IFAC PROFESSIONAL BRIEF

Computer Control: An Overview

Björn Wittenmark http://www.control.lth.se/~bjorn

Karl Johan Åström http://www.control.lth.se/~kja

Karl-Erik Årzén http://www.control.lth.se/~karlerik

Department of Automatic Control
Lund Institute of Technology
Lund, Sweden

Abstract

Computer control is entering all facets of life from home electronics to production of different products and material. Many of the computers are embedded and thus "hidden" for the user. In many situations it is not necessary to know anything about computer control or real-time systems to implement a simple controller. There are, however, many situations where the result will be much better when the sampled-data aspects of the system are taken into consideration when the controller is designed. Also, it is very important that the real-time aspects are regarded. The real-time system influences the timing in the computer and can thus minimize latency and delays in the feedback controller.

The paper introduces different aspects of computer-controlled systems from simple approximation of continuous time controllers to design aspects of optimal sampled-data controllers. We also point out some of the pitfalls of computer control and discusses the practical aspects as well as the implementation issues of computer control.

Contents

1. Introduction

Computers are today essential for implementing controllers in many different situations. The computers are often used in *embedded systems*. An embedded system is a built-in computer/microprocessor that is a part of a larger system. Many of these computers implement control functions of different physical processes, for example, vehicles, home electronics, cellular telephones, and stand-alone controllers. The computers are often hidden for the end-user, but it is essential that the whole system is designed in an effective way.

Using computers has many advantages. Many problems with analog implementation can be avoided by using a computer, for instance, there are no problems with the accuracy or drift of the components. The computations are performed identically day after day. It is also possible to make much more complicated computations, such as iterations and solution of system of equations, using a computer. All nonlinear, and also many linear operations, using analog technique are subject to errors, while they are much more accurately made using a computer. Logic for alarms, start-up, and shut-down is easy to include in a computer. Finally, it is possible to construct good graphical user interfaces. However, sometimes the full advantages of the computers are not utilized. One situation is when the computer is only used to approximately implement an analog controller. The full potential of a computer-controlled system is only obtained when also the design process is regarded from a sampled-data point of view. The theory presented in this paper is essentially for linear deterministic systems. Even if the control algorithms give the same result for identical input data the performance of the closed-loop system crucially depends on the timing given by the real-time operating system. This paper gives a brief review of some of the tools that are important for understanding, analyzing, designing, and implementing sampled-data control systems.

Computer-controlled Systems

A schematic diagram of a computer-controlled system is shown in Figure 1. The system consists of

- Process
- Sampler together with Analog-to-Digital (A-D) converter
- Digital-to-Analog (D-A) converter with a hold circuit
- Computer with a clock, software for real-time applications, and control algorithms
- Communication network

The process is a continuous-time physical system to be controlled. The input and the output of the process are continuous-time signals. The A-D converter is converting the analog output signal of the process into a finite precision digital number depending on how many bits or levels that are used in the conversion. The conversion is also quantized in time determined by the clock. This is called the *sampling process*. The control algorithm thus receives data that are quantized both in time and in level. The control algorithm consists of a computer program that transforms the measurements into a desired control signal. The control signal is transfered to the D-A converter, which with finite precision converts the number in the computer into a continuous-time signal. This implies that the D-A converter contains both a conversion unit and a hold unit that

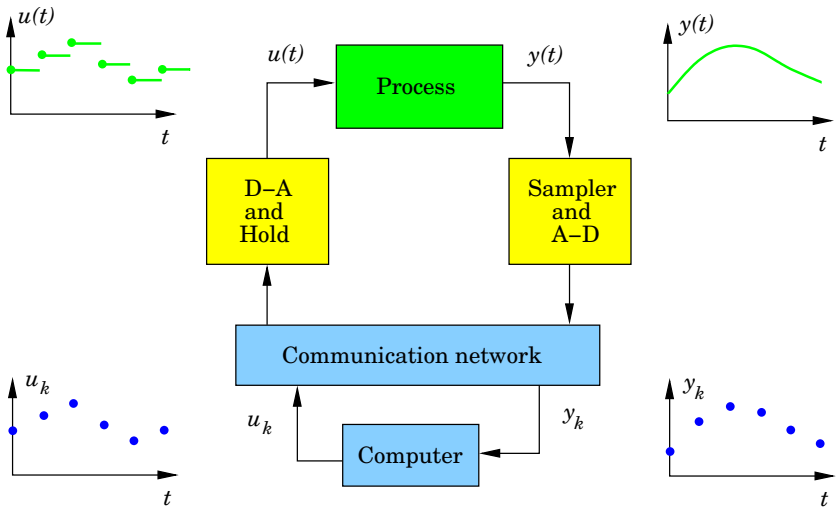

Figure 1 Schematic diagram of a computer-controlled system.

translates a number into a physical variable that is applied to the process. The communication between the process, or more accurately the A-D and D-A converters, and the computer is done over a communication link or network. All the activities in the computer-controlled system are controlled by the clock with the requirement that all the computations have to be performed within a given time. In a distributed system there are several clocks that have to be synchronized. The total system is thus a *real-time system* with hard time constraints.

The system in Figure 1 contains a continuous-time part and a sampled-data part. It is this mixture of different classes of signals that causes some of the problems and confusion when discussing computer-controlled systems. The problem is even more pronounced when using decentralized control where the control system is shared between several computers. Many of the problems are avoided if only considering the sampled-data part of the system. It is, however, important to understand where the difficulties arise and how to avoid some of these problems.

The Sampling Process

The times when the measured physical variables are converted into digital form are called the *sampling instants*. The time between two sampling instants is the *sampling period* and is denoted h. Periodic sampling is normally used, implying that the sampling period is constant, i.e. the output is measured and the control signal is applied each hth time unit. The sampling frequency is $\omega_s = 2\pi/h$. In larger plants it may also be necessary to have control loops with different sampling periods, giving rise to *multi-rate sampling* or *multi-rate systems*.

Time Dependence

The mixture of continuous-time and discrete-time signals makes the sampled-data systems time dependent. This can be understood from Figure 1. Assume that there is a disturbance added at the output of the process. Time invariance implies that a shift of the input signal to the system should result in a similar shift in the response of the system. Since the A-D conversion is governed by a clock the system will react differently when the disturbance is shifted in time. The system will, however, remain time independent provided that all changes

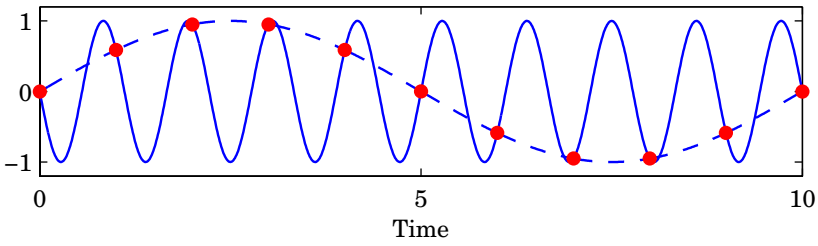

Figure 2 Two signals with different frequencies, 0.1 Hz (dashed) and 0.9 Hz (full), have the same values at all sampling instants (dots) when $h = 1$.

in the system, inputs *and* disturbances, are synchronized with the sampling instants.

Aliasing and New Frequencies

The A-D and D-A converters are the interface between the continuous-time reality and the digital world inside the computer. A natural question to ask is: Do we loose any information by sampling a continuous-time signal?

EXAMPLE 1—ALIASING

Consider the two sinusoidal signals $\sin((1.8t - 1)\pi)$ and $\sin 0.2\pi t$ shown in Figure 2. The sampling of these signals with the sampling period $h = 1$ is shown with the dots in the figure. From the sampled values there is thus no possibility to distinguish between the two signals. It is not possible to determine if the sampled values are obtained from a low frequency signal or a high frequency signal. This implies that the continuous-time signal cannot be recovered from the sampled-data signal, i.e. information is lost through the sampling procedure. □

The sampling procedure does not only introduce loss of information but may also introduce new frequencies. The new frequencies are due to interference between the sampled continuous-time signal and the sampling frequency $\omega_s = 2\pi/h$. The interference can introduce fading or beating in the sampled-data signal.

Approximation of Continuous-time Controllers

One way to design a sampled-data control system is to make a continuous-time design and then make a discrete-time approximation of this controller. The computer-controlled system should now behave as the continuous-time system provided the sampling period is sufficiently small. An example is used to illustrate this procedure.

EXAMPLE 2—DISK-DRIVE POSITIONING SYSTEM

A disk-drive positioning system will be used throughout the paper to illustrate different concepts. A schematic diagram is shown in Figure 3. Neglecting reso-

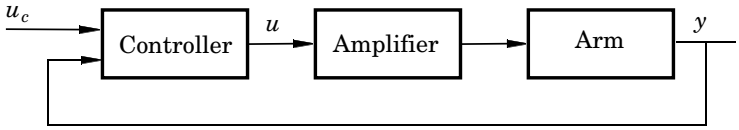

Figure 3 A system for controlling the position of the arm of a disk drive.

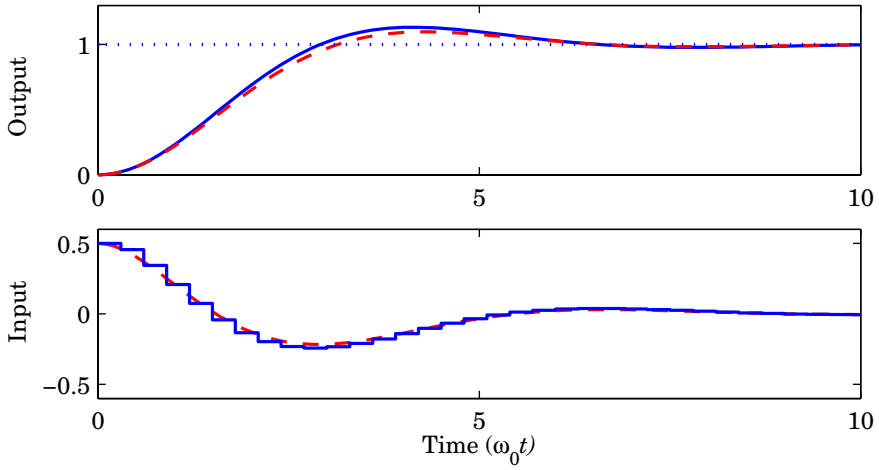

Figure 4 The step response when simulating the disk arm servo with the analog controller (2) (dashed/red) and sampled-data controller (3) (solid/blue). The design parameter is $\omega_0 = 1$ and the sampling period is $h = 0.3$.

nances the dynamics relating the position y of the arm to the voltage u of the drive amplifier is approximately described by a double integrator process

$$G(s) = \frac{k}{Js^2} \tag{1}$$

where k is the gain in the amplifier and J is the moment of inertia of the arm. The purpose of the control system is to control the position of the arm so that the head follows a given track and that it can be rapidly moved to a different track. Let u_c be the command signal and denote Laplace transforms with capital letters. A simple continuous-time servo controller can be described by

$$U(s) = \frac{bK}{a} U_c(s) - K\frac{s+b}{s+a} Y(s) \tag{2}$$

This controller is a two-degrees-of-freedom controller. Choosing the controller parameters as

$$\begin{aligned} a &= 2\omega_0 \\ b &= \omega_0/2 \\ K &= \frac{2J\omega_0^2}{k} \end{aligned}$$

gives a closed system with the characteristic polynomial

$$P(s) = (s^2 + \omega_0 s + \omega_0^2)(s + \omega_0) = s^3 + 2\omega_0 s^2 + 2\omega_0^2 s + \omega_0^3$$

where the parameter ω_0 is used to determine the speed of the closed-loop system. The step response when controlling the process with the continuous-time controller (2) is shown as the dashed line in Figure 4. The control law given by (2) can be written as

$$U(s) = \frac{bK}{a} U_c(s) - KY(s) + K\frac{a-b}{s+a} Y(s) = K\left(\frac{b}{a} U_c(s) - Y(s) + X(s)\right)$$

or in the time-domain as

$$u(t) = K\left(\frac{b}{a}u_c(t) - y(t) + x(t)\right)$$
$$\frac{dx(t)}{dt} = -ax(t) + (a-b)y(t)$$

The first of these equations can be implemented directly on a computer. To find an algorithm for the second equation the derivative dx/dt at time kh is approximated with a forward difference. This gives

$$\frac{x(kh+h) - x(kh)}{h} = -ax(kh) + (a-b)y(kh)$$

The following approximation of the continuous algorithm (2) is then obtained:

$$\begin{aligned} u(kh) &= K\left(\frac{b}{a}u_c(kh) - y(kh) + x(kh)\right) \\ x(kh+h) &= x(kh) + h\Big((a-b)y(kh) - ax(kh)\Big) \end{aligned} \tag{3}$$

where $y(kh)$ is the sampled output, $u(kh)$ is the signal sent to the D-A converter, and $x(kh)$ is the internal state of the controller. This difference equation is updated at each sampling instant. Since the approximation of the derivative by a difference is good if the interval h is small, we can expect the behavior of the computer-controlled system to be close to the continuous-time system. Figure 4 shows the arm position and the control signal for the system when $\omega_0 = 1$ and $h = 0.3$. Notice that the control signal for the computer-controlled system is constant between the sampling instants due to the hold circuit in the D-A converter. Also notice that the difference between the outputs of the two simulations is very small. The difference in the outputs is negligible if h is less than $0.1/\omega_0$.

