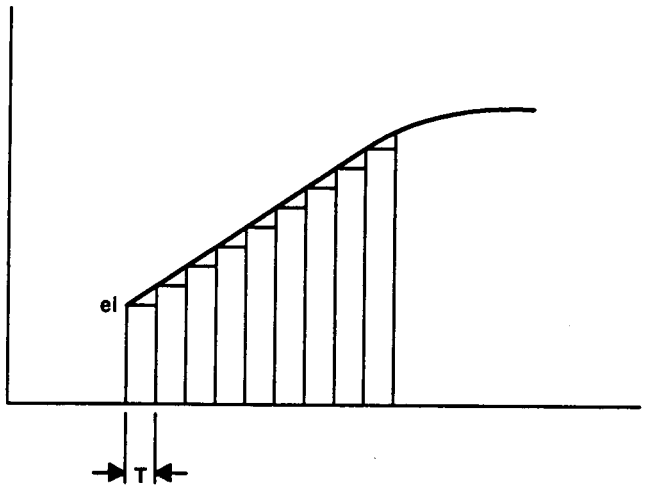
Rectangular Approximation

To discretize, assume that the sampling interval for the system is T. The rectangular approximation is the easiest to use and gives satisfactory results. For the rectangular approximation, assume the integral $\int$ edt is an accumulation of small rectangles given by $T \times \sum e(i)$ for i = 1 to n (see Figure 4). The differential de/dt (if T is sufficiently small) can be approximated by

$$\frac{e(n) - e(n-1)}{T} \tag{9}$$

where e(n) = the nth sample.

Figure 4. Rectangular Approximation



After conversion into its discrete form and transferring into time domain, the final form is shown in (7). The complete derivation is described in Appendix B.

$$u(n) = u(n-2) + K_1 e(n) + K_2 e(n-1) + K_3 e(n-2) \tag{10}$$

where

K_1 = $K_p + K_d/T + K_i T$
K_2 = $K_i T - 2K_d/T$
K_3 = $K_d/T - K_p$
$u(n)$ = control signal at time interval n
$u(n-2)$ = $(n-2)$th sample
$e(n)$ = error signal at time n
$e(n-1)$ = error signal at time $n-1$
T = sampling interval

Appendix E shows the code to implement (7) on the PID controller with rectangular approximation. The code takes 11 instructions on the TMS320C15 and executes in 2.6 μs at 20 MHz. The

MPY instruction performs a multiply in a single cycle. The LTD instruction loads the T register, implements a data shift or z^{-1} operation, and adds the result of the multiply to the previous value in the accumulator, all in a single cycle.

K_1, K_2, and K_3 are obtained by designing K_p, K_i, and K_d and using conventional techniques from classical control[6, 7] (see Appendix B). Figure 7 through Figure 6 show the step response of the PID controller with different values of K_p, K_i, and K_d.

Figure 5. Step Response of the PID Controller with First Set of K_p, K_i, and K_d Values

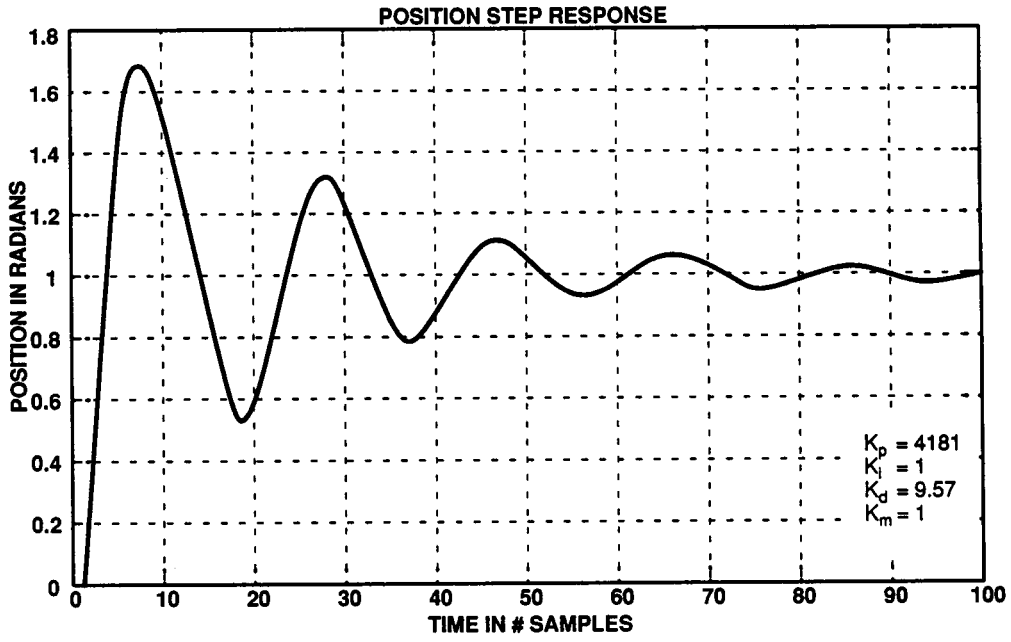


Figure 6. Step Response of the PID Controller with Second Set of K_p, K_i, and K_d Values

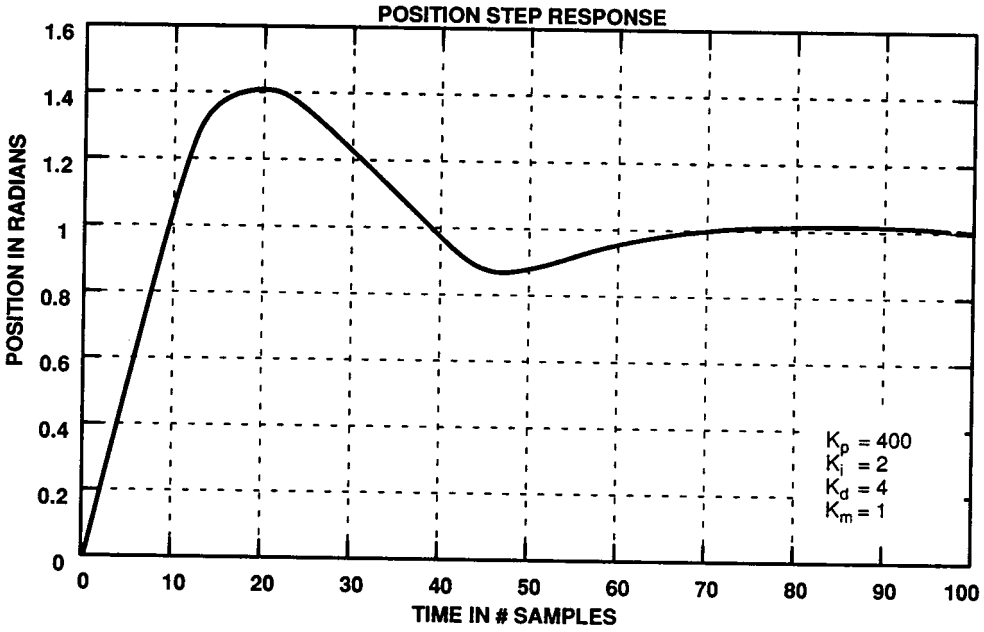
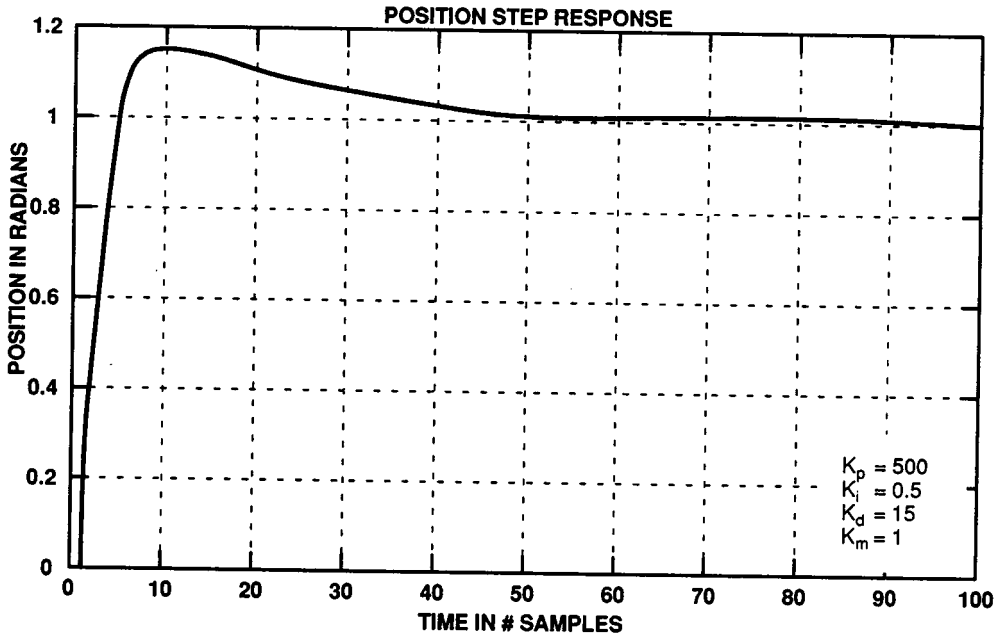


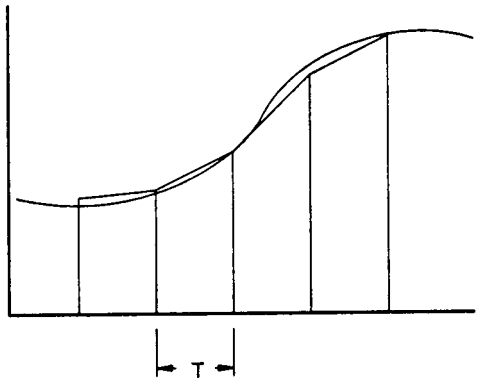
Figure 7. Step Response of the PID Controller with Third Set of K_p, K_i, and K_d Values



Trapezoidal Approximation

If a more accurate conversion is required, the trapezoidal approximation or Tustin transformation is used. The area of the integral $\int edt$ is given by the summation of small trapezoids (see Figure 8).