The computer-controlled system has slightly higher overshoot and the settling time is a little longer. The difference between the systems decreases when the sampling period decreases. When the sampling period increases the computer-controlled system will, however, deteriorate and the closed-loop system becomes unstable for long sampling periods. □

The example shows that it is straightforward to obtain an algorithm for computer control simply by writing the continuous-time control law as a differential equation and approximating the derivatives by differences. The example also indicates that the procedure seems to work well if the sampling period is sufficiently small. The overshoot and the settling time are, however, a little larger for the computer-controlled system, i.e. there is a deterioration due to the approximation.

Is There a Need for Special Design Methods?

The approximation of a continuous-time controller discussed above shows that it is possible to derive a discrete-time controller based on continuous-time design. There are many different approximations methods that can be used. Normally, they are crucially dependent on choosing fairly short sampling periods. The approximation methods are possible to use also for nonlinear continuous-time controllers. Deriving a sampled-data description of the process makes it possible to utilize the full potential of a sampled-data systems and to derive other classes

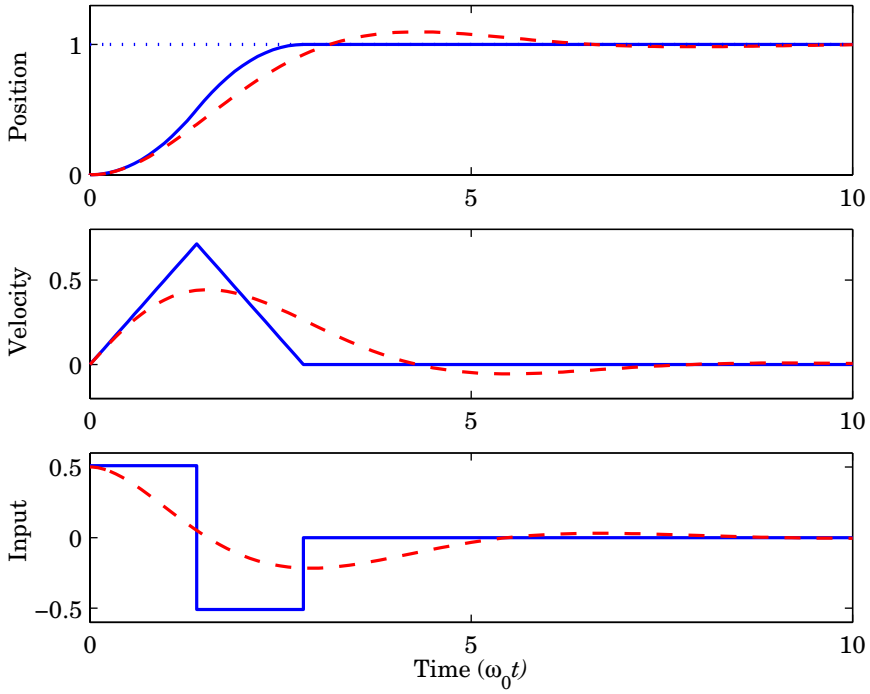

Figure 5 The step response when simulating the disk arm servo using the analog controller (3) (dashed/red) and the deadbeat controller (solid/blue). The sampling period is $h = 1.4$.

of controllers that are not possible to use in continuous-time. The sampled-data theory is also needed to explain the inter-sample and periodic behavior of the sampled-data systems. Notice that the theory for sampled-data systems is derived mainly for linear systems.

EXAMPLE 3—DEADBEAT CONTROL

Consider the disk drive in the previous example. Using a sampling interval of $h = 1.4$ with $\omega_0 = 1$ gives an unstable closed-loop system if the controller (3) is used. However, designing a discrete-time controller with the same structure as (3) gives the performance shown in Figure 5 when $h = 1.4$. The controller can be written as

$$u(kh) = t_0 u_c(kh) + t_1 u_c(kh - h) - s_0 y(kh) - s_1 y(kh - h) - r_1 u(kh - h)$$

where the coefficients now are chosen such that the output of the system will reach the reference value after two samples. This is called a *deadbeat controller* and it has not a continuous-time counterpart. The design parameter of the deadbeat controller is the sampling period h, which here is chosen such that the maximum control signal is the same when the continuous-time and the sampled-data controllers are used. □

Summary

The examples in this chapter clearly show that there are some important issues that are specific for sampled-data systems.

- Sampling makes the system time-varying.

- Information may be lost through sampling.
- Sampled controller have behaviors that are not achievable with continuous-time controllers.

This requires an understanding of the interplay between the continuous-time signals and the discrete-time signals. The rest of the paper gives a brief overview of some of the tools that are important for the analysis, design, and implementation of sampled-data control system.

2. Sampling and Reconstruction

A *sampler* is a device, driven by the clock in the system, that is converting a continuous-time signal into a sequence of numbers. Normally, the sampler is combined into the A-D converter, which also quantizes the sequence of numbers into a finite, although it may be a high, precision number that is then stored in the computer. Typically the A-D converter has 8–16 bits resolution giving 2^8–2^{16} levels of quantization. This is normally a much higher resolution than the precision of the physical sensor. The input to the process is reconstructed from the sequence of numbers from the computer using a D-A converter combined with a *hold circuit* that is determining the input to the process until a new number is delivered from the computer. The hold circuit can be of different kinds. Most common is the *zero-order-hold*, which holds the input constant over the sampling period. Another type of hold device that will be discussed is the *first-order-hold*, where the input is computed from current and previous outputs from the computer using a first order polynomial in time.

Shannon's Sampling Theorem

It is clear that if the signal to be sampled is not changing very fast and if the sampling is done sufficiently often very little should be lost in the sampling procedure. This intuition was formalized by Claude E. Shannon in 1949 in his famous *sampling theorem*. Shannon proved that if the signal contains no frequencies above ω_0 then the continuous-time signal can be uniquely reconstructed from a periodically sampled sequence provided the sampling frequency is higher than $2\omega_0$.

In Figure 2 the sinusoidal signals has the frequencies 0.1 Hz and 0.9 Hz, respectively, while the sampling frequency is 1 Hz. From Shannon's sampling theorem it follows that the slow 0.1 Hz signal can be uniquely reconstructed from its sampled values. To be able to reconstruct the 0.9 Hz signal the sampling frequency must be higher than 1.8 Hz.

The Shannon reconstruction of the continuous-time signal from the sampled values is characterized by the filter

$$h(t) = \frac{\sin(\omega_s t/2)}{\omega_s t/2}$$

The impulse response of this filter is given in Figure 6. The frequency $\omega_N = \omega_s/2$ plays an important role and is called the *Nyquist frequency*. The filter for the Shannon reconstruction is not causal, which makes it impossible to use in practice and simpler and less accurate reconstructions, such as the zero-order hold, are therefore used.

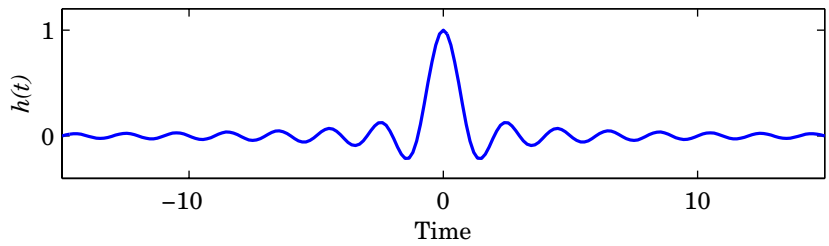

Figure 6 The impulse response of the Shannon reconstruction when $h = 1$.

Aliasing and Antialiasing Filters

The phenomenon shown in Figure 2 that a high frequency signal will be interpreted as a low frequency signal when sampled is called *aliasing* or *frequency folding*. The fundamental alias for a frequency ω_1 is given by

$$\omega = |(\omega_1 + \omega_N) \mod (\omega_s) - \omega_N| \tag{4}$$

Equation (4) implies that the signal ω_1 has an alias in the interval $[0\ \omega_N)$ and that the signal above the Nyquist frequency cannot be reconstructed after the sampling. Equation (4) gives that 0.9 Hz has the alias frequency 0.1 Hz when the sampling frequency is 1 Hz as could be seen in Figure 2.

The implication of the aliasing is that we need to remove all frequencies above the Nyquist frequency before sampling the signal. The simplest way of doing this is to use an analog filter. This type of filters are called *antialiasing filters*. The bandwidth of the filter must be such that the attenuation above the Nyquist frequency is sufficiently high. Bessel filters of orders 2–6 are in practice sufficient to eliminate most of the influence of higher frequencies. A second order Bessel filter with bandwidth ω_B has the transfer function

$$\frac{\omega^2}{(s/\omega_B)^2 + 2\zeta\omega(s/\omega_B) + \omega^2}$$

with $\omega = 1.27$ and $\zeta = 0.87$. Other filters such as Butterworth or ITAE can also be used. The Bessel filters has the property that they can be well approximated by a time delay. This is an advantage in the design of the controller since the dynamics of the antialiasing filter normally has to be included in the design of the sampled-data controller.

EXAMPLE 4—PREFILTERING

The influence of a prefilter is shown in Figure 7. An analog signal consisting of a square-wave and a superimposed sinusoidal disturbance with frequency 0.9 Hz is shown in (a). In (c) the signal in (a) is sampled using a sampling frequency of 1 Hz. This gives rise to an alias frequency of 0.1 that is seen in the sampled signal. The result of filtering the signal in (a) using a sixth-order Bessel filter

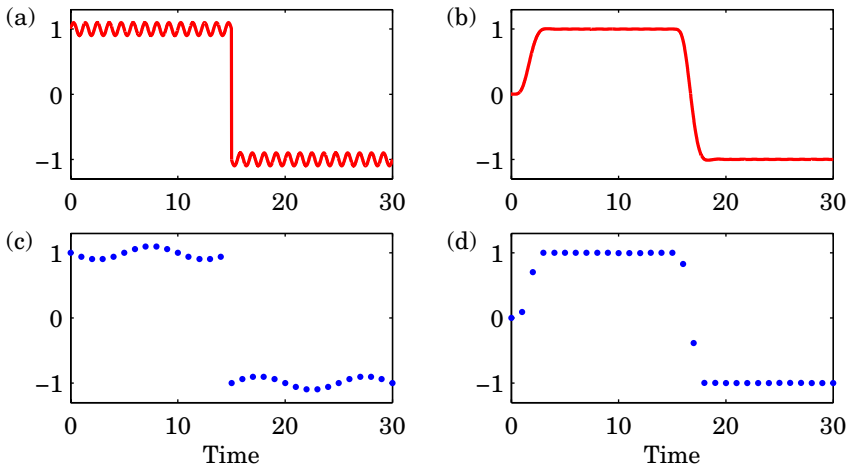

Figure 7 (a) Signal plus sinusoidal disturbance. (b) The signal filtered through a sixth-order Bessel-filter. (c) Sampling of the signal in (a). (d) Sampling of the signal in (b).

with a bandwidth of 0.25 Hz is shown in (b) and the sampling of this signal is given in (d). The disturbance signal is eliminated in (b), but the filter also has an influence on the "useful" signal, i.e. the square-wave is transformed into a smoother signal, which is sampled. □

Zero-order Hold (ZOH)

The simplest and the most common way to make a reconstruction of a sampled-data signal is to let the output of the hold circuit be constant until the next sampled value is obtained. This is called *zero-order hold*, since the continuous time signal is a zeroth order polynomial between the sampling points. The reconstructed signal $f(t)$ is given by

$$f(t) = f(kh) \qquad kh \leq t < kh + h$$

Standard D-A converters are usually constructed such that the output is constant until a new conversion is ordered. The zero-order hold circuit can easily also be used for non-periodic sampling.