Figure 8. Trapezoidal Approximation



The integral $\int edt$ can also be solved by taking the Laplace transform of (8) and substituting

$$s = 2/T \times (z-1/z+1) \tag{11}$$

After substitution and solving,

$$u(n) = u(n-2) + K_1e(n) + K_2e(n-1) + K_3e(n-2) \tag{12}$$

where

K_1 $= K_p + 2K_d/T + K_iT/2$
K_2 $= K_iT - 4K_d/T$
K_3 $= 2K_d/T - K_p + K_iT/2$
$u(n)$ = nth sample of output of controller
$u(n–2)$ = $(n–2)$nd sample of output

The complete derivation of this equation is described in Appendix B. The code to implement (12) on the PID controller is shown in Appendix E; it takes 12 instructions and executes in 2.8 μs.

The gain constants K_1, K_2, and K_3 are designed by selecting the poles for the system transfer function[15] (see Appendix B). 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. 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 decreases; the system may become unstable if the poles are selected too near or outside the unit circle. Figure 9 through Figure 11 show the step response of the PID controller for various pole locations for the system.

Figure 9. PID Controller Step Response for System's First Pole Locations

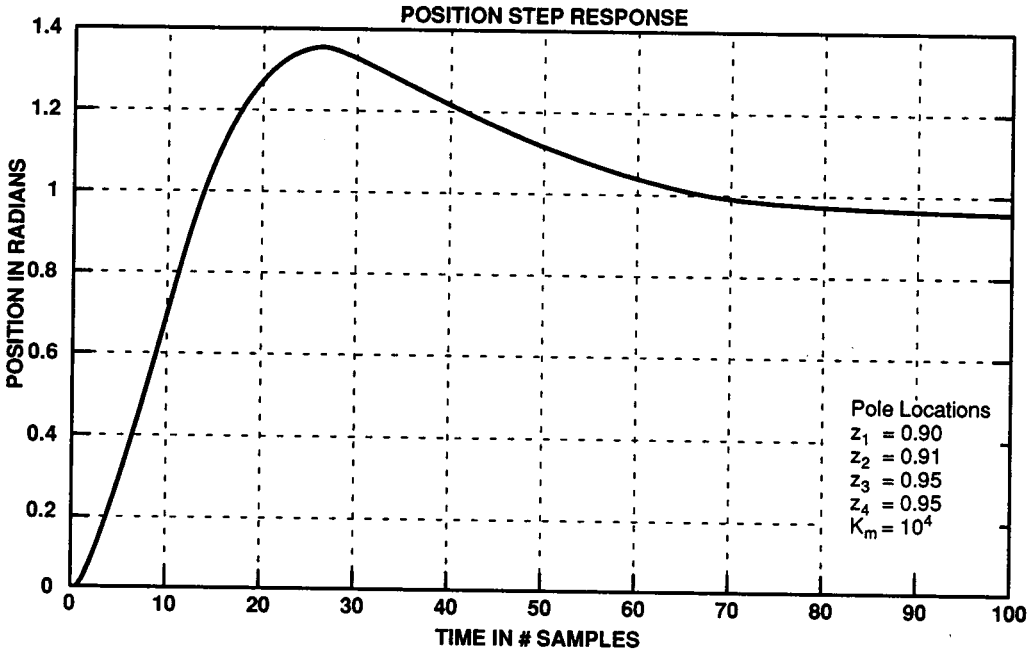


Figure 10. PID Controller Step Response for System's Second Pole Locations

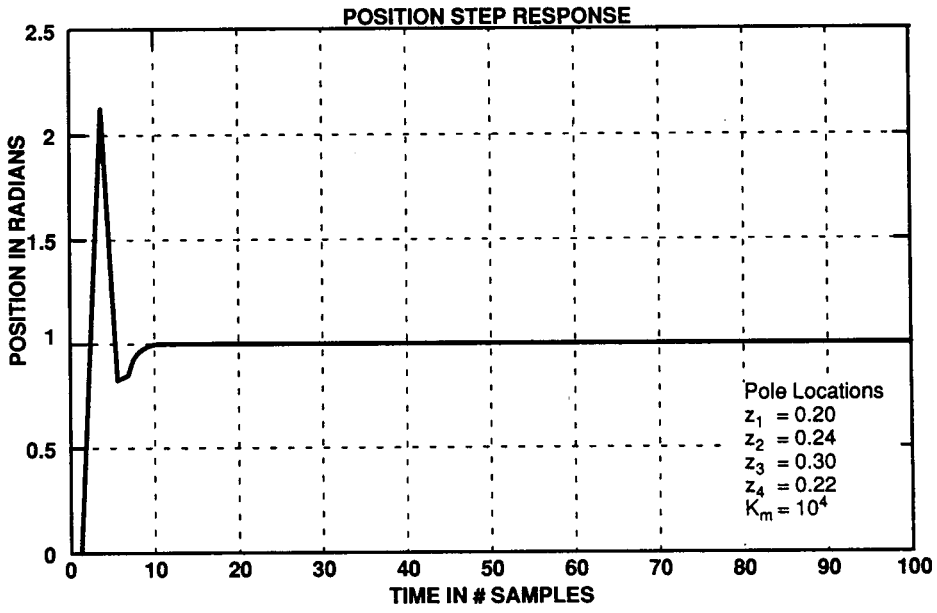
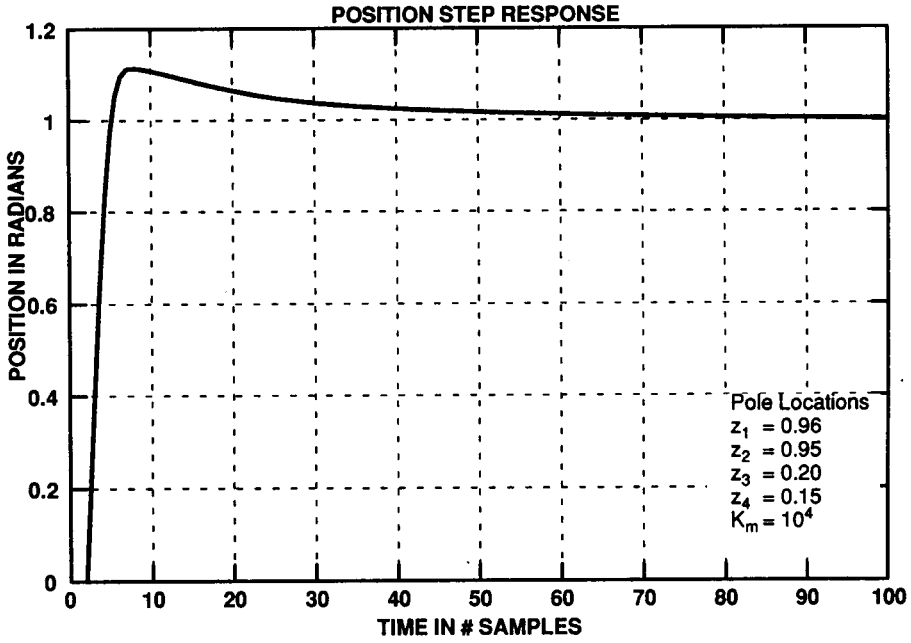


Figure 11. PID Controller Step Response for System's Third Pole Locations

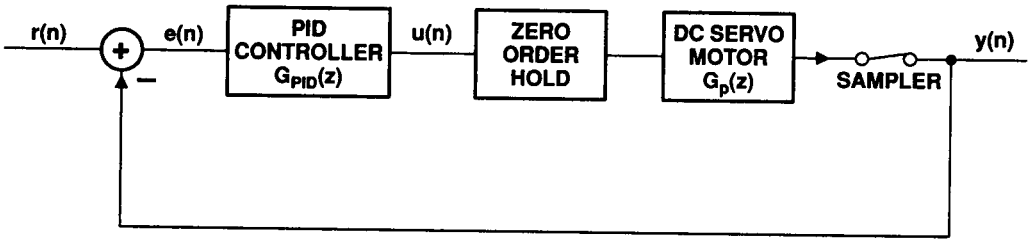


Our final algorithm comes out to

$$u(n) = u(n-2) + 0.162e(n) - 0.439e(n-1) + 0.3185e(n-2) \quad (13)$$

Figure 12 shows the the block diagram of a system using a PID controller. The zero-order hold represents the function of a D/A, and the sampler represents the function of an A/D.

Figure 12. PID Controller System Block Diagram



Deadbeat

One of the desired characteristics in a control system design is a quick settling time. In an analog controller, it takes the system output 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 reaches 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., in adaptive control systems).

Deadbeat controllers compensate for the poles of the system; therefore, they should not be applied to systems with poles outside or in the vicinity of the unit circle in the z-plane. Thus, deadbeat controllers should be used only with stable plants or processes; otherwise they may cause instability. Deadbeat controllers may also require a large amount of gain, thus leading to actuator saturation.

Deadbeat controllers also give a large overshoot. The only design parameter in deadbeat controllers is the sampling period that influences the magnitude of the control signal. When deadbeat control is used, the magnitude of the control signal increases as the sampling period decreases, otherwise, a larger overshoot occurs. Thus, it is important to choose the sampling period carefully when using deadbeat control. Besides increasing the sampling period, there are two other ways to reduce the overshoot.

1) One is to design an extended-order deadbeat controller[8], which allows u(0) or the initial control action to be specified. Since u(0) has the largest magnitude, this allows the overshoot to be controlled.
2) An alternate method is divide the r(t) (the desired final state) into two or three sublevels and reach final steady state in 2(n+1) or 3(n+1) sample times instead of n+1 sample times. This has the same effect as increasing the sample time. However, the final overshoot can be more precisely controlled, depending on how r(t) is subdivided.

The transfer function of a deadbeat controller is given by

$$G_{db} = \frac{p_O + p_1 z^{-1} + p_2 z^{-2} + \ldots p_n z^{-n}}{q_O + q_1 z^{-1} + q_2 z^{-2} + \ldots q_n z^{-n}} \tag{14}$$

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. The coefficients p_0, p_1, p_2, q_0, q_1, and q_2 are found from the plant parameters. Appendix C describes the complete equations to find these parameters. Solving for the parameters of the controller, the final transfer function for the controller is

$$G_{db} = \frac{0.1566 - 0.3129z^{-1} + 0.1564z^{-2}}{1 - 0.4218z^{-1} - 0.4216z^{-2}} \tag{15}$$

Figure 13 shows the block diagram of a deadbeat controller and Figure 16 through Figure 15 show the step response of the deadbeat controller. The code to implement the deadbeat controller is given in Appendix E.

Figure 13. Deadbeat Controller Block Diagram

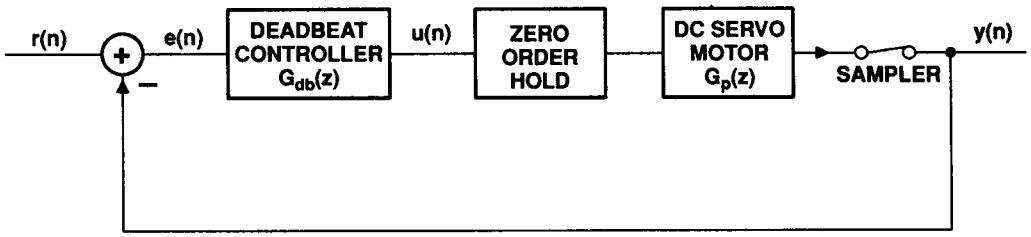


Figure 14. Step Response of the Deadbeat Controller

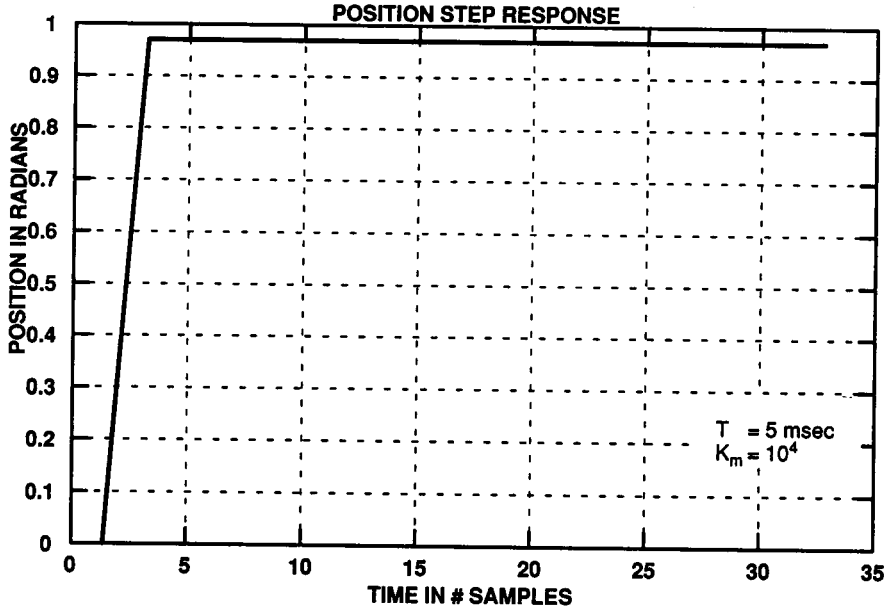


Figure 15. Step Response of the Deadbeat Controller

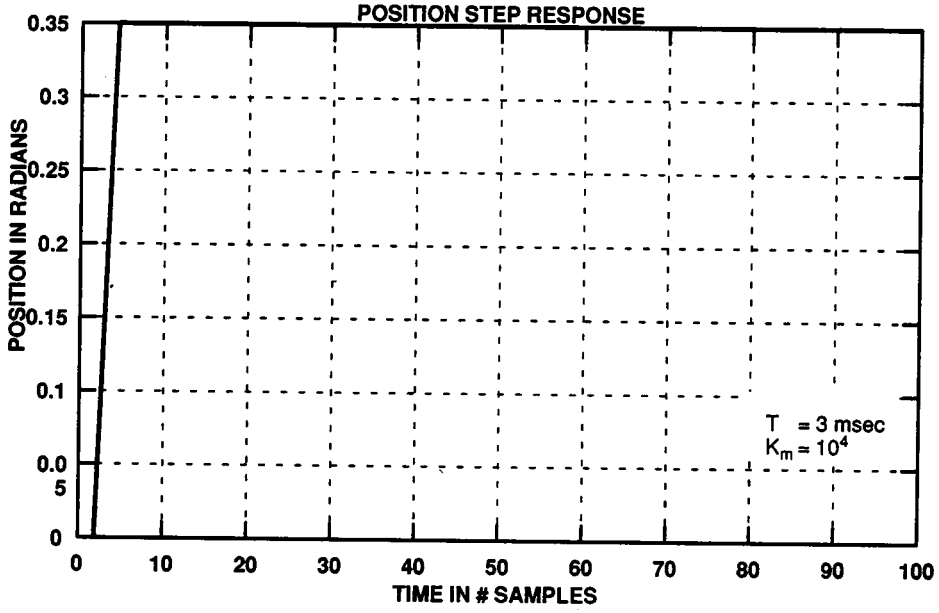
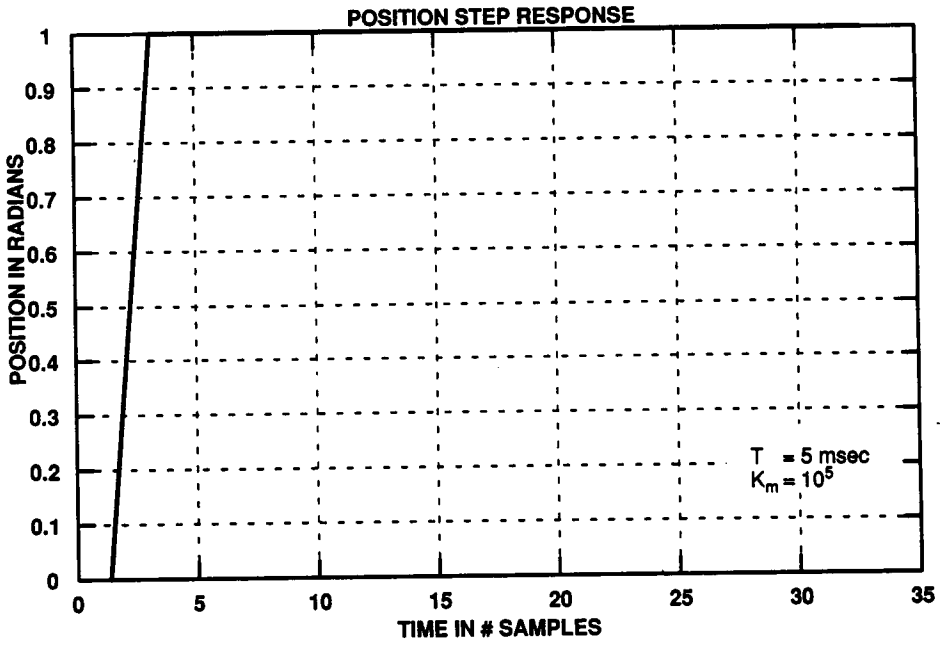


Figure 16. Step Response of the Deadbeat Controller



Implementing Digital Controllers

This section will discuss some of the issues in implementing digital controllers[5, 17] without going into mathematical details. For mathematical models refer to references [6 through 11]. Perhaps the most critical issue in implementation is the effects of finite wordlength.

Finite Wordlength Effects

Most digital controllers use fixed-point arithmetic processors. In a fixed-point arithmetic processor, only a finite amount of storage length (i.e., 4, 8, or 16 bits) is available to represent the magnitude of a signal or a gain constant. Signals and coefficients have to be scaled to fit in the dynamic range and wordlength of the processor. This limited storage capacity is referred to as the finite wordlength issue. The effects of finite wordlength show up as noise or limit cycles in the system, and may even cause instability. These effects are also referred to as quantization noise.

Another effect of finite wordlength shows up in the processing of signals. As intermediate calculations are carried out, higher precision is needed. For example, a 16×16 multiply needs a 32-bit register to store the result. If only 16 bits are available, the lower 16 bits are thrown away. This results in a loss of precision in the result, referred to as roundoff error. As successive calculations are carried out, these errors will accumulate. Another effect of finite wordlength is overflow management. Too often, registers will overflow during calculations. This usually causes registers to wrap around from most positive to most negative.

To minimize finite wordlength effects, a minimum of 16-bit wordlength is required, with 32-bit registers for internal precision. In addition, extensive simulations should be carried out to determine the dynamic range of the signals. Once system dynamics are known, proper scaling factors along with structure optimization techniques can reduce most of the effects of finite wordlength.

Selection of a proper scaling factor is critical in minimizing the effects of finite wordlength. The scale factor should support the full dynamic range of signals and coefficients. A large scale factor may cause overflows. Although overflow protection is built into the TMS320 architecture, it is advisable to minimize overflows. To minimize overflow, sometimes it may be necessary to choose a smaller (12–13 bits) scale factor. A small scale factor, on the other hand, may cause quantization noise or even underflow.