First-order Hold

The hold circuit can be made more sophisticated by allowing the continuous-time signal to be a higher-order polynomial between the sampling points. A drawback of the zero-order hold is that the output of the hold circuit is discontinuous. The discontinuities can excite poorly damped mechanical modes of the physical process and also cause wear in the actuators of the system. One way to make the output continuous is to use a *first-order hold*. The signal between the sampling points is now a linear interpolation. The reconstructed signal can be expressed as

$$f(t) = f(kh) + \frac{t - kh}{h}\left(f(kh + h) - f(kh)\right) \qquad kh \leq t < kh + h$$

Notice that this is not a causal reconstruction since $f(kh + h)$ must be available at time kh. The value $f(kh + h)$ can be replaced by a prediction, which easily can be done in a feedback loop.

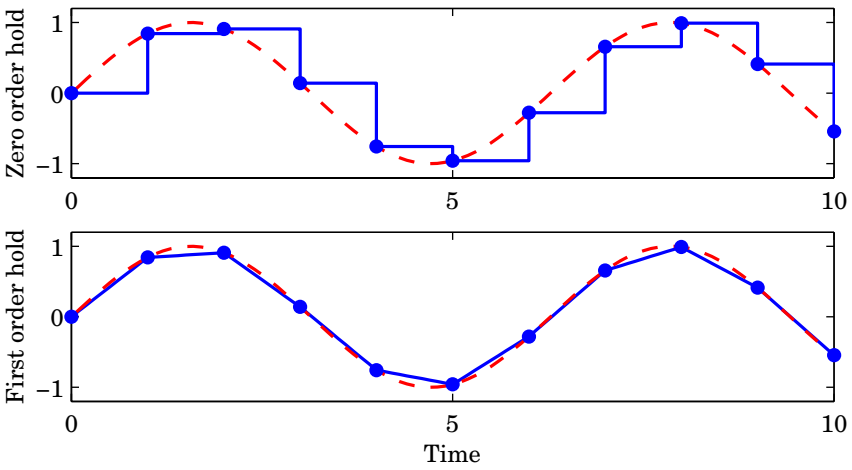

Figure 8 Sampling and reconstruction (solid/blue) of a sinusoidal signal (dashed/red) using a zero-order hold and first-order hold for the sampling period $h = 1$. The sampling times are indicated by dots.

Figure 8 gives a comparison of reconstruction of a sampled sinusoidal signal when using zero-order hold and first-order hold circuits. The figure shows clearly the advantage of using first-order hold when the input signal is smooth.

Summary

- Information can be lost through sampling if the signal contains frequencies higher than the Nyquist frequency.
- Sampling creates new frequencies (aliases).
- Aliasing or frequency folding makes it necessary to filter the signals before they are sampled.
- The dynamics of the antialiasing filters must be considered when designing the sampled-data controller. The dynamics can be neglected if the sampling period is sufficiently short.
- A standard D-A converter can be described as a zero-order hold. There are special converters that give first-order hold. They give a smoother control signal.

3. Mathematical Models

The problem of having a mixture of continuous-time and discrete-time signals can be avoided by observing the sampled-data system only at the sampling points t_k. This implies that the process is looked upon from the view of the computer, i.e. by considering the sampled or discrete-time sequences $u(t_k)$ and $y(t_k)$. This results in a simple description of the sampled-data system, which is sometimes called the *stroboscopic model*. It should, however, be emphasized that the physical process is still a continuous-time system and that we are not getting any information about what is happening between the sampling points. By using the stroboscopic model it is straightforward to derive a system description of the sampled-data system either as a state-space system or as an input-output system.

For simplicity, we assume that periodic sampling is used, i.e. that the sampling instances are $t_k = kh$, and that a zero-order hold is used at the input of the process.

Zero-order Hold Sampling of a Continuous-time System

Assume that the physical process is described by the time-invariant state-space system

$$\begin{aligned} \frac{dx}{dt} &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{aligned} \tag{5}$$

The system has n states, r inputs, and p outputs. Assume that at time kh the state $x(kh)$ is available and that the constant input is applied to the system over the time $kh \le t < kh + h$. On this interval the state is described by

$$\begin{aligned} x(t) &= e^{A(t-kh)}x(kh) + \int_{kh}^{t} e^{A(t-s')}Bu(s')\,ds' \\ &= e^{A(t-kh)}x(kh) + \int_{kh}^{t} e^{A(t-s')}ds'\,Bu(kh) \\ &= e^{A(t-kh)}x(kh) + \int_{0}^{t-kh} e^{As}\,ds\,Bu(kh) \\ &= \Phi(t,kh)x(kh) + \Gamma(t,kh)u(kh) \end{aligned} \tag{6}$$

The second equality follows since $u(t)$ is constant over the sampling interval. Notice that (6) give the change of the state over the sampling period. By considering the situation at the next sampling instance we get

$$\begin{aligned} x(kh+h) &= \Phi x(kh) + \Gamma u(kh) \\ y(kh) &= Cx(kh) + Du(kh) \end{aligned} \tag{7}$$

where

$$\begin{aligned} \Phi &= \Phi(kh+h, kh) = e^{Ah} \\ \Gamma &= \Gamma(kh+h, kh) = \int_{0}^{h} e^{As}\,dsB \end{aligned} \tag{8}$$

At the sampling times the system is described by a linear time-invariant difference equation. The simplification is obtained without any approximations by regarding the system *only* at the sampling points and by assuming that the input is piece-wise constant and is changed synchronized with the sampling of the

signals. The model (7) is called a zero-order hold (ZOH) sampling of the system (5). Since the output $y(kh)$ in most cases is measured before the control signal $u(kh)$ is applied to the system it then follows that $D = 0$. The zero-order hold sampling can also be done for processes containing time-delays.

It follows from (8) that Φ and Γ satisfy the equation

$$\frac{d}{dt}\begin{bmatrix}\Phi(t) & \Gamma(t)\\ 0 & I\end{bmatrix} = \begin{bmatrix}\Phi(t) & \Gamma(t)\\ 0 & I\end{bmatrix}\begin{bmatrix}A & B\\ 0 & 0\end{bmatrix}$$

where I is a unit matrix of the same dimension as the number of inputs. To sample the system we have to calculate a matrix exponential

$$\begin{bmatrix}\Phi(h) & \Gamma(h)\\ 0 & I\end{bmatrix} = exp\left(\begin{bmatrix}A & B\\ 0 & 0\end{bmatrix}h\right)$$

The matrix exponential is defined as a series expansion, but there are commands in scientific computational programs such as Matlab that directly calculates the sampling of a continuous-time system when using different hold devices.

EXAMPLE 5—SAMPLING THE DISK DRIVE SYSTEM

The disk drive system (1) can be written in the state-space form

$$\frac{dx}{dt} = \begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}x + \frac{k}{J}\begin{bmatrix}0\\ 1\end{bmatrix}u$$
$$y = \begin{bmatrix}1 & 0\end{bmatrix}x$$

This gives

$$\Phi = e^{Ah} = I + Ah + A^2h^2/2 + \cdots = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} + \begin{bmatrix}0 & h\\ 0 & 0\end{bmatrix} = \begin{bmatrix}1 & h\\ 0 & 1\end{bmatrix}$$
$$\Gamma = \int_0^h e^{As}B\,ds = \frac{k}{J}\int_0^h \begin{bmatrix}s\\ 1\end{bmatrix}ds = \frac{k}{J}\begin{bmatrix}h^2/2\\ h\end{bmatrix}$$

□

First-order Hold Sampling

A difference equation for the state at the sampling instants can be obtained when the control signal has a specified form over the sampling interval. To illustrate this we will give formulas equivalent to (7) when the input signal is piecewise affine instead of piecewise constant.

Consider a continuous-time system described by (5). Assume that the input signal is linear between the sampling instants. Integration of (5) over one sampling period gives

$$x(kh+h) = e^{Ah}x(kh) + \int_{kh}^{kh+h} e^{A(kh+h-s')}B\left[u(kh) + \frac{s'-kh}{h}\Big(u(kh+h) - u(kh)\Big)\right]ds'$$

Hence

$$\begin{aligned} x(kh+h) &= \Phi x(kh) + \Gamma u(kh) + \frac{1}{h}\,\Gamma_1\Big(u(kh+h) - u(kh)\Big) \\ &= \Phi x(kh) + \frac{1}{h}\,\Gamma_1 u(kh+h) + \Big(\Gamma - \frac{1}{h}\,\Gamma_1\Big)u(kh) \\ y(kh) &= Cx(kh) + Du(kh) \end{aligned} \tag{9}$$

where

$$\begin{aligned} \Phi &= e^{Ah} \\ \Gamma &= \int_0^h e^{As}\,ds\,B \\ \Gamma_1 &= \int_0^h e^{As}(h-s)\,ds\,B \end{aligned} \tag{10}$$

The matrices Φ, Γ and Γ_1 are also given by

$$\begin{pmatrix} \Phi & \Gamma & \Gamma_1 \end{pmatrix} = \begin{pmatrix} I & 0 & 0 \end{pmatrix} \exp\left(\begin{pmatrix} A & B & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{pmatrix} h \right)$$

The discrete system given by (9) is not a standard state-model. Replacing the coordinates in (9) by $\xi = x - \Gamma_1 u(kh+h)/h$, we obtain the standard model

$$\begin{aligned} \xi(kh+h) &= \Phi x(kh) + \Big(\Gamma + \frac{1}{h}\,(\Phi - I)\Gamma_1\Big)u(kh) \\ y(kh) &= C\xi(kh) + (D + \frac{1}{h}\,C\Gamma_1)u(kh) \end{aligned} \tag{11}$$

Notice that the sampled-data system has a direct term even if $D = 0$. First-order hold sampling is particularly useful when approximating continuous transfer functions by sampled systems, because a piecewise affine curve is a good approximation to a continuous function, see Figure 8.

Solution of the System Equation

Assume that the input sequence $u(k_0)$, $u(k_0+1)$, ... and the initial condition $x(k_0)$ are given. The solution to the system equation (7) is then obtained through iterations

$$\begin{aligned} x(k_0+1) &= \Phi x(k_0) + \Gamma u(k_0) \\ x(k_0+2) &= \Phi x(k_0+1) + \Gamma u(k_0+1) \\ &= \Phi^2 x(k_0) + \Phi\Gamma u(k_0) + \Gamma u(k_0+1) \\ &\vdots \\ x(k) &= \Phi^{k-k_0} x(k_0) + \Phi^{k-k_0-1}\Gamma u(k_0) + \cdots + \Gamma u(k-1) \\ &= \Phi^{k-k_0} x(k_0) + \sum_{j=k_0}^{k-1} \Phi^{k-j-1}\Gamma u(j) \end{aligned} \tag{12}$$

The solution consists of two parts, one depending on the initial condition $x(k_0)$ and the other is a weighted sum of the inputs over the interval $[k_0,\ k-1]$. The solution shows that the eigenvalues of Φ play an important role for the development of the states. The influence of the initial condition will decay if the

magnitude of all the eigenvalues of Φ are strictly less than unity, i.e. that they are inside the unit circle. This is thus the condition for stability of the discrete time system (7). The eigenvalues are obtained from the *characteristic equation*

$$\det(\lambda I - \Phi) = 0$$

Operator Calculus and Input-output Descriptions

The discrete-time system (7) is given in state-space form. To derive the input-output description of the system we will use the *forward shift operator* q defined in the following way

$$qf(k) = f(k+1)$$

For convenience the sampling period is chosen as the time unit. The argument of a signal is thus not real time but instead the number of sampling intervals. We call this the *sampling-time convention*. The q should be interpreted as an operator acting on time-functions in the same way as the differential operator is used for continuous-time signals. In shift-operator calculus, all signals are considered as doubly infinite sequences $\{f(k) : k = \cdots -1, 0, 1, \ldots\}$. Multiplying a time-sequence by q implies that the sequence is shifted one step ahead. Multiplying with q^n implies a shift of n steps forward. In the same way multiplying with q^{-n} means a backward shift of n steps. The operator q^{-1} is called the *backward shift operator*.

To simplify the notation it is assumed that the system is a SISO system, i.e. that it has one input and one output. Using the forward shift operator in (7) gives

$$\begin{aligned} x(k+1) = qx(k) &= \Phi x(k) + \Gamma u(k) \\ y(k) &= Cx(k) + Du(k) \end{aligned}$$

Solving for $x(k)$ in the first equation and using that in the second equation gives an elimination of the state variables

$$\begin{aligned} y(k) &= Cx(k) + Du(k) = C(qI - \Phi)^{-1}Bu(k) + Du(k) \\ &= \left[C(qI - \Phi)^{-1}B + D\right] u(k) = H(q)u(k) = \frac{B(q)}{A(q)}u(k) \end{aligned} \tag{13}$$

$H(q)$ is the *pulse-transfer operator* of the system (7) and describes how the input and output are related. The polynomials $A(q)$ and $B(q)$ are defined as

$$\begin{aligned} A(q) &= q^n + a_1 q^{n-1} + \cdots + a_n \\ B(q) &= b_0 q^{nb} + b_1 q^{nb-1} + \cdots + b_{nb} \end{aligned}$$

with $\deg A = n$ and $\deg B = nb \leq n$, where n is the number of states in the system. The solution to $A(q) = 0$ gives the *poles* and $B(q) = 0$ the *zeros* of the system. From (13) it follows that

$$A(q) = \det(qI - \Phi)$$

which is the characteristic polynomial of the matrix Φ. Stability of the system then requires that all the poles should be strictly inside the unit circle.

The system (13) can be written as

$$A(q)y(k) = B(q)u(k)$$

or

$$(q^n + a_1 q^{n-1} + \cdots + a_n)y(k) = (b_0 q^{nb} + \cdots + b_{nb})u(k)$$

which should be interpreted as

$$\begin{aligned} y(k+n) + a_1 y(k+n-1) + \cdots + a_n y(k) \\ = b_0 u(k+nb) + \cdots + b_{nb} u(k) \end{aligned}$$

where $n \geq nb$. The input-output relation is thus a higher order difference equation.

EXAMPLE 6—DISK ARM DRIVE

Consider the sampled-data description of the disk arm drive from Example 5 with $h = 1$ and $k/J = 1$. From (13)

$$H(q) = [\,1 \quad 0\,] \begin{bmatrix} q-1 & -1 \\ 0 & q-1 \end{bmatrix}^{-1} \begin{bmatrix} 0.5 \\ 1 \end{bmatrix} = \frac{0.5(q+1)}{(q-1)^2} = \frac{0.5(q^{-1}+q^{-2})}{1-2q^{-1}+q^{-2}} \qquad (14)$$

The system has two poles in 1 and one zero in -1. □

Z-transform

The Laplace transform is important in the input-output analysis of continuous-time systems. The z-transform was introduced to play a similar role for sampled-data systems and is a convenient tool to handle linear difference equations with or without initial values.

Consider the *discrete-time signal* $\{f(kh) : k = 0, 1, \ldots\}$. The z-transform of $f(kh)$ is defined as

$$\mathcal{Z}\{f(kh)\} = F(z) = \sum_{k=0}^{\infty} f(kh) z^{-k}$$

where z is a complex variable. The inverse transform is given by

$$f(kh) = \frac{1}{2\pi i} \oint F(z) z^{k-1}\, dz$$

where the contour of integration encloses all singularities of $F(z)$. Some properties of the z-transform are collected in Table 1. The z-transform maps semi-infinite time sequences into a function of the complex variable z. It should be remarked that this is a difference with respect to the shift operator which acts on infinite time sequences.

EXAMPLE 7—Z-TRANSFORM OF A RAMP

Consider a ramp signal defined by $y(kh) = kh$ for $k \geq 0$, which has the Laplace transform $Y(s) = 1/s^2$, then

$$Y(z) = 0 + hz^{-1} + 2hz^{-2} + \cdots = h(z^{-1} + 2z^{-2} + \cdots) = \frac{hz}{(z-1)^2}$$

□

Table 1 Some properties of the z-transform.

1. Definition
   $$F(z) = \sum_{k=0}^{\infty} f(kh)z^{-k}$$
2. Inversion
   $$f(kh) = \frac{1}{2\pi i} \oint F(z)z^{k-1}\,dz$$
3. Linearity
   $$\mathcal{Z}\{af + \beta g\} = a\mathcal{Z}f + \beta\mathcal{Z}g$$
4. Time shift
   $$\mathcal{Z}\{q^{-n}f\} = z^{-n}F$$
   $$\mathcal{Z}\{q^{n}f\} = z^{n}(F - F_1) \text{ where } F_1(z) = \sum_{j=0}^{n-1} f(jh)z^{-j}$$
5. Initial-value theorem
   $$f(0) = \lim_{z\to\infty} F(z)$$
6. Final-value theorem
   If $(1 - z^{-1})F(z)$ does not have any poles on or outside the unit circle, then $\lim_{k\to\infty} f(kh) = \lim_{z\to 1}(1 - z^{-1})F(z)$.
7. Convolution
   $$\mathcal{Z}\{f * g\} = \mathcal{Z}\left\{\sum_{n=0}^{k} f(n)g(k-n)\right\} = (\mathcal{Z}f)(\mathcal{Z}g)$$

The Pulse-transfer Function

Consider the system in Figure 9, where there is a zero-order sample-and-hold at the input of the system and a sampler at the output of the process. We now would like to derive the relation between the z-transforms of the sequences $\{u(kh)\}$ and $\{y(kh)\}$. This will be the *pulse-transfer function* $H(z)$ of the sampled-data system, which will correspond to the continuous-time transfer function. The pulse-transfer function is defined through

$$Y(z) = H(z)U(z) \tag{15}$$

where $Y(z)$ and $U(z)$ are the z-transforms of the sequences $y(kh)$ and $u(kh)$, respectively. The description (15) has the advantage that it separates the transform of the output into one part that depends on the input signal, $U(z)$, and one part that characterizes the system, $H(z)$. Table 2 gives the pulse-transfer function $H(z)$ for some continuous-time transfer functions $G(s)$ in Figure 9. Notice

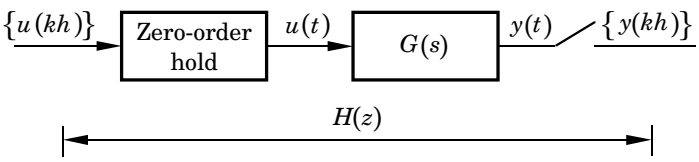

Figure 9 Sampling a continuous-time system.

that the table does not give the z-transforms of the time functions corresponding to the Laplace transforms $G(s)$, but that also the zero-order hold is included in the system.

The z-transform can be used to solve difference equations. Consider (7) for $h = 1$ and take the z-transform of both sides, then

$$\sum_{k=0}^{\infty} z^{-k} x(k+1) = z\left(\sum_{k=0}^{\infty} z^{-k} x(k) - x(0)\right) = \sum_{k=0}^{\infty} \Phi z^{-k} x(k) + \sum_{k=0}^{\infty} \Gamma z^{-k} u(k)$$

Hence

$$z\Big(X(z) - x(0)\Big) = \Phi X(z) + \Gamma U(z)$$
$$X(z) = (zI - \Phi)^{-1}\Big(zx(0) + \Gamma U(z)\Big)$$

and

$$Y(z) = C(zI - \Phi)^{-1} zx(0) + \Big(C(zI - \Phi)^{-1}\Gamma + D\Big)U(z)$$

The solution depends on the initial state and the input signal from time $k = 0$, compare (12). The pulse-transfer function is thus given by

$$H(z) = C(zI - \Phi)^{-1}\Gamma + D$$

which is the same as (13) with q replaced by z. The time sequence $y(k)$ can now be obtained using the inverse transform.

EXAMPLE 8—PULSE-TRANSFER FUNCTION OF THE DISK ARM DRIVE

The disk arm drive with $k/J = 1$ is a double integrator, i.e. $G(s) = 1/s^2$. From Table 2 it follows that the zero-order hold sampling of this system gives the pulse-transfer function

$$H(z) = \frac{h^2(z+1)}{2(z-1)^2}$$

which is the same as the pulse-transfer operator in (14), but with q replaced by z. Notice that this differs from the z-transform of the ramp $1/s^2$ derived in Example 7. The z-transform tables should thus be used with care if we want to find the pulse-transfer function/operator. □

Zero-order Hold Sampling

The pulse-transfer function $H(z)$ for zero-order-hold sampling can be determined directly from the continuous-time transfer function $G(s)$ using the formula

$$\begin{aligned} H_{zoh}(z) &= \frac{z-1}{z}\,\frac{1}{2\pi i}\int_{\gamma-i\infty}^{\gamma+i\infty} \frac{e^{sh}}{z-e^{sh}}\,\frac{G(s)}{s}\,ds \\ &= \sum_{s=s_i} \frac{1}{z-e^{sh}}\,\mathrm{Res}\left\{\frac{e^{sh}-1}{s}\right\} G(s) \end{aligned} \tag{16}$$

where s_i are the poles of $G(s)$ and Res denotes the residue. The second equality requires that the transfer function $G(s)$ goes to zero at least as fast as $|s|^{-1}$ for a large s and has distinct poles, none of which are at the origin. If $G(s)$ has multiple poles or a pole in the origin, the equation must be modified to take multiple poles into consideration.

Table 2 Zero-order hold sampling of a continuous-time system with transfer function $G(s)$. The table gives the pulse-transfer function $H(z)$ that is the zero-order-hold equivalent of the continuous-time system.

$G(s)$	$H(z) = \dfrac{b_1 z^{n-1} + b_2 z^{n-2} + \cdots + b_n}{z^n + a_1 z^{n-1} + \cdots + a_n}$
$\dfrac{1}{s}$	$\dfrac{h}{z-1}$
$\dfrac{1}{s^2}$	$\dfrac{h^2(z+1)}{2(z-1)^2}$
e^{-sh}	z^{-1}
$\dfrac{a}{s+a}$	$\dfrac{1-\exp(-ah)}{z-\exp(-ah)}$
$\dfrac{a}{s(s+a)}$	$b_1 = \dfrac{1}{a}(ah - 1 + e^{-ah})$ $\quad b_2 = \dfrac{1}{a}(1 - e^{-ah} - ahe^{-ah})$ $a_1 = -(1 + e^{-ah})$ $\quad a_2 = e^{-ah}$
$\dfrac{a^2}{(s+a)^2}$	$b_1 = 1 - e^{-ah}(1 + ah)$ $\quad b_2 = e^{-ah}(e^{-ah} + ah - 1)$ $a_1 = -2e^{-ah}$ $\quad a_2 = e^{-2ah}$
$\dfrac{s}{(s+a)^2}$	$\dfrac{(z-1)he^{-ah}}{(z-e^{-ah})^2}$
$\dfrac{ab}{(s+a)(s+b)}$ $a \neq b$	$b_1 = \dfrac{b(1-e^{-ah}) - a(1-e^{-bh})}{b-a}$ $b_2 = \dfrac{a(1-e^{-bh})e^{-ah} - b(1-e^{-ah})e^{-bh}}{b-a}$ $a_1 = -(e^{-ah} + e^{-bh})$ $a_2 = e^{-(a+b)h}$
$\dfrac{\omega_0^2}{s^2 + 2\zeta\omega_0 s + \omega_0^2}$	$b_1 = 1 - \alpha\left(\beta + \dfrac{\zeta\omega_0}{\omega}\gamma\right)$ $\quad \omega = \omega_0\sqrt{1-\zeta^2}$ $\quad \zeta < 1$ $b_2 = \alpha^2 + \alpha\left(\dfrac{\zeta\omega_0}{\omega}\gamma - \beta\right)$ $\quad \alpha = e^{-\zeta\omega_0 h}$ $a_1 = -2\alpha\beta$ $\quad \beta = \cos(\omega h)$ $a_2 = \alpha^2$ $\quad \gamma = \sin(\omega h)$

First-Order-Hold Sampling

The pulse-transfer function for first-order-hold sampling of a system with a transfer function $G(s)$ can also be obtained by a direct calculation. Let u be the input of the system and let y denote the output. The piecewise affine input u can be generated as the output $v(t)$ of an integrator whose input is the piecewise constant signal, i.e.

$$\frac{dv(t)}{dt} = \frac{u(kh+h) - u(kh)}{h} \tag{17}$$

Because the signal at the right hand side of (17) is constant over the sampling intervals, the results of zero-order hold sampling can be applied and we find that the z-transform of the output is given by

$$Y(z) = \mathcal{S}_{\text{zoh}}\left(\frac{G(s)}{s}\right) V(z) \tag{18}$$

where $\mathcal{S}_{\text{zoh}}$ denotes the map of transfer functions to pulse-transfer functions through zero-order-hold sampling. Combining (17) and (18) we get

$$Y(z) = \mathcal{S}_{\text{zoh}}\left(\frac{G(s)}{s}\right) \frac{z-1}{h} U(z)$$

We have thus obtained the input-output relation for sampling with a first-order-hold that can be expressed as follows.

$$\mathcal{S}_{\text{foh}}(G(s)) = \frac{z-1}{h} \mathcal{S}_{\text{zoh}}\left(\frac{G(s)}{s}\right) \tag{19}$$

By using (16) it follows that the pulse-transfer function obtained by first-order-hold sampling of a continuous system with the transfer function $G(s)$ can be expressed by

$$H_{foh}(z) = \frac{(z-1)^2}{zh} \frac{1}{2\pi i} \int_{\gamma-i\infty}^{\gamma+i\infty} \frac{e^{sh}}{z-e^{sh}} \frac{G(s)}{s^2}\, ds$$

The pulse-transfer function for first-order-hold sampling can also be determined by state space methods by sampling the system and taking z-transform. Applying this to the representations (9) and (11) we get the pulse-transfer function

$$\begin{aligned} H(z) &= D + C(zI-\Phi)^{-1}\left(\frac{z}{h}\Gamma_1 + \Gamma - \frac{1}{h}\Gamma_1\right) \\ &= D + \frac{1}{h} C\Gamma_1 + C(zI-\Phi)^{-1}\Big(\Gamma + \frac{1}{h}(\Phi - I)\Gamma_1\Big) \end{aligned}$$

where the matrices Φ, Γ and Γ_1 are given by (10).