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 arithmetic processor. Simulations are required to determine the dynamic range. However, in some cases, it may be possible to switch modes and change scale factors.

If gain coefficients have a large dynamic range, direct structures should be avoided and broken into smaller cascaded structures. Different scale factors can be chosen for different sections. Another approach is to use structure transformation techniques like Schur transformation or Modal transformation 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 processor calculations.

Fixed-Point Versus Floating-Point Arithmetic Processors

One of the ways to avoid finite wordlength effects is to use a floating-point arithmetic processor. Floating-point arithmetic processors have a very large dynamic range. A 32-bit floating-point arithmetic processor has a dynamic range large enough for most control system applications. Floating-point arithmetic processors are usually expensive; their cost can be justified in only a few applications. Floating-point arithmetic processors may be needed in applications where either signals or gain coefficients are time varying and have a large dynamic range. Another case that justifies floating-point arithmetic processors is one in which cost of development cost is more significant than component cost and very low volumes are required.

If gain coefficients have a large dynamic range but are constant, their dynamic range can usually be reduced by structure optimization techniques. Floating-point arithmetic processors usually allow code to be developed in high-level languages and reduce the need to fully identify system dynamic range.

Sampling Rate Selection

An important consideration is the selection of sampling rate. In signal processing, the sampling rate is chosen to be at least twice the bandwidth, or the highest frequency component in the systems. In control systems, the sampling rate chosen is six to ten times the system bandwidth. If lower sampling rates are selected, noise from higher frequency components may be introduced into the system and can be indistinguishable from the signal. Anti-aliasing filters are used before the controller to filter out high frequency components. A first-order filter should be used to minimize the phase shift.

Controller Design Tools

A major consideration in using digital controllers is the design of hardware and software. One advantage of using digital controllers is that a large number of CASE (computer aided software engineering) tools are becoming available. These tools tremendously increase the productivity of the control designer.

Code Development

Software or code development cost is a major concern in implementing digital controllers. The programmable approach to controllers allows easy upgrade and maintenance. It also protects development investment, but at the same time it requires more initial development effort. Six different approaches can be taken for software development:

- *High-Level Languages (HLL)* – Use of an HLL like C, Pascal, or FORTRAN can substantially cut development effort. Such languages are familiar and easy to program. However, they are not optimized with respect to signal processing or to a particular processor's architecture. Code compiled on a processor may be two to four times the size of assembly code. This is a high penalty in time-critical signal processing applications.
- *Assembly Language* – The most efficient coding occurs in assembly language. Even when an HLL language is used, it may be necessary to use assembly language for the more

time-critical sections. Assembly language programming requires an intimate knowledge of the processor architecture. However, the nature of performance requirements in signal processing requires maximum code efficiency, leaving very little choice in some cases.

- *Signal Processing Languages* – This may provide a mid-ground between the two approaches discussed above. Special languages designed for signal processing may give the ease of development of standard HLL languages. At the same time, they can give some of the efficiency of assembly language since they are designed for specific signal processing applications. One example of these special languages is DSPL from dSPACE[18]. However, there are no standards for these languages, and none of the languages are widely known.
- *Code Generation Software* – Code generation packages are becoming available that automatically generate assembly code for particular processors. For example, the Impex software package[19] from dSPACE will generate TMS320 assembly code from a mathematical description of the controller. The DFDP (Digital Filter Design Package)[20] 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 processor architecture in developing software.
- *Controller Design and Simulation Software* – Controller design and simulation packages are also becoming popular. Packages like Matrix-X[21] and PC-Matlab[22] can be used for simulation and design of controllers. These packages allow use of pole placement and other techniques. Packages like Simnon[23] are extremely good for simulation of continuous and discrete controllers. These packages greatly increase the productivity of the control engineer.
- *Device Simulators* – Another very useful tool in designing software is the device simulator. Simulators for the TMS320 family run on common platforms like PC or VAX and provide full simulation of the instruction set, along with instruction timing. These allow simulation of the controller software to fully check the effects of math on internal registers and memory without the necessity of building hardware.

Hardware Design

A large variety of tools is available for designing the hardware for a controller. These include target systems and EVMs that plug into a PC or stand alone, and in-circuit emulators that can be used for complete system debugging. Also available are device behavioral models. These models can be used to simulate the timing and behavior of a complete target system without building any actual hardware. Logic Automation provides behavioral models for most members of the TMS320 family that run on popular workstations. Also available are logic analyzers from manufacturers like HP and Tektronix that can be used for extensive tracing. These logic analyzers can disassemble captured data to allow debugging of code[24].

Applications

An increasing number of designers for control system applications are turning to DSPs to solve their problems. The capabilities of DSPs are also making applications practical that were previously impossible to implement or not cost effective. As costs of DSPs come down, they will replace microcontrollers and analog components in most control applications. Some of the control system applications in which DSPs are already cost effective are servo control in computer peripherals, vector control in AC motors, motion control in robotics and NC machines, and power control in power supplies and uninterruptible power supplies.

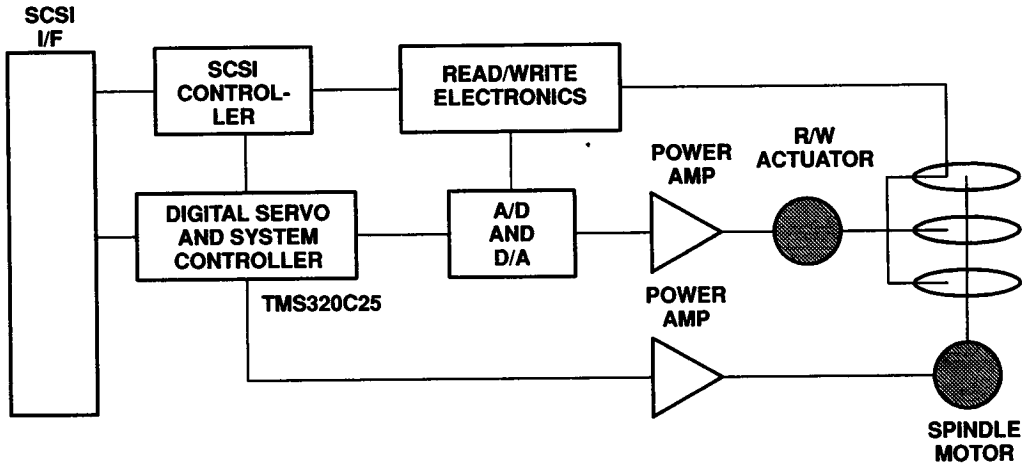
Computer Peripherals

A large number of applications in computer peripherals are starting to use DSPs. These applications include read/write head control in winchester disk drives, tape control in tape drives, pen control in plotters, beam positioning and focusing in optical disks and in paper feed and print head control in printers.

Disk Drives

Disk drive designers were early adopters of DSPs. DSPs are used for servo control of the actuator driving the read/write head. Disk drives employ a voice-coil motor with high bandwidth. In addition, data is read from the disk at a very high rate. Sampling rates of up to 50 kHz are sometimes used. Besides implementing the compensator, DSPs implement notch filters to cancel mechanical resonances or vibrations[25]. Figure 17 shows the block diagram of a disk drive.

Figure 17. Disk Drive Block Diagram

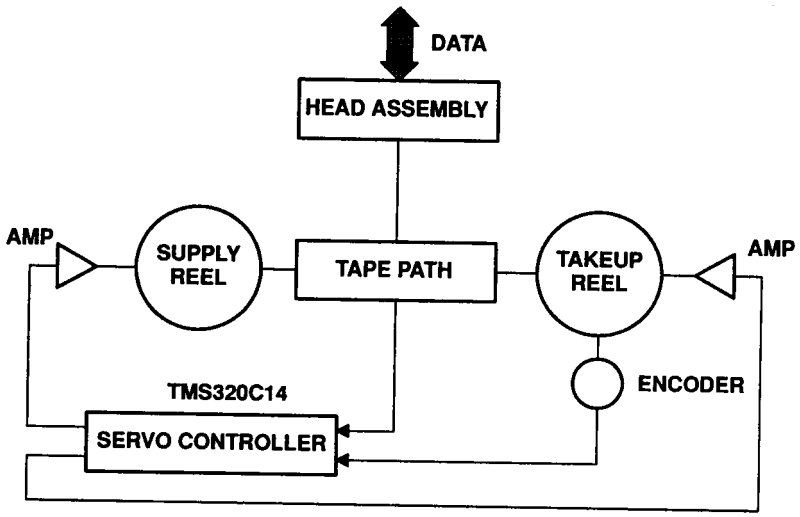


Tape Drives

In tape drives, DSPs are used for control of the tape mechanism. A tape drive has two servo loops that control tape speed and tension on the tape. Position feedback is obtained from an optical

encoder, and tension information is fed from a tension sensor. In addition, DSPs are also used to cancel mechanical resonances. Figure 18 shows the block diagram of a tape drive.