EXAMPLE 9—FIRST-ORDER-HOLD SAMPLING OF A DOUBLE INTEGRATOR

A double integrator has the transfer function $G(s) = 1/s^2$. Zero-order-hold sampling of $1/s^3$ gives

$$\frac{h^3}{6}\frac{z^2+4z+1}{(z-1)^3}$$

It then follows from (19) that the first-order-hold sampling of the double integrator is

$$H(z) = \frac{h^2}{6}\frac{z^2+4z+1}{(z-1)^2}$$

Notice that the orders of the numerator and denominator polynomials are the same. This reflects the predictive nature of the first-order-hold. □

Shift-operator Calculus and Z-transforms

In the presentation we have distinguished between the operator q and the complex variable z. The pulse-transfer operator $H(q)$ is the same as the pulse-transfer function $H(z)$ with z replaced by q. With this similarity it would be tempting to use the same notation for both the shift operator and z-transform variable. This is used in many text-books. This is a bad practice because it is important not to confuse operators with complex variables.

Summary

- Sampled systems are time-varying systems because of the sampling mechanism.
- Simple system descriptions are obtained by observing the system at the sampling instants. This leads to a time-invariant sampled-data system if the the continuous-time system is time-invariant.
- Operator and transform theory can be used to define input-output descriptions.
- The system is stable if the system matrix Φ has all eigenvalues inside the unit circle or equivalently that all roots of $A(z)$ are inside the unit circle.

4. Frequency Response

Many powerful methods for analysis and design of control systems are based on frequency response. The key idea is to use the fact that a linear time-invariant system can be completely characterized by its steady-state response to sinusoidal signals. Frequency response is particularly useful for discussions of robustness. The frequency response of a discrete-time system is given by $H(e^{i\omega h})$, where $H(z)$ is the pulse-transfer function of the system. The frequency response of a sampled system is unfortunately much more complicated. The reason is that a sampled system is a linear periodic system. Such a system is infinite dimensional even if the continuous system is finite dimensional. We will first investigate the propagation of sinusoidal signals through a sampled system and we will then briefly discuss the frequency response of a sampled system.

Propagation of Sinusoidal Signals

Consider the system in Figure 10 consisting of an A-D converter, the computer, a D-A converter, and the process. It is assumed that the D-A converter holds the signal constant over the sampling interval. It is also assumed that the calculations performed by the computer can be expressed by the pulse-transfer function $H(z)$ and that the process is described by the transfer function $G(s)$.

The actions of the A-D converter can be represented by a sample and hold mechanism. In continuous time this can be represented by a modulation by an impulse train followed by the filter

$$G_h(s) = \frac{1}{s}\left(1 - e^{-sh}\right) \tag{20}$$

The impulse response of $G_h(s)$ is a pulse of unit amplitude and length h. Modulation can be represented by

$$v^*(t) = v(t)m(t) \tag{21}$$

where the modulation function m is given by

$$m(t) = \sum_{k=-\infty}^{\infty} \delta(t - kh) = \frac{1}{h}\left(1 + 2\sum_{k=1}^{\infty} \cos k\omega_s t\right)$$

The operation (21) followed by the transfer function (20) is a mathematical model of sampling and a zero-order hold. If a sinusoid

$$v(t) = \sin(\omega t + \varphi) = \operatorname{Im}\left(\exp i(\omega t + \varphi)\right)$$

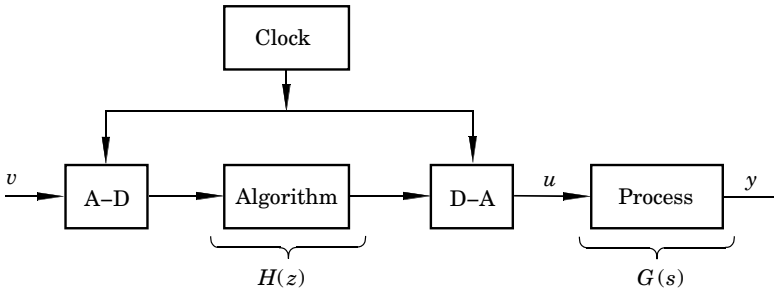

Figure 10 Open-loop computer-controlled system.

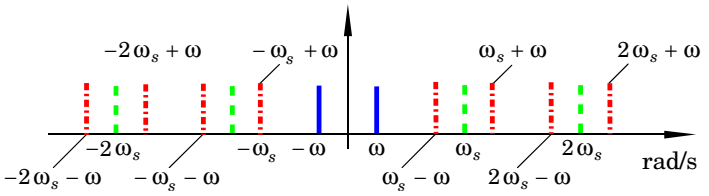

Figure 11 Frequency content of the sampled input signal v^* when $v = \sin(\omega t + \varphi)$.

is applied to the system then the modulated signal is

$$v^*(t) = v(t)m(t) = \frac{1}{h}\left[\sin(\omega t + \varphi) + 2\sum_{k=1}^{\infty}\cos(k\omega_s t)\sin(\omega t + \varphi)\right]$$
$$= \frac{1}{h}\left[\sin(\omega t + \varphi) + \sum_{k=1}^{\infty}\Big(\sin((k\omega_s + \omega)t + \varphi) - \sin((k\omega_s - \omega)t - \varphi)\Big)\right]$$

The signal v^* has a component with the frequency ω of the input signal. This component is multiplied by $1/h$ because the steady-state gain of a sampler is $1/h$. The signal also has components corresponding to the *sidebands* $k\omega_s \pm \omega$. The frequency content of the output v^* of the sampler is shown in Figure 11. The modulation thus creates several frequencies even if a single sinusoid is applied. The propagation of the sinusoids can then be described by standard linear methods. The transmission of the fundamental component may be described by the transfer function

$$K(i\omega) = \begin{cases} \dfrac{1}{h}\, H(e^{i\omega h})G_h(i\omega)G(i\omega) & \omega \neq k\omega_N \\ \dfrac{2}{h}\, H(e^{i\omega h})G_h(i\omega)G(i\omega)e^{i(\pi/2-\varphi)}\sin\varphi & \omega = k\omega_N \end{cases}$$

where ω_N is the Nyquist frequency. When ω is not a multiple of the Nyquist frequency, the signal transmission of the fundamental component can be characterized by a transfer function that is a product of four terms: the gain $1/h$ of the sampler, the transfer function $G_h(s)$ of the hold circuit, the pulse-transfer function $H(\exp(sh))$ of the algorithm in the computer, and the transfer function $G(s)$ of the process. Notice, however, that there are other frequencies in the output of the system because of the sampling. At the Nyquist frequency the fundamental component and the lowest sideband coincide. Close to the Nyquist frequency there will be interference between the fundamental and the side band. We illustrate what happens with an example.

EXAMPLE 10—FREQUENCY RESPONSE OF A SAMPLED-DATA SYSTEM

Consider a system composed of a sampler and a zero-order hold, given by (20), followed by a linear system, with the transfer function

$$G(s) = \frac{1}{s+1}$$

The sampling period is $h = 0.05$ s. The Nyquist frequency is $\pi/0.05 = 62.8$ rad/s. If a sine wave of frequency ω is applied, the output signal is the sum of the outputs of the sine wave and all its aliases. This is illustrated in Figure 12, which shows the steady-state outputs for three different frequencies. For

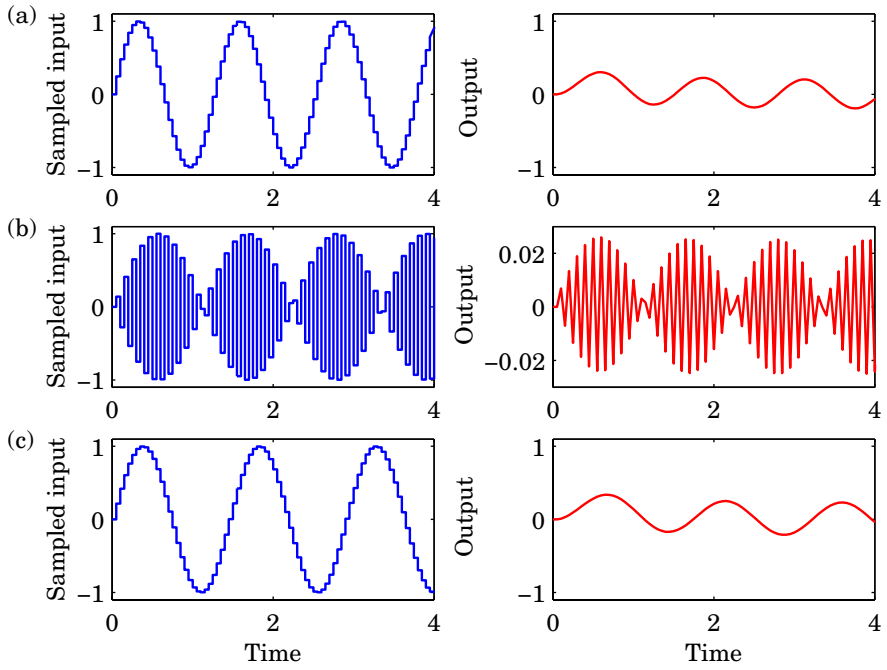

Figure 12 Steady-state responses to sinusoids with different frequencies for a zero-order hold followed by a first-order system with a unit time constant. The sampling period is 0.05 s. The frequencies are 5 rad/s in (a), 60 rad/s in (b), and 130 rad/s in (c).

frequencies smaller than the Nyquist frequency (a), the contribution from the fundamental frequency dominates. At frequencies close to the Nyquist frequency (b), there is a substantial interaction with the first alias, $\omega_s - \omega$. Typical beats are thus obtained. At the Nyquist frequency, the signal and its first alias have the same frequency and magnitude. The resulting signal then depends on the phase shift between the signals. For frequencies higher than the Nyquist frequency (c), the contribution from the alias in the frequency range $(0, \omega_N)$ dominates. □

The example clearly shows how important it is to filter a signal before the sampling, so that the signal transmission above the Nyquist frequency is negligible. If proper antialiasing filters are used the contributions of the aliases will be small. Notice, however, that it is often a problem around the Nyquist frequency. This can to some extent be alleviated by designing the control algorithm so that $H(-1) = 0$, which implies that the signal transmission of the controller at the Nyquist frequency is zero.

Lifting

The notion of *lifting* is an elegant way to deal with periodically sampled systems. The idea is to represent a finite-dimensional sampled system as a time-invariant infinite-dimensional discrete-time system. In this way it is possible to define a notion of frequency response properly. It is also possible to give a nice description of inter-sample behavior.

Consider a system described by (5). Assume that the system is sampled with a period h, and that the input signal and the states are in L_2. We introduce the discrete-time signal $u_k \in L_2(0, h)$ defined by

$$u_k(\tau) = u(kh + \tau) \qquad 0 \leq \tau \leq h$$

and the signals x_k and y_k, which are defined analogously. It follows from (5) that

$$\begin{aligned} x_{k+1}(\tau) &= \varphi(\tau)x_k(h) + \int_0^\tau \psi(\tau - s)Bu_k(s)\,ds \\ y_k(\tau) &= Cx_k(\tau) \end{aligned} \tag{22}$$

where

$$\begin{aligned} \varphi(\tau) &= e^{A\tau} \\ \psi(\tau) &= e^{A\tau}B \end{aligned}$$

This system is a time-invariant discrete-time system. Equation (22) gives a complete description of the inter-sample behavior because the function $y_k(\tau)$, which is defined for $0 \leq \tau \leq h$, is the output in the interval $kh \leq t \leq kh + h$. The description thus includes the phenomenon of aliasing. Notice, however, that u_k, x_k, and y_k are elements of function spaces. Because the system is linear and time-invariant, the frequency response can be defined as $H(e^{i\omega h})$, where H is the transfer function of the infinite-dimensional system (22). The transfer function H is, however, not a simple mathematical object. It can be computed numerically by a finite-dimensional approximation of the state. This can, for example, be obtained through the discrete-time system obtained by dividing the sampling interval h into N equal parts. A complete treatment requires functional analysis and is outside the scope of this paper.

Another way to deal with frequency response of sampled-data systems is to realize that the output generated by a sinusoid with frequency ω_0 contains the frequencies $\omega = k\omega_s \pm \omega_0$. The system can then be properly characterized by an infinite matrix that represents the transmission of all sinusoids and the frequency response can be defined as the norm of that matrix.

Practical Consequences

The fact that sampled systems are time-varying means practically that some care must be exercised when interpreting frequency responses of sampled-data systems. The key difficulty is that injection of a sinusoid will result in an output which is the sum of several sinusoids. There is a correct frequency response obtained by lifting but this is unfortunately quite complicated. The approximate frequency response obtained by neglecting all frequencies except the injected frequency is simple and straightforward to compute. It may, however, give misleading results particularly if there are resonances close to the Nyquist frequency. The maximum sensitivity of the approximate frequency response may be smaller than the true frequency response resulting in wrong estimates of the robustness of the closed-loop system. With an ideal antialiasing filter the signal components with frequencies different from ω_0 will not be present and the difficulty disappears. Ideal antialiasing filters cannot be implemented practically. There will not be much difficulties with plants with good attenuation of high frequencies if the sampling period and the antialiasing filter are chosen properly. There may, however, be severe problems if there are resonant modes close to the Nyquist frequency. In such cases it is necessary to choose sampling rates and antialiasing filters very carefully. An additional safeguard is to make control designs so that the pulse transfer function of the controller has zero gain at the Nyquist frequency, i.e. $H(-1) = 0$. In questionable cases it is advisable to use the theory of lifting to compute the proper frequency responses.

Summary

- Sampled systems are time-varying systems because of the sampling mechanism.
- New frequencies are introduced through the sampling.
- The frequency response of a sampled-data system requires careful interpretation, especially for frequencies close to the Nyquist frequency.

5. Control Design and Specifications

Control system design is a very rich problem area because there are so many factors that have to be considered, for example:

- Attenuation of load disturbances
- Reduction of the effect of measurement noise
- Command signal following
- Variations and uncertainties in process behavior

Design of discrete-time systems is very similar to design of continuous-time system. In this chapter we will summarize some relevant facts. A block diagram of a generic control system is shown in Figure 13. The system is influenced by three external signals: the command signal u_c and the disturbances v and e. The command signal is used to make the system behave in a specified way. The signal v represents load disturbances that drive the process away from its desired behavior. The signal e represents measurement noise that corrupts the information about the process obtained from the sensors. Process disturbances can enter the system in many different ways. It is convenient to consider them as if they enter the system in the same way as the control signal; in this way they will excite all modes of the system. For linear systems it follows from the principle of superposition that inputs entering the system in different ways can be represented by equivalent systems at the process input. Measurement noise is injected into the process by feedback. Load disturbances typically have low frequencies and measurement noise has high frequencies.

Control problems can broadly speaking be classified into *regulation problems* and *servo problems*. Model uncertainty is essential for both problems. The major issue in regulation is to compromise between reduction of load disturbances and injection of measurement noise. The key issue in the servo problem is to make the output respond to command signals in the desired way.

The key ingredients of a design problem are

- Purpose of the system
- Process model
- Model of disturbances
- Model variations and uncertainties
- Admissible control strategies
- Design parameters

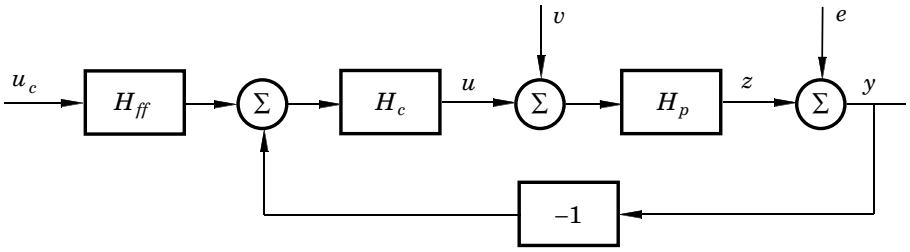

Figure 13 Block diagram of a typical control system.

It is difficult to find design methods that consider all aspects of a design problem. Most design methods focus on one or two aspects of the problem and the control-system designer then has to check that the other requirements are also satisfied. To do this it is necessary to consider the signal transmission from command signals, load disturbances, and measurement noise to process variables, measured signals, and control signals, see Figure 13. There are today many computer tools available for design of continuous-time as well as sampled-data controllers. Software packages, such as Matlab, have many commands and toolboxes aimed at analysis and design of control systems. In the following chapters we will give an overview of several methods for design of sampled-data controllers. It turns out that all methods result in systems having very similar structure. Before going into the details of the design methods we will give an overview of the structure of a controller.

The Process

It is assumed that the process to be controlled can be described by the model

$$\frac{dx}{dt} = Ax + Bu \tag{23}$$

where u represents the control variables, x represents the state vector, and A and B are constant matrices. Further, only the single-input–single-output case will be discussed. Because computer control is considered, the control signals will be constant over sampling periods of constant length. Sampling the system in (23) gives the discrete-time system

$$x(kh + h) = \Phi x(kh) + \Gamma u(kh)$$

where the matrices Φ and Γ are given by (8). To simplify we use the sampling-time convention and write the system as

$$x(k + 1) = \Phi x(k) + \Gamma u(k)$$

Admissible Controls

It is important to specify the information available for generating the control signal. A simple situation is when all state variables are measured without error. The general linear controller is then a linear feedback from all states, i.e.

$$u(k) = -Lx(k) \tag{24}$$

This feedback is called state feedback. It is important conceptually but is rarely used in practice because of the cost or difficulty of measuring all the states. A more common situation is that the control signal is a function of measured output signals, past control signals, and past and present reference signals.

Design Parameters

All design problems are compromises. It is often practical to have a few parameters that make the compromise explicit. These parameters are called *design parameters*. The design parameters make it possible to fine tune the design on line. Typical design parameters relate to trade-offs between attenuation of load disturbances and injection of measurement noise for regulation problems or response speed and model uncertainty for servo problems.

Criteria and Specifications

The closed loop system in Figure 13 has three inputs u_c, v and e and three outputs z, y and u. It can be shown that the system is completely characterized by six transfer functions.

$$\begin{aligned} -H_{ze} = T = \frac{H_p H_c}{1 + H_p H_c} \quad -H_{ue} = \frac{H_c}{1 + H_p H_c} \qquad H_{zu_c} = \frac{H_p H_c H_{ff}}{1 + H_p H_c} \\ H_{zv} = \frac{H_p}{1 + H_p H_c} \qquad H_{ye} = S = \frac{1}{1 + H_p H_c} \quad H_{uu_c} = \frac{H_c H_{ff}}{1 + H_p H_c} \end{aligned} \tag{25}$$

Specifications on the closed loop system are thus naturally expressed in terms of these transfer functions. Notice that it is necessary to give specifications on all six transfer functions. The specifications can be given in the time or frequency domain.

Robustness

Robustness to process variations are well captured by the sensitivity function and the complementary sensitivity function, i.e.

$$S(z) = \frac{1}{1 + H_p(z)H_c(z)} \quad \text{and} \quad T(z) = \frac{H_p(z)H_c(z)}{1 + H_p(z)H_c(z)}$$

Typical criteria are the maximum values

$$M_s = \max_{\omega} |S(e^{i\omega h})| \quad \text{and} \quad M_t = \max_{\omega} |T(e^{i\omega h})| \tag{26}$$

where the values of M_s and M_t typically should be in the range of 1.2 to 2.0. The smaller values give more robust closed loop systems. In critical situations the values should be replaced by the exact frequency responses, see Chapter 4.

Attenuation of Load Disturbances

The pulse-transfer function from load disturbance to process output is

$$H_{zv}(z) = \frac{H_p(z)}{1 + H_p(z)H_c(z)}$$

Since load disturbances typically have low frequencies, the low frequency behavior is most important. Since the loop transfer function is also large at low frequencies we have approximately

$$H_{zv}(z) = \frac{H_p(z)}{1 + H_p(z)H_c(z)} \approx \frac{1}{H_c(z)}$$

The low frequency gain of the controller is thus a good measure of load disturbance attenuation. Other measures are the the maximum value or some some weighted integral of $|H_{zv}(e^{i\omega h})|$. See Chapter 10.

Injection of Measurement Noise

The pulse-transfer function from measurement noise to controller output is

$$H_{ue}(z) = -\frac{H_c(z)}{1 + H_p(z)H_c(z)} = -S(z)H_c(z)$$

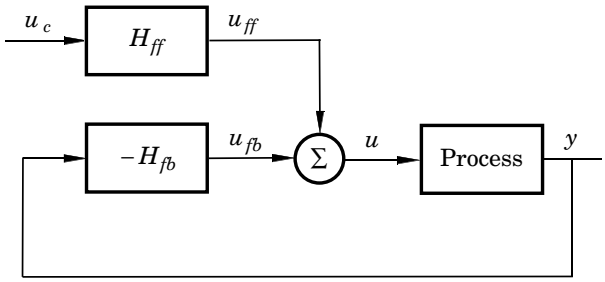

Figure 14 Block diagram of a feedback system with a two-degree-of-freedom structure.

Since measurement noise typically has high frequencies, the high frequency behavior is most important. For high frequencies we have approximately

$$H_{ue}(z) = -\frac{H_c(z)}{1 + H_p(z)H_c(z)} \approx -M_s H_c(z)$$

The high frequency gain of the controller is thus a good measure of measurement noise attenuation. Other measures are some weighted integral of $|H_{ue}(e^{i\omega h})|$ or the maximum of this quantity for high frequencies.

Command Signals Following

The response to command signals can be decoupled from the response to disturbances by using a controller having two degrees of freedom. This is illustrated in Figure 14. The reason that this controller is said to have two degrees of freedom is that the signal transmission from output y to control u is different from the signal transmission from command signal u_c to control u. This configuration has the advantage that the servo and regulation problems are separated. The feedback controller $-H_{fb}$ is designed to obtain a closed-loop system that is insensitive to process disturbances, measurement noise, and process uncertainties. The feedforward compensator H_{ff} is then designed to obtain the desired servo properties.

Summary

- The design can be separated into regulation and servo problems.
- The regulator problem is solved using feedback.
- The servo problem can mainly be solved using feedforward.
- The sensitivity S and complementary sensitivity T functions are important measures of the robustness of the closed loop system.
- Good software packages are available for design of control systems.

6. Approximation of Analog Controllers

There are situations when a continuous-time controller is already available. A typical case is when an analog system is replaced by a computer controlled system. It is then natural to convert the continuous-time controller to a discrete-time controller directly. In all other situations it is more convenient to make the design directly in the sampled-data domain.

A straightforward approach to approximate an analog controller is to use a short sampling interval and to make some discrete-time approximations of the continuous-time controller, compare Example 2. The problem is illustrated in Figure 15. The continuous-time controller is a dynamical system which can be given in terms of a state model or a transfer function. It is desired to find an algorithm for a computer so that the digital system approximates the continuous-time system. The algorithm can be given in terms of a difference equation or a pulse-transfer function. This problem is interesting for implementation of both analog controllers and digital filters. The approximation may be done in many different ways. Digital implementation includes a data reconstruction, which also can be made in different ways—for example, zero- or first-order hold.

State Model of the Controller

A controller is a dynamical system. We will first consider the case when the controller is specified as a state model. To simplify the notation assume that the controller $G(s)$ in Figure 15 is represented as a generic dynamic system

$$\begin{aligned} \frac{dx}{dt} &= Ax + Bu \\ y &= Cx + Du \end{aligned} \tag{27}$$

where x is the controller state, which consists of the observer state and additional states that are introduced to generate feedforward signals and integral action.