Figure 18. Block Diagram of a Tape Drive Controller



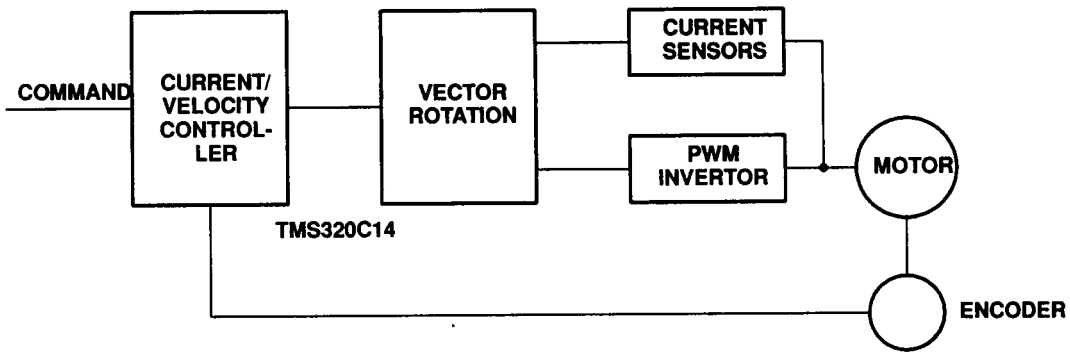
Power Electronics

DSPs can be used for multiple applications in power electronics. These applications include AC servo drives, converter 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 resulting from the cross-coupling of three-phase currents. Vector rotation techniques are used to transform three-phase to rotating two-phase d-q axes. This simplifies the analysis to a field-wound DC motor[26, 27, 28]. Figure 19 shows the block diagram of AC servo control.

Figure 19. AC Servo Control



UPS/Power Converters

In uninterruptible power supplies and power converters, DSPs are used for PWM generation along with power factor correction and harmonic elimination. Advanced mathematical techniques can be used to control the firing angles of the inverter to create low harmonic PWM with unity power factors[29, 30].

Robotics/Motion Control

DSPs are being used in large applications in robotics and other axis-control applications. DSPs allow high-precision control along with implementation of advanced techniques like state estimators and adaptive control. A single controller can handle speed/position control along with current control. Time-varying loads can be handled by using 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 can be used to implement notch filters to cancel resonances or vibrations[31].

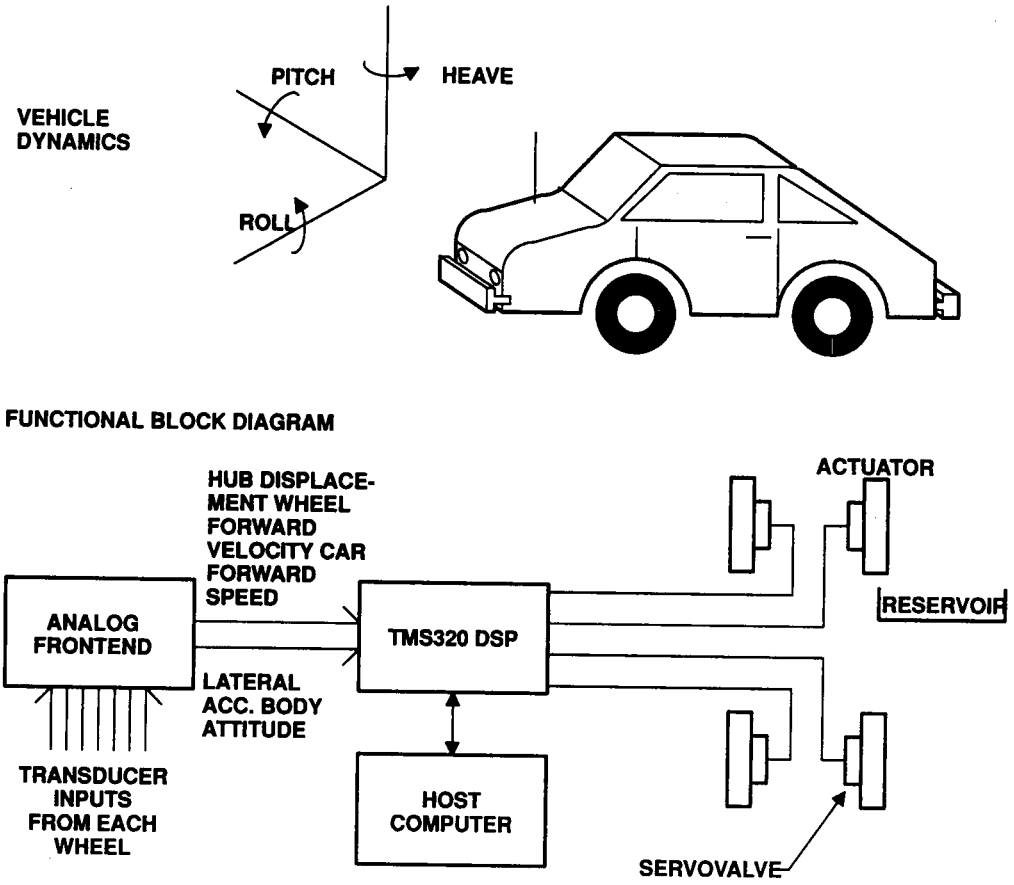
Automotive

DSPs can be used for many applications in automotive design. These applications include active suspension, anti-skid braking, engine and transmission control, and noise cancellation [32, 33].

Active Suspension

Active suspension systems use four hydraulic actuators, one at each corner of the vehicle. DSPs can take into consideration body dynamics, such as pitch, heave, and roll. This information is used to control the four actuators independently, and to dynamically counter the external forces and car attitude changes[34]. Figure 20 shows block diagram of an active suspension system.

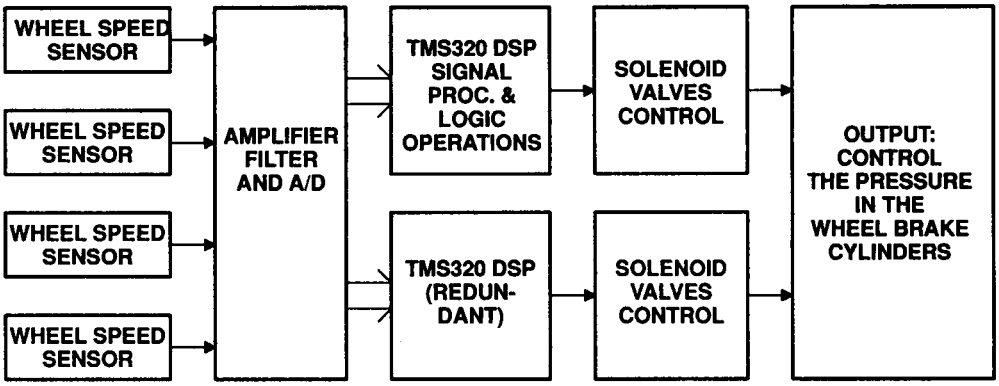
Figure 20. An Active Suspension System



Anti-Skid Braking

In anti-skid braking systems, DSPs can be used to read the wheel speed from sensors, calculate the skid, and control the pressure in the wheel brake cylinder. Traction control can be added to control the vehicle in extreme condition (wheel lock and spinning) and to further increase vehicle stability, steerability, and drivability. Figure 21 shows a block diagram of an ABS system.

Figure 21. Anti-Lock Braking System (ABS) Block Diagram



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 used to determine the best spark timing, firing angles, and the optimal air/fuel ratios. The closed-loop engine control scheme can tolerate external turbulences, aging, wearing, etc., and maintains optimum engine performance and fuel efficiency.

Summary

This report has discussed implementation of digital control systems with the TMS320 family of processors, using various control algorithms. The report has focused on showing design procedures and implementation of generic digital control systems without going into specific applications or choosing a particular approach or algorithm. Obviously, selection of a specific approach depends on the requirements of a particular application. However, the procedures outlined in this report can be applied with minor modifications to a wide range of applications. This report has also attempted to provide a bridge for control system designers who have been trained in analog control design and want to convert their analog designs into digital designs for stability, higher performance, or other reasons.

TMS320-based digital control systems have numerous advantages over analog-based designs. The high processing speed of the TMS320 family allows sophisticated control techniques to be used to build a high-precision control system. Digital systems are insensitive to component aging and temperature drift, thus minimizing variation in controller gain coefficients. With the TMS320 an adaptive control system can also be designed, thus creating a truly robust system that is insensitive to plant parameter variation. A digital control system using the TMS320 can also be employed to control multiple devices or time shared between different processes. An observer system may also be designed with a TMS320-based system to eliminate expensive sensors.

In addition to the advantages outlined above for TMS320-based control systems, the tradeoffs and disadvantages in implementing digital control are no longer applicable. The 16-bit word length, and 32-bit ALU and 32-bit accumulator of the TMS320 make quantization errors negligible. The hardware scaling shifters of the TMS320 family further minimize errors due to quantization and truncation. The fast processing speed of the TMS320 allows high sampling rates to be used, thus giving analog-like performance and minimizing delay time.