A simple way of converting the differential equation to a difference equation is to approximate the derivative of the state by a difference. There are two simple choices, a *forward difference*, also called *Eulers's method*

$$\frac{dx(t)}{dt} = px(t) \approx \frac{x(t+h) - x(t)}{h} = \frac{q-1}{h}\, x(t) \tag{28}$$

or a backward difference.

$$\frac{dx(t)}{dt} = px(t) \approx \frac{x(t) - x(t-h)}{h} = \frac{q-1}{qh}\, x(t) \tag{29}$$

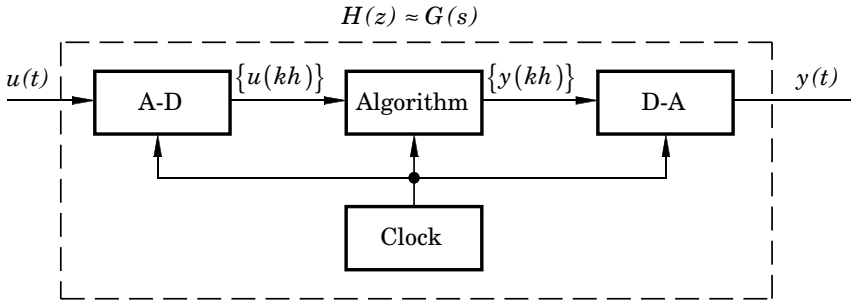

Figure 15 Approximating a continuous-time transfer function, $G(s)$, using a computer.

where p is the differential operator. Using these approximations we find that (27) can be approximated by

$$\begin{aligned} x(t+h) &= (I + hA)x(t) + hB(t) \\ u(t) &= Cx(t) + Du(t) \end{aligned}$$

when forward differences are used and

$$\begin{aligned} x(t+h) &= (I - hA)^{-1}(x(t) + Bu(t+h)) \\ u(t) &= Cx(t) + Du(t) \end{aligned}$$

when backward differences are used. A nice feature of these approximations is that there is a close correspondence between the continuous and discrete versions of the controllers. The scheme works very well if the sampling frequency is sufficiently high compared with the frequency content of the control signal.

More elaborate schemes are to compute the zero-order hold or the first-order-hold equivalences. The first-order-hold is probably the most natural approximation because the continuous analysis assumes that the signals are continuous. A sophisticated scheme is to assume that the measured signal is piecewise linear and that the control signal is piecewise constant. It is a straight forward extension to derive the appropriate formulas in these cases.

Transfer Functions

When the controller is specified as a transfer function it is natural to look for methods that will transform a continuous transfer function $G(s)$ to a pulse-transfer function $H(z)$ so that the systems in Figure 15 are close to each other. A transfer function represents a differential equation. The approximations given by (28) and (29) imply that the differential operator p is replaced by some difference approximation. In the transform variables, this corresponds to replacing s by $(z-1)/h$ or $(z-1)/zh$. The variables z and s are related as $z = \exp(sh)$. The difference approximations correspond to the series expansions

$$z = e^{sh} \approx 1 + sh \qquad \text{(Forward difference or Euler's method)}$$

$$z = e^{sh} \approx \frac{1}{1 - sh} \qquad \text{(Backward difference)}$$

Another approximation, which corresponds to the trapezoidal method for numerical integration, is

$$z = e^{sh} \approx \frac{1 + sh/2}{1 - sh/2} \qquad \text{(Trapezoidal method)} \tag{30}$$

In digital-control context, the approximation in (30) is often called *Tustin's approximation*, or the *bilinear transformation*. Using the approximation methods above, the pulse-transfer function $H(z)$ is obtained simply by replacing the argument s in $G(s)$ by s', i.e. $H(z) = G(s')$ where

$$s' = \frac{z-1}{h} \qquad \text{(Forward difference or Euler's method)} \tag{31}$$

$$s' = \frac{z-1}{zh} \qquad \text{(Backward difference)} \tag{32}$$

$$s' = \frac{2}{h} \cdot \frac{z-1}{z+1} \qquad \text{(Tustin's approximation)} \tag{33}$$

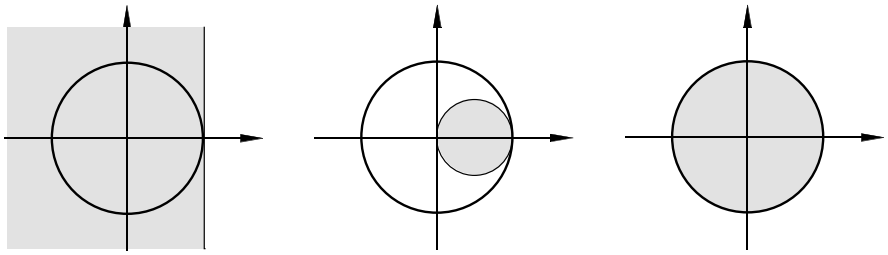

Figure 16 Mapping of the stability region in the s-plane on the z-plane for the transformations (31), (32), and (33).

The methods are very easy to apply even for hand calculations. Figure 16 shows how the stability region $\operatorname{Re} s < 0$ in the s-plane is mapped on the z-plane for the mappings (31), (32), and (33).

With the forward-difference approximation it is possible that a stable continuous-time system is mapped into an unstable discrete-time system. When the backward approximation is used, a stable continuous-time system will always give a stable discrete-time system. There are, however, also unstable continuous-time systems that are transformed into stable discrete-time systems. Tustin's approximation has the advantage that the left half-s-plane is transformed into the unit disc. Stable continuous-time systems are therefore transformed into stable discrete-time systems, and unstable continuous-time systems are transformed into unstable discrete-time systems.

Frequency Prewarping

One problem with the approximations discussed above is that the frequency scale is distorted. For instance, if it is desired to design band-pass or notch filters, the digital filters obtained by the approximations may not give the correct frequencies for the band-pass or the notches. This effect is called *frequency warping*. Consider an approximation obtained by Tustin's approximation. The transmission of sinusoids for the digital filter is given by

$$H(e^{i\omega h}) = \frac{1}{i\omega h}\,(1 - e^{-i\omega h})\,G\left(\frac{2}{h}\cdot\frac{e^{i\omega h}-1}{e^{i\omega h}+1}\right)$$

The first two factors are due to the sample-and-hold. The argument of G is

$$\frac{2}{h}\,\frac{e^{i\omega h}-1}{e^{i\omega h}+1} = \frac{2}{h}\,\frac{e^{i\omega h/2}-e^{-i\omega h/2}}{e^{i\omega h/2}+e^{-i\omega h/2}} = \frac{2i}{h}\,\tan\left(\frac{\omega h}{2}\right)$$

The frequency scale is thus distorted. Assume, for example, that the continuous-time system blocks signals at the frequency ω'. Because of the frequency distortion, the sampled system will instead block signal transmission at the frequency ω, where

$$\omega' = \frac{2}{h}\,\tan\left(\frac{\omega h}{2}\right)$$

That is,

$$\omega = \frac{2}{h}\,\tan^{-1}\left(\frac{\omega' h}{2}\right) \approx \omega'\left(1 - \frac{(\omega' h)^2}{12}\right) \tag{34}$$

This expression gives the distortion of the frequency scale (see Figure 17). It

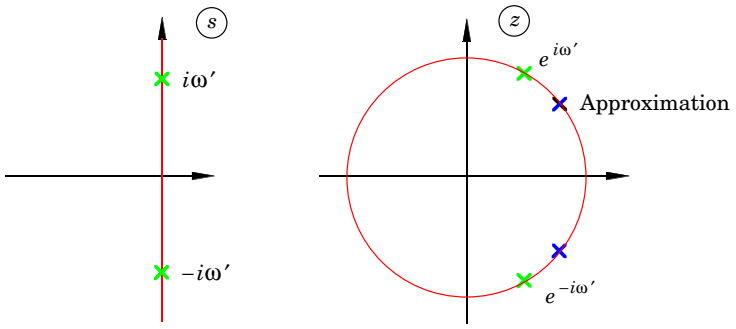

Figure 17 Frequency distortion (warping) obtained with approximation.

follows from (34) that there is no frequency distortion at $\omega = 0$ and that the distortion is small if ωh is small. It is easy to introduce a transformation that eliminates the scale distortion at a specific frequency ω_1 by modifying Tustin's transformation from (33) to the transformation

$$s' = \frac{\omega_1}{\tan(\omega_1 h/2)} \cdot \frac{z-1}{z+1} \qquad \text{(Tustin with prewarping)} \tag{35}$$

From (35), it follows that

$$H\left(e^{i\omega_1 h}\right) = G(i\omega_1)$$

that is, the continuous-time filter and its approximation have the same value at the frequency ω_1. There is, however, still a distortion at other frequencies.

EXAMPLE 11—FREQUENCY PREWARPING
Assume that the integrator

$$G(s) = \frac{1}{s}$$

should be implemented as a digital filter. Using the transformation of (33) without prewarping gives

$$H_T(z) = \frac{1}{\frac{2}{h} \cdot \frac{z-1}{z+1}} = \frac{h}{2} \cdot \frac{z+1}{z-1}$$

Prewarping gives

$$H_P(z) = \frac{\tan(\omega_1 h/2)}{\omega_1} \cdot \frac{z+1}{z-1}$$

The frequency function of H_P is

$$H_P(e^{i\omega h}) = \frac{\tan(\omega_1 h/2)}{\omega_1} \cdot \frac{e^{i\omega h}+1}{e^{i\omega h}-1} = \frac{\tan(\omega_1 h/2)}{\omega_1} \cdot \frac{1}{i\tan(\omega h/2)}$$

thus $G(i\omega)$ and $H_P\left(e^{i\omega h}\right)$ are equal for $\omega = \omega_1$. □

Step Invariance

Another way to approximate is to use the idea of sampling a continuous-time system. In this way it is possible to obtain approximations that give correct values at the sampling instants for special classes of input signals. For example,

if the input signal is constant over the sampling intervals (16) gives a pulse-transfer function $H(z)$ that corresponds to a transfer function G(s) that gives correct values of the output when the input signal is a piecewise constant signal that changes at the sampling instants. This approximation is therefore called *step invariance*.

Ramp Invariance

The notion of step invariance is ideally suited to describe a system where the input signal is generated by a computer, because the input signal is then constant over the sampling period. The approximation is, however, not so good when dealing with input signals that are continuous. In this case it is much better to use an approximation where the input signal is assumed to vary linearly between the sampling instants. The approximation obtained is called *ramp invariance* because it gives the values of the output at the sampling instants exactly for ramp signals and it is identical to first-order-hold sampling.

Comparison of Approximations

The step-invariant method is not suitable for approximation of continuous-time transfer functions. The reason is that the approximation of the phase curve is unnecessarily poor. Both Tustin's method and the ramp-invariant method give better approximations. Tustin's method is a little simpler than the ramp-invariant method. The ramp-invariant method does give correct sampled poles. This is not the case for Tustin's method. This difference is particularly important when implementing notch filters where Tustin's method gives a frequency distortion. Another drawback with Tustin's method is that very fast poles of the continuous-time system transform into sampled poles close to $z = -1$, which will give rise to ringing in the digital filter. The different approximations are illustrated by an example.

EXAMPLE 12—SAMPLED APPROXIMATIONS OF TRANSFER FUNCTION
Consider a continuous-time system with the transfer function

$$G(s) = \frac{(s+1)^2(s^2+2s+400)}{(s+5)^2(s^2+2s+100)(s^2+3s+2500)}$$

Let $H(z)$ be the pulse-transfer function representing the algorithm in Figure 15. The transmission properties of the digital filter in Figure 15 depend on the nature of the D-A converter. If it is assumed that the converter keeps the output constant between the sampling periods, the transmission properties of the filter are described by

$$\hat{G}(s) = \frac{1}{sh}\,(1 - e^{-sh})H(e^{sh})$$

where the pulse-transfer function H depends on the approximation used. Figure 18 shows Bode diagrams of H for the different digital filters obtained by step equivalence, ramp equivalence, and Tustin's method. The sampling period is 0.03 s in all cases. This implies that the Nyquist frequency is 105 rad/s. All methods except Tustin's give a good approximation of the amplitude curve. The frequency distortion by Tustin's method is noticeable at the notch at 20 rad/s and very clear at the resonance at 50 rad/s.

The step-equivalence method gives a small error in the gain but a significant error in the phase. The phase error corresponds approximately to a time delay of half a sampling interval. The approximation based on ramp equivalence gives

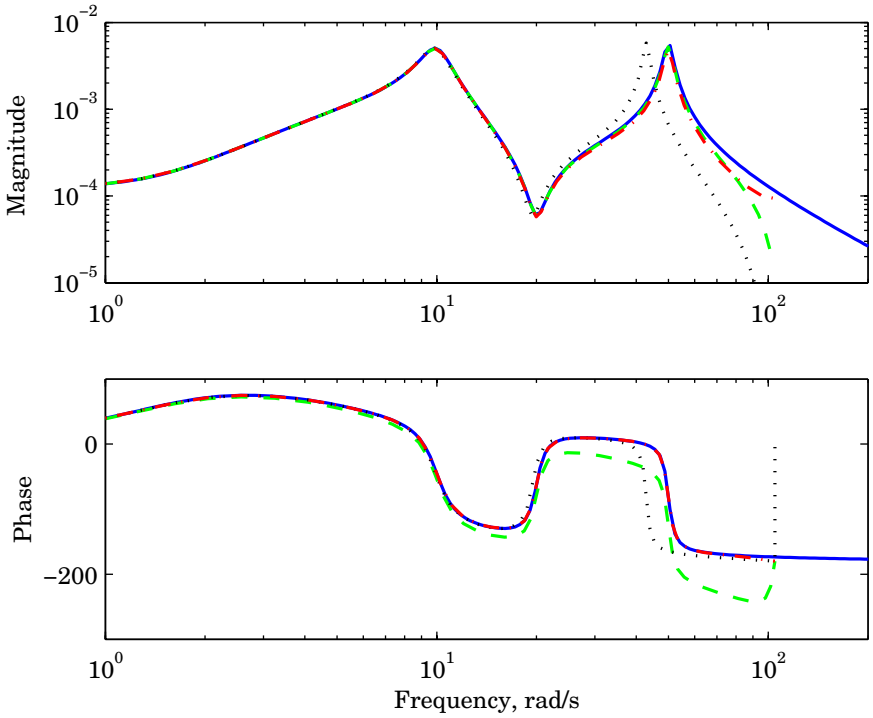

Figure 18 Bode diagrams of a continuous-time transfer function $G(s)$ and different sampled approximations $H(e^{sh})$, continuous-time transfer function (solid/blue), ramp invariance (dashed-dotted/red), step invariance (dashed/green), and Tustin's approximation (dotted/black).

very small errors in gain and phase. The phase curve for Tustin's approximation also deviates because of the frequency warping. Ramp equivalence gives the best approximation of both amplitude and phase. □

Selection of Sampling Interval and Antialiasing Filters

Choice of sampling rates and antialiasing filters are important issues in design of digital control. It is also necessary to augment process dynamics with the dynamics of the antialiasing filter. The choice of sampling period depends on many factors. One way to determine the sampling period is to use continuous-time arguments. The sampled system can be approximated by the hold circuit, followed by the continuous-time system. For small sampling periods, the transfer function of the hold circuit can be approximated as

$$\frac{1-e^{-sh}}{sh} \approx \frac{1-1+sh-(sh)^2/2+\cdots}{sh} = 1-\frac{sh}{2}+\cdots$$

The first two terms correspond to the Taylor series expansion of $\exp(-sh/2)$. That is, for small h, the hold can be approximated by a time delay of half a sampling interval. The antialiasing filter will also decrease the phase margin. Consider a second order filter with transfer function

$$G_f(s) = \frac{\omega_f^2}{s^2+2\zeta_f\omega_f s+\omega_f^2}$$

The gain of the filter at the Nyquist frequency ω_N is approximately

$$g_N = \left(\frac{\omega_f}{\omega_N}\right)^2$$

Hence $\omega_f = \omega_N\sqrt{g_N}$. The phase lag of the filter is approximately

$$\arg G_f(i\omega) \approx \frac{2\zeta_f\omega}{\omega_f} = \frac{2\zeta_f\omega}{\omega_N\sqrt{g_N}} = \frac{2\zeta_f\omega h}{\pi\sqrt{g_N}}$$

Summarizing we find that the hold circuit and the antialiasing filter decrease the phase margin with

$$\varphi = \left(0.5 + \frac{2\zeta_f}{\pi\sqrt{g_N}}\right)\omega_c h$$

Assuming that the phase margin can be decreased by 5° to 15° and introducing the values $\zeta_f = 0.707$ and $g_N = 0.1$ the following rule of thumb is obtained.

$$h\omega_c \approx 0.05 \text{ to } 0.14 \tag{36}$$

where ω_c is the crossover frequency (in radians per second) of the continuous-time system. This rule gives a Nyquist frequency that is about 23 to 70 times higher than the crossover frequency.

Summary

Different ways of translating a continuous-time controller to a digital controller have been presented. These methods make it possible to use continuous-time design and simply translate the results to a pulse-transfer function that can be implemented in a computer. This approach is particularly useful when continuous-time designs are available.

- Forward, backward differences, and Tustins's method are the simplest.
- Tustin's method distort the frequency scale of the filter.
- Ramp invariance gives very good results and is only moderately more complicated than Tustin's method. With computer tools like Matlab there is no reason to use any other method.
- The sampling period should be short to have good approximations $\omega_c h \approx 0.05 - 0.14$.
- Antialiasing filters are important in all cases.

7. Feedforward Design

Before treating design methods in detail we will discuss the structure of the controllers. Figure 14 is an input-output representation which only gives an overview of the system. Additional insight can be obtained by considering the state space representation in Figure 19 which gives more details. The controller is represented by three blocks: a generator for feedforward signals, an observer and a state feedback L. The feedforward generator has the command signal as the input and it generates the desired behavior of the states x_m of the system and the feedforward signal u_{ff}. The feedforward signal u_{ff} produces an input to the process that will generate the desired response to the command signal. The feedforward generator can be implemented as a dynamical system

$$\begin{aligned} x_m(k+1) &= \Phi_m x_m(k) + \Gamma_m u_c(k) \\ u_{ff}(k) &= C_m x_m(k) + D_m u_c(k) \end{aligned}$$

The block called observer also has two inputs, the process input u and output y, and it generates an estimate of the state of the process. The observer is also a dynamical system

$$\hat{x}(k+1) = \Phi\hat{x}(k) + \Gamma u(k) + K(y(k) - C\hat{x}(k)) \tag{37}$$

The estimated state $\hat{x}$ is compared with the ideal state from the feedforward generator creating the feedback signal

$$u_{fb}(k) = L\Big(x_m(k) - \hat{x}(k)\Big)$$

The feedback signal $L(x_m - \hat{x})$ is zero if the error $e = x_m - \hat{x}$ is zero. If there is a difference between $\hat{x}$ and x_m, the feedback will generate corrective actions. The feedback term can be viewed as a generalization of error feedback in ordinary control systems, because the error represents deviations of all state variables and not just the output errors. The control signal is the sum of the feedback and the feedforward signals, i.e.

$$u(k) = u_{fb} + u_{ff} = L\Big(x_m(k) - \hat{x}(k)\Big) + u_{ff}$$

Many design methods result in the controller structure represented by Figure 19. There are also several refinements. It is, for example, possible to include models of the disturbances acting on the system.

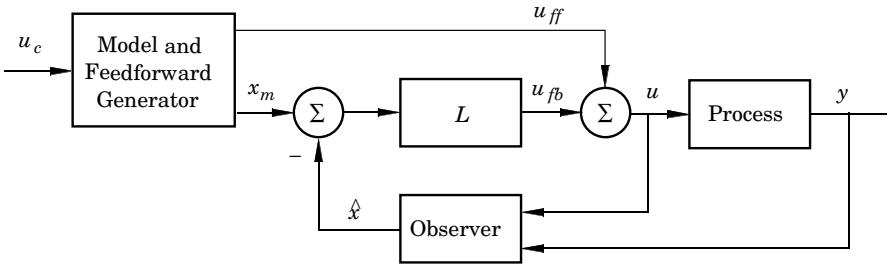

Figure 19 A two-degree-of-freedom controller based on state feedback and an observer.

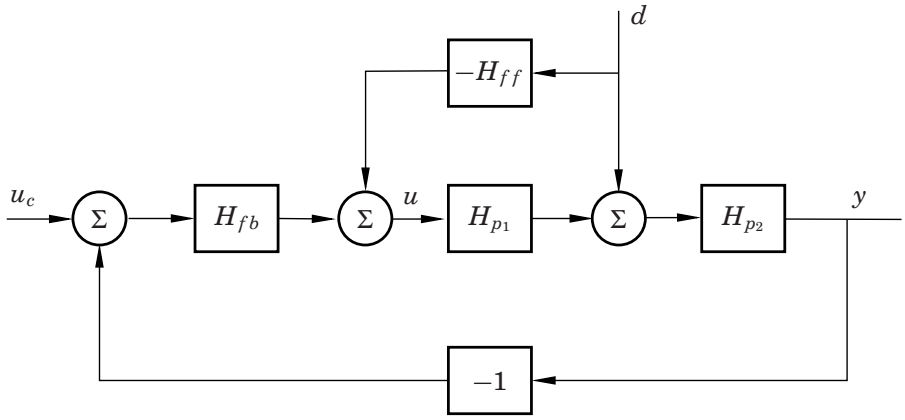

Figure 20 Block diagram of a system where measured disturbance d is reduced by a combination of feedback and feedforward.

Reduction of Measurable Disturbances by Feedforward

Feedforward is a complement to feedback which can be used both to reduce the effect of measurable disturbances and to improve the response to command signals as is illustrated in Figure 20. Assuming that the disturbance d is piecewise constant over the sampling intervals and sampling the system we find that the pulse-transfer function from load disturbance to process output is

$$\frac{Y(z)}{D(z)} = \frac{H_{p_2}(z)\left(1 - H_{p_1}(z)H_{ff}(z)\right)}{1 + H_{fb}(z)H_p(z)} = H_{p_2}(z)\left(1 - H_{p_1}(z)H_{ff}(z)\right)S(z)$$

This equation shows that the effects of the disturbances can be reduced in two ways: by making $1 - H_{p_1}(z)H_{ff}(z)$ (feedforward) small or by making the sensitivity function $S(z)$ small (feedback). To totally eliminate the disturbances by feedforward the feedforward transfer function should be chosen as

$$H_{ff}(z) = H_{p_1}^{-1}(z) \tag{38}$$

which means that the process dynamics should be inverted.

System Inverses

Equation (38) shows that design of feedforward is essentially a problem of inverting a system. It is particularly simple to invert a system if it is given as a pulse-transfer function. Consider a system given by the pulse-transfer function

$$H(z) = \frac{B(z)}{A(z)}$$

The inverse is then

$$H^{-1}(z) = \frac{A(z)}{B(z)}$$

There are, however, some problems because the inverse is not causal if the degree of the polynomial $A(z)$ is greater than the degree of $B(z)$. Further, the inverse is unstable if the polynomial $B(z)$ has zeros outside the unit circle. For feedforward design it must be required that the feedforward transfer function is stable and causal. It is therefore often necessary to replace the exact inverse of the transfer

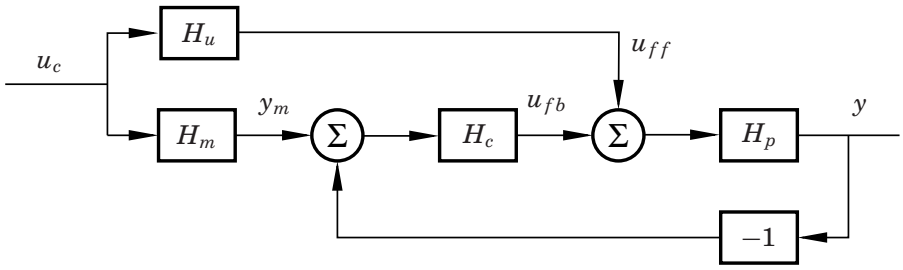

Figure 21 Another representation of the system with two-degrees-of-freedom in Figure 19.

function $H_{p_1}^{-1}(z)$ with an approximate inverse $H_{p_1}^{\dagger}(z)$. If the disturbance d has know properties it is possible to compute optimal inverses.

The following procedure gives an inverse that is optimal under certain conditions. Write the transfer function $H(z)$ as

$$H(z) = \frac{B^+(z)B^-(z)}{A(z)}$$

where the polynomial $B^+(z)$ has all its zeros inside the unit disc and $B^-(z)$ has all its zeros outside the unit disc. An approximate inverse is then given by

$$H(z) = \frac{A(z)}{z^{\deg A - \deg B} B^+(z) B^{*-}(z)}$$

where

$$B^{*-}(z) = z^{\deg B^-} B^-(z^{-1})$$

Using Feedforward to Improve Response to Command Signals

Feedforward can also be used to improve the response to command signals. This has already been discussed in connection with two-degrees-of-freedom controllers, see Figure 19. An alternative representation of this figure is given in Figure 21. In this figure H_m represents the desired response of the output to the command signal u_c, and H_u is the transfer function that generates the feedforward signal. From Figure 21

$$y(k) = \frac{H_p(H_u + H_c H_m)}{1 + H_p H_c} u_c(k) = H_m u_c(k)$$

This gives the following relation

$$H_m = H_p H_u$$

which implies that

$$H_u = H_p^{-1} H_m$$

The pulse-transfer function H_m must clearly be chosen to be causal and stable. A pulse-transfer function H_u that is also causal and stable can be obtained by choosing H_m to have the same pole excess and the same zeros outside the unit circle as H_p.

Summary

- Feedforward is efficient for reducing measurable disturbances.
- Feedforward is used to shape the response from the reference signal.
- Feedforward requires inversion of the open loop system and the inverse often requires approximations to be causal and stable.

8. PID Control

The PID controller is unquestionably the most common controller, therefore it is appropriate to give it special attention. The "textbook" version of the PID-controller can be described by the equation

$$u(t) = K\left(e(t) + \frac