References

1) *TMS320C1x User's Guide*, literature number SPRU013B, Texas Instruments, 1989.
2) *TMS320C14/E14 Users Guide*, literature number SPRU032, Texas Instruments, 1988.
3) *TMS320C2x User's Guide*, literature number SPRU014A, Texas Instruments, 1989.
4) *TMS320C3x User's Guide*, literature number SPRU031, Texas Instruments, 1988.
5) *Digital Signal Processing Applications with the TMS320 Family*, literature number SPRA012, Texas Instruments, 1986.
6) Astrom, K., and Wittenmark, B., *Computer Controlled Systems*, Prentice-Hall Inc., 1984.
7) Phillips, C., and Nagel, H., *Digital Control System and Analysis and Design*, Prentice-Hall Inc., 1984.
8) Iserman, R., *Digital Control Systems*, Springer-Verlag, 1981.
9) Franklin, G., and Powell, D., *Digital Control of Dynamic Systems*, Addison-Wesley, 1980.
10) Jacquot, R., *Digital Control Systems*, Marcel Dekker, 1981.
11) Katz, P., *Digital Control Using Microprocessors*, Prentice-Hall Inc., 1981.
12) Lewis, F., *Optimal Control*, John Wiley and Sons, 1986.
13) Lewis, F., *Optimal Estimation*, John Wiley and Sons, 1986
14) Kyriakopoulos, N. and Tan, J., "Implementation of a Tracking Kalman Filter on a Digital Signal Processor," *IEEE Transactions on Industrial Electronics*, pp. 126–134, Volume 35, Number 1, February 1988.
15) Astrom, K. and Hagglund, T., "Automatic Tuning of PID Controllers," *Instrument Society of America*, 1988.
16) Astrom, K., and Wittenmark, B., "Adaptive Control," Addison-Wesley, 1988.
17) Hanselmann, H., "Implementation of Digital Controllers – A Survey," *Automatica*, Volume 23, Number 1, pp. 7–32, (1987).
18) Hanselmann, H., and Schwarte, A. "Generation of Fast Target Processor Code from High Level Controller Descriptions," *Proceedings of 10th IFAC World Congress*, Munich, July 1987.
19) Impex is a trademark of dSPACE, Paderborn, Germany.
20) DFDP is a trademark of Atlanta Signal Processors, Atlanta, GA.
21) Matrix-X is a trademark of Integrated Controls, Santa Clara, CA.
22) Matlab is a trademark of Mathworks, Inc, South Natick, Ma.
23) Simnon is a trademark of Lund Institute of Technology, Lund, Sweden.
24) *TMS320 Family Development Support Reference Guide*, literature number SPRA011A, Texas Instruments, 1989.
25) Hanselmann, H., and Engelke, A., "LQG – Control of a Highly Resonant Disc Drive Head Positioning Actuator," *IEEE Transactions on Industrial Electronics*, Volume 35, Number 1, pp. 100–104, February 1988.
26) Bose, B. K., and Szczesny, P., "A Microcomputer-Based Control and Simulation of an Advanced IPM Synchronous Machine Drive System for Electric Vehicle Propulsion," *Proceedings of IECON'87*, pp. 454–463, October 1987.
27) Matsui, N., and Ohasi, H., "DSP-Based Adaptive Control of a Brushless Motor," *Proceedings of IECON'88*, pp. 375–380, October 1988.

28) Leonhard, W., Lessmeier, R., and Schumacher, W., "Microprocessor-Controlled AC-Servo Drives With Synchronous or Induction Motors: Which is Preferable?," *IEEE Transactions on Industry Applications*, September/ October 1986.
29) Chance, R., and Taufiq, T., "A TMS32010 Based Near Optimized Pulse Width Modulated Waveform Generator," *Third International Conference on Power Electronics & Variable Speed Drives*, Conference Publication Number 291, July 1988.
30) Garate, I., Carrasco, R., and Bowden, A., "An Integrated Digital Controller for Brushless AC Motors Using a DSP Microprocessor," *Third International Conference on Power Electronics & Variable Speed Drives*, Conference Publication Number 291, pp. 249–252, July 1988
31) Tomizuka, M., Horowitz, R., and Anwar, G., "Implementation of a MRAC for a Two Axis Direct Drive Robot Manipulator Using a Digital Signal Processor," *Proceedings of American Control Conference*, pp. 658–660, 1988.
32) Costin, M. and Elzinga, D., "Active Reduction of Low-Frequency Tire Impact Noise Using Digital Feedback Control," *IEEE Control Systems Magazine*, pp. 3–6, August 1989.
33) Lin, K., " Trends of Digital Signal Processing In Automotive," *Proceedings of CONVERGENCE '88*, October 1988.
34) Williams, D. and Oxley, S., "Application of the Digital Signal Processor to an Automotive Control System," *Proceedings of the Sixth International Conference on Automotive Electronics*, Great Britain, October 1987.

Appendix A

Plant Modeling

The discrete time model for a DC motor can be derived from the electrical and mechanical characteristics of the motor. The mechanical characteristics are:

$$J_m\ddot{\theta} + B\dot{\theta} + K\theta = T_L - J_L\ddot{\theta} \tag{16}$$

where

Jm = motor inertia
B = viscous damping
K = stiffness
J_L = load inertia
θ = position or angular displacement
$\dot{\theta}$ = $d\theta/dt$ = angular velocity
$\ddot{\theta}$ = $d^2\theta/dt^2$ = angular acceleration
T_L = load torque = $K_t \times i$
K_t = torque constant
i = current

The electrical characteristics are given by

$$L\frac{di}{dt} + Ri = V - EMF \tag{17}$$

where

L = inductance
R = resistance
V = applied voltage
EMF = $K_e \times \dot{\theta}$ = back emf
K_e = emf constant
$\dot{\theta}$ = angular velocity

The electrical time constant is given by L/R, and the mechanical time constant is given by B/J.

In practice, L/R << B/J; i.e., electrical steady-state conditions are reached quickly. Assuming steady-state current is reached, (17) is reduced to

$$Ri = V - EMF = V - K_e\dot{\theta} \tag{18}$$

Combining (16) and (18) gives

$$(J_m + J_L)\ddot{\theta} + B\dot{\theta} + K\theta = K_t\,(V - K_e\dot{\theta})/R \tag{19}$$

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

$$\ddot{\theta} + 1/J(B + K_t K_e/R)\,\dot{\theta} = 1/J(K_t/R)V \tag{20}$$

where

$a = 1/J\,(B + K_t\,K_e/R)$
$b = 1/J\,(Kt/R)$

The Laplace transform of (20) is

$$\text{or} \qquad \begin{aligned} (s^2 + as)\theta(s) &= bV(s) \\ \theta(s)/V(s) &= b/s(s + a) \end{aligned} \tag{21}$$

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

ZOH states that

$$G(z) = (1 - z^{-1})\mathbf{Z}(\mathbf{L}^{-1}G(s)/s) \tag{22}$$

Then,

$$\frac{G\,(s)}{s} = \frac{b}{s(s+a)(s)} = \frac{b}{s^2(s+a)} \tag{23}$$

Expanding as partial fractions, (23) is expressed as

$$\frac{G(s)}{s} = \frac{A_1}{s} + \frac{A_2}{s^2} + \frac{A_3}{s+a} \tag{24}$$

Solving for A_1, A_2, and A_3 gives

$$\frac{G(s)}{s} = \frac{(-b/a^{2})}{s} + \frac{(b/a)}{s^2} + \frac{(b/a^2)}{s+a} \tag{25}$$

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

$$G(z) = \frac{b/a^2\,(e^{-aT} - 1 + aT)z^{-1} + b/a^2(1 - e^{-aT} - aTe^{-aT})\,z^{-2}}{1 - (1 + e^{-aT})\,z^{-1} + e^{-a}Tz^{-2}} \tag{26}$$

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

$$G_m(z) = \frac{\theta(z)}{V(z)} = \frac{0.269z^{-1} + 0.269z^{-2}}{1 - 1.999z^{-1} + 0.999z^{-2}} \times K_g 10^{-4} \tag{27}$$

where K_g is a gain constant.

By introducing a numerator gain factor, (27) can be rewritten as

$$G_m(z) = \frac{\theta(z)}{V(z)} = \frac{0.269z^{-1} + 0.269z^{-2}}{1 - 1.999z^{-1} + 0.999z^{-2}} \times K_m \tag{28}$$

where K_m is a numerator gain factor.

Appendix B

PID Controller

The PID algorithm is given by

$$u(t) = K_p e(t) + K_i \int edt + K_d \frac{de}{dt} \tag{29}$$

where

K_p, K_i, K_d = PID gain constants
$u(t)$ = control signal
$e(t)$ = error signal

Rectangular Approximation

To convert to discrete form, the integral term edt is approximated by the summation of rectangles $\sum e_i \times T$, where T is the sampling interval and e_i is the value of the error signal at sample time i, written as

$$\int edt = \sum e_i T \tag{30}$$

If the sampling interval T is small enough, the differential term d_e/d_t can be approximated by

$$\frac{de}{dt} = \frac{e\ (n) - e\ (n-1)}{T} \tag{31}$$

where $e(n)$ and $e(n–1)$ are values of the error signal e at time intervals n and $n–1$.

The PID algorithm can now be approximated in discrete form by

$$u(n) = K_p e(n) + K_i \sum e_i T + K_d[e(n) - e(n-1)]/T \tag{32}$$

To reduce (32) into a difference equation, (32) for time interval $n–2$ is written as

$$u(n-2) = K_p e(n-2) + K_i \sum_{i=1}^{n-2} e_i T + K_d[e(n-1) - e(n-2)]/T \tag{33}$$

Subtracting (33) from (32) gives

$$\begin{aligned} u(n) - u(n-2) &= K_p e(n) - e(n-2) + K_i[e(n) + e(n-1)]T \\ &\quad + (K_d/T)\{\ e(n) - 2e(n-1) + e(n-2)\ \} \end{aligned} \tag{34}$$

Combining similar terms gives

$$u(n) = u(n-2) + (K_p + K_d/T + K_iT)e(n) - (K_iT - 2K_d/T)e(n-1) + (K_d/T - K_p)e(n-2) \quad (35)$$

or

$$u(n) = u(n-2) + K_1e(n) + K_2e(n-1) + K_3e\ (n-2)$$

where

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

K_1, K_2, and K_3 are obtained from the design of K_p, K_i, and K_d, which are designed by using conventional techniques. One way of designing is to use the Ziegler and Nichols ultimative-sensitivity method [6]. With this approach, a proportional controller is first used to control the system. The gain of the controller, Kmax, and the period time, T_p, when the closed loop is on the stability boundary, are measured. The parameters K_p, K_i, and K_d can then obtained as follows:

$K_p = 0.6$ Kmax
$K_i = T_p/2$
$K_d = T_p/8$

Another tuning method is to use the phase margin and critical frequencies [7]. Using this technique, K_p, K_i, and K_d can be computed as follows.

$$\theta = 180 + \phi_m - G_m(j\omega) \mid \omega = \omega_c$$

$$Kp = \frac{\cos(\theta)}{|G_m(j\omega)|\omega = \omega_c} \quad (36)$$

$$K_d\omega_c - K_i/\omega_c = \frac{\sin(\theta)}{|G_m(j\omega)| \omega = \omega_c}$$

By substituting an arbitrary value of K_i in the above equation, we can obtain K_d. Using this technique and designing with the following parameters

$\phi_m = 55°$ (phase margin)
$\omega_c = 628.315$ radians (critical frequency = 100 hz)

we obtain

$$|G_m(j\omega| \omega = \omega_c = 0.0001365$$

$$G_m(j\omega)|\omega = \omega_c = -179.8982 \quad (37)$$

The PID constants are then found to be

$K_p = 4181$
$K_d = 9.569$
$K_i = 1$

K_1, K_2, and K_3 are then obtained as

K_1 = 13751
K_2 = −19138
K_3 = 5387

Both these methods give approximate answers. Further fine tuning of the parameters may be necessary to get the desired response from the system.

The controller is obtained as

$$u(n) = u(n-1) + K_1e(n) + K_2e(n-1) + K_3e(n-2) \tag{38}$$

Trapezoidal Approximation

Another method for converting the analog form of the PID algorithm into a discrete form is to use a trapezoidal approximation, sometimes referred to as the Tustin approximation.

The PID is again given as

$$u(t) = K_pe(t) + K_i \int edt + K_d\frac{de}{dt} \tag{39}$$

where

$$e(t) = \frac{de}{dt} \tag{40}$$

The Laplace transform of that is expressed as

$$U(s) = K_pE(s) + K_iE(s)/s + K_dsE(s) \tag{41}$$

Combining gives

$$U(s) = (K_p + K_ds + K_i/s)E(s) \tag{42}$$

Conversion into discrete form requires a transfer from s-domain to z-domain by using the Tustin approximation,

$$s = \frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}} \tag{43}$$

where T = sampling period.

Substituting (43) in (42) gives

$$\frac{U(z)}{E(z)} = \frac{(2K_pT + 4K_d + T^2K_i) + (2T^2K_i - 8K_d)z^{-1} + (4K_d - 2TK_p + T^2K_i)z^{-2}}{2T(1 - z^{-2})} \tag{44}$$

Further computation yields

$$U(z) - U(z)z^{-2} = K_1E(z) + K_2E(z)z^{-1} + K_3E(z)z^{-2} \tag{45}$$

where

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

In (45), z^{-1} represents a delay of one sample time. Taking an inverse z-transform of (45) gives

$$u(n) - u(n-2) = K_1e(n) + K_2e(n-1) + K_3e(n-2)$$

or

$$u(n) = u(n-2) + K_1e(n) + K_2e(n-1) + K_3e(n-2) \tag{46}$$

This is the final form of the PID controller. However, before implementing the controller, the constants K_p, K_i, and K_d must be located. Alternatively, constants K_1, K_2, and K_3 have to be determined. These constants can be found by locating the poles of the equivalent system transfer function (i.e., controller + plant).

The transfer function for the controller can be stated as

$$G_c(z) = \frac{K_1 + K_2z^{-1} + K_3z^{-2}}{1 - z^{-2}} \tag{47}$$

The transfer function of the plant is given by

$$G_p(z) = \frac{0.2694z^{-1} + 0.2693z^{-2}}{1 - 1.999z^{-1} + 0.999z^{-2}} \times K_m \tag{48}$$

The overall system transfer function is expressed as

$$G_s(z) = \frac{G_p(z)G_c(z)}{1 + G_p(z)G_c(z)} \tag{49}$$

The denominator of the system transfer function provides the poles of the overall system. The stability and robustness of the system depends 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 for the denominator is obtained. Program 4 for PC-Matlab, in Appendix D, lists the steps to obtain the characteristics equation from pole locations and obtain values of K1, K_2 and K_3. To solve for values of K_1, K_2, K_3, the coefficients of powers of z for the denominator of the system transfer function are compared with the desired polynomial.

Solving for K_1, K_2, and K_3 gives
$K_1 = 1.4795,$
$K_2 = -2.845,$
$K_3 = 1.3636$

The controller is

$$u(n) = u(n-2) + 1.4795e(n) - 2.8405e(n-1) + 1.3636e(n-2) \tag{50}$$

Appendix C

Deadbeat Controller

A deadbeat controller has the property of settling to a final state in a finite time. It has the form,

$$G_{db}(z) = \frac{p_0 + p_1 z^{-1} + p_2 z^{-2} \ \ldots\ldots\ p_n z^{-n}}{q_0 + q_1 z^{-1} + q_2 z^{-2} \ \ldots\ldots\ q_n z^{-n}} \tag{51}$$

To design the deadbeat controller, its coefficients p_0, p_1, ... q_0, q_1, .. have to be found from the parameters of the motor.

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

$$G_{dp}(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} \ \ldots\ldots\ b_n z^{-n}}{a_0 + a_1 z^{-1} + a_2 z^{-2} \ \ldots\ldots\ a_n z^{-n}} \tag{52}$$

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

$p_0 = r / \sum b_i = r / (b_0 + b_1 + b_2)$
$p_1 = a_1 p_0$
$p_2 = a_2 p_0$
.
.
.
$p_n = a_n p_0$

and

$q_0 = r - p_0 b_0$
$q_1 = -b_1 p_0$
$q_2 = -b_2 p_0$
.
.
.
$q_n = -b_n p_0$

The transfer function of the dc servo motor is

$$G_m(z) = \frac{0.269z + 0.269}{z^2 - 1.999z + 0.999} \times K_m \qquad (K_m = 4000) \tag{53}$$

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,

$a_0 = 1, \quad a_1 = -1.999, \quad a_2 = 0.999$
$b_0 = 0, \quad b_1 = 0.269 \quad b_2 = 0.269$

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

Solving for the coefficients yields

$p_0 = 1/(b_0 + b_1 + b_2) = 0.1566$
$p_1 = a_1 p_0 = -0.3129$
$p_2 = a_2 p_0 = 0.1564$
$q_0 = 1 - p_0 b_0/r = 1$
$q_1 = -b_1 p_0 = -0.4218$
$q_2 = -b_2 p_0 = -0.4216$

The controller becomes

$$G_{db}(z) = \frac{0.1566 - 0.3129z^{-1} + 0.1564z^{-2}}{1 - 0.4218z^{-1} - 0.4216z^{-2}} \tag{54}$$

or, in time domain, it can be represented as

$$\begin{aligned} u(n) &= 0.4218u(n-1) + 0.4216u(n-2) \\ &\quad + 0.1566e(n) - 0.3129e(n-1) + 0.1564e(n-2) \end{aligned} \tag{55}$$

Appendix D

PC-Matlab is an interactive program developed by Mathworks for scientific and engineering numerical calculations. Included are many routines for design and simulations for signal processing and control systems. The programs in this appendix design the PID and deadbeat controllers and display step responses for the system. The programs are interactive and allow the user to change certain parameters. The programs use PC-Matlab and the Control Tool Box. Control Tool Box is a set of PC-Matlab utilities for control system design and simulation. PC-Matlab runs on MS-DOS computers. More information on PC-Matlab can be obtained from Mathworks. [22]

PC-Matlab Programs

Program 1

```
%    This program implements 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 give transfer function in s-domain
b=Kt/(j*R)
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
end
```

Program 2

```
%    This program implements design of a PID controller using classical
%    techniques. The design is then be converted into discrete form. This
%    design calculates values of Kp, Ki, and Kd or lets you enter these
%    values manually
%
' Enter 0 if you want to enter PID constants manually'
' Enter 1 if you want program to calculate these for you'
input('input 0 or 1   ')
X=ans
if X==1,
  N1=[0  0  b]              % numerator of transfer function in s-domain
  D1=[1  a  0]              % denominator of transfer function in s-domain
  input('input critical frequency in Hz  ')
  Wi=ans;                   % get critical frequency
  w=Wi*2*pi                 % Assuming critical frequency Wc=100Hz
  [mag,phase]=bode(N1,D1,w)  % calculate phase and magnitude of motor
  input(' input phase margin in degrees   ')
  Pm=ans;                   % get phase margin
  O=180 + Pm - phase - 360   % calculate theta
  Or=O/57                   % convert into radian
%
  Kp=cos(Or)/mag            % calculate  Kp
  Ki=1                      % assume Ki =1
  Kd =(Ki/w+sin(Or)/mag)/w   % Calculate Kd
  pause
elseif X==0,
  input('input p ')
  Kp=ans;
  input('input i ')
  Ki=ans;
  input('input d ')
  Kd=ans;
else 'No assigned values for constants, 0 or 1 was not input'
pause
end
K1 = Kp + Kd/T + Ki*T         % convert design into discrete form
K2 = Ki*T - 2*Kd/T
K3 = Kd/T - Kp
end
```

Program 3

```
%    This program simulates a closed loop system using a PID controller. The
%    PID controller has been designed using classical techniques and then
%    converted into discrete form using rectangular approximation.
%
motrans                         % call program to simulate motor
pidrect                         % call program to design PID constants
numl=[K1 K2 K3];                % numerator of controller
denl=[1  0  -1];                % denominator of controler
%
compnum=numl
compden=denl
procnum=num                     % numerator of motor
procden=den                     % denominator of motor
%
num5=conv(numl,num)             % multiply both numerators
den5=conv(denl,den)             % multiply both denominators
pause
input('specify the time in secs over which you want to see the step: ')
t=ans;
n=t/T;                          % calculate number of samples for simulation
input('input a loop gain: ')
g=ans;
u=ones(n,1);
closnum=g*num5                  % numerator of closed loop system
closden=g*num5+den5             % denominator of closed loop system
pause
y=dlsim(closnum,closden,u);     % do simulation
plot(y)
title('Position Step Response')
xlabel('Time in 0 samples')
ylabel('Position in radian')
grid
pause
end
end
```

Program 4

```
%    This program designs a PID controller using pole placement techniques.
%    Desired pole locations have to be input. The PID is converted into
%    discrete form using trapezeidal 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 a 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
end
```

Program 5

```
%    This file simulates 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
motrans                         % Call moter transfer function
dbeatdes                        % Call deadbeat parameters
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
```

Program 6

```
%   This file implements 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   + a1*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
end
```

Program 7

```
%   This program simulates 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
motrans                 % Call program to calculate motor transfer function
pidtdes                 % Call program to calculate controller gains
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 transfer 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 E

```
*       Program 1
*
*       PID Rectangular Controller
*
*
        .title      "PID Controller"
        .def        PID
*
*       This routine implements a PID controller
*
RV      .set    0           ; reference value
XN      .set    1           ; input from A/D
E0      .set    2           ; Latest error sample
E1      .set    3           ; Previous error sample
E2      .set    4           ; oldest error sample
K1      .set    5           ; gain constant
K2      .set    6           ; gain constant
K3      .set    7           ; gain constant
UN      .set    8           ; output to controller
U1      .set    9           ; previous output
U2      .set   10           ; oldest output
*
*       Processor initialization
*
reset   B       init        ; RS- processing begins here
int     B       isr
*
        .word   5632        ; coefficient K1
        .word  -7839        ; coefficient K2
        .word   2206        ; coefficient K3
*
*
init    LDPK    0           ; set DP pointer
        SOVM
        LARK    0,255
*
        ZAC
        LARP    0
loop    SACL    *           ; initialize memory
        BANZ    loop
        LACK    4           ; set program memory pointer to 4h
        TBLR    K1          ; load coeffcients into data memory
        LACK    5
        TBLR    K2
        LACK    6
        TBLR    K3
        EINT
        B       self        ; wait for interrupts
*
*    Process input sample
*
*       e(n) = r - x(n)
*
isr     IN      RV,PA2      ; read reference command input
        IN      XN,PA0      ; read input position signal on upper 13 bits
        LAC     XN,13
        SACH    XN
        LAC     RV
        SUB     XN          ; subtract from reference to give error
        SACL    E0

*
*   PID routine
*
*       u(n) = u(n-2) + K1*e(n) + K2*e(n-1) + K3*e(n-2)
*
PID     LAC     U2          ; Transfer u(n-2) to accumulator
        LT      E2          ; load T register with oldest sample e(n-2)
        MPY     K3          ; Preg = K3*e(n-2)
        LTD     E1          ; ACC = u(n-2) + K3*e(n-2), Treg = e(n-1)
        MPY     K2          ; Preg = K2*e(n-1)
        LTD     E0          ; ACC = u(n-2) + K3*e(n-2) + K2*e(n-1)
        MPY     K1          ; Preg = K1*e(n)
        APAC                ; ACC=u(n-2) + K3*e(n-2) + K2*e(n-1) + K1*e(n)
        SACH    UN,4        ; shift out 4 sign bbits
        OUT     UN,PA1      ; write to D/A - two's complement form
        DMOV    U1          ; transfer u(n-1) --->  u(n-2)
        DMOV    UN          ; transfer u(n)   --->  u(n-1)
        EINT
self    NOP
        B       self        ; wait for next interrupt
```

```
*       Program 2
*
*       PID Trapezoidal Controller
*
*
        .title      "PID Controller"
        .def        PID
*
*       This routine implements a PID controller
*
RV      .set    0           ; reference value
XN      .set    1           ; input from A/D
E0      .set    2           ; Latest error sample
E1      .set    3           ; Previous error sample
E2      .set    4           ; oldest error sample
K1      .set    5           ; gain constant
K2      .set    6           ; gain constant
K3      .set    7           ; gain constant
UN      .set    8           ; output to controller
U1      .set    9           ; previous output
U2      .set    10          ; oldest output
*
*       Processor initialization
*
reset   B       init        ; RS- processing begins here
int     B       isr
*
        .word   4060        ; coefficient K1 =  1.4795
        .word -11653        ; coefficient K2 = -2.8450
        .word   5585        ; coefficient K3 =  1.3636
*
*
init    LDPK    0           ; set DP pointer
        SOVM
        LARK    0,255       ; clear memory
*
        ZAC
        LARP    0
loop    SACL    *           ; initialize memory
        BANZ    loop
        LACK    4           ; set program memory pointer to 4h
        TBLR    K1          ; load coeffcients into data memory
        LACK    5           ; set program memory pointer to 5h
        TBLR    K2
        LACK    6           ; set program memory pointer to 6h
        TBLR    K3
        EINT
        B       self        ; wait for interrupt
*
*     Process input sample
*
*       e(n) = r - x(n)
*
isr     IN      RV,PA2      ; read reference command input
        IN      XN,PA0      ; read input position signal on upper 13 bits
        LAC     XN,13
        SACH    XN
        LAC     RV
        SUB     XN          ; subtract from reference to give error
        SACL    E0

*
*   PID routine
*
*       u(n) = u(n-2) + K1*e(n) + K2*e(n-1) + K3*e(n-2)
*
PID     LAC     U2          ; Transfer u(n-2) to accumulator
        LT      E2          ; load T register with oldest sample e(n-2)
        MPY     K3          ; Preg = K3*e(n-2)
        LTD     E1          ; ACC = u(n-2) + K3*e(n-2), Treg = e(n-1)
        MPY     K2          ; Preg = K2*e(n-1)
        LTD     E0          ; ACC = u(n-2) + K3*e(n-2) + K2*e(n-1)
        MPY     K1          ; Preg = K1*e(n)
        APAC                ; ACC=u(n-2) + K3*e(n-2) + K2*e(n-1) + K1*e(n)
        SACH    UN,4        ; store to memory and shift out 4 sign bits
        OUT     UN,PA1      ; write to D/A  - in two's complement form
        DMOV    U1          ; transfer u(n-1) --->  u(n-2)
        DMOV    UN          ; transfer u(n)   --->  u(n-1)
        EINT
self    NOP
        B       self        ; wait for next interrupt
```

```
*       Program 3
*
*       Deadbeat Controller
*
*
        .title      "Deadbeat Controller"
        .def        DBEAT
*
*       This routine implements a Deadbeat controller
*
RV      .set    0           ; reference value
XN      .set    1           ; input from A/D
E0      .set    2           ; Latest error sample
E1      .set    3           ; Previous error sample
E2      .set    4           ; oldest error sample
P0      .set    5           ; gain constant
P1      .set    6           ; gain constant
P2      .set    7           ; gain constant
Q1      .set    8           ; gain constant
Q2      .set    9           ; gain constant
UN      .set   10           ; output to controller
U1      .set   11           ; previous output
U2      .set   12           ; oldest output
*
*       Processor initialization
*
reset   B       init        ; RS- processing begins here
int     B       isr
*
        .word   5131        ; coefficient P0 =  .1566
        .word -10253        ; coefficient P1 = -.3129
        .word   5125        ; coefficient P2 =  .1564
        .word -13215        ; coefficient Q1 = -.4218
        .word -13815        ; coefficient Q2 = -.4216
*
*
init    LDPK    0           ; set DP pointer
        SOVM
        LARK    0,255
*
        ZAC
        LARP    0
loop    SACL    *           ; initialize memory
        BANZ    loop
        LACK    4           ; set program memory pointer to 4h
        TBLR    P0          ; load coeffcients into data memory
        LACK    5           ; set program memory pointer to 5h
        TBLR    P1
        LACK    6           ; set program memory pointer to 6h
        TBLR    P2
        LACK    7           ; set program memory pointer to 7h
        TBLR    Q1
        LACK    8           ; set program memory pointer to 8h
        TBLR    Q2
        EINT                ; enable interrupts
        B       self        ; wait for interrupt
*
*    Process input sample
*
*       e(n) = r - x(n)
```

```
*
isr     IN      RV,PA2          ; read reference command input
        IN      XN,PA0          ; read input position signal on upper 13 bits
        LAC     XN,13
        SACH    XN
        LAC     RV
        SUB     XN              ; subtract from reference to give error
        SACL    E0
*
*   Deadbeat Controller routine
*
*   u(n) = Q2*u(n-2) + Q1*u(n-1) + P0*e(n) + P1*e(n-1) + P2*e(n-2)
*
DBEAT   ZAC                     ; clear accumulator
        LT      E2              ; load T register with oldest sample e(n-2)
        MPY     P2              ; Preg = P2*e(n-2)
        LTD     E1              ; ACC =  P2*e(n-2), Treg = e(n-1)
        MPY     P1              ; Preg = P1*e(n-1)
        LTD     E0              ; ACC =  P2*e(n-2) + P1*e(n-1)
        MPY     P0              ; Preg = P0*e(n)
        LTA     U2              ; ACC =  P2*e(n-2)+P1*e(n-1)+P0*e(n), Treg=u(n-2)
        MPY     Q2              ; Preg = Q2*u(n-2)
        LTD     U1              ; ACC =  P2*e(n-2)+P1*e(n-1)+P0*e(n)+Q2*(n-1)
        MPY     Q1              ; Preg = Q1*u(n-1)
        APAC                    ; ACC=Q2*u(n-2)+Q1*u(n-1)+P0*e(n)+P1*e(n-1)+P2*e(n-2
        SACH    UN,4            ; write to memory and shift out 4 sign bits
        OUT     UN,PA1          ; write to D/A - in two's complement form
        DMOV    UN              ; transfer u(n)   --->  u(n-1)
        EINT
self    NOP
        B       self            ; wait for next interrupt
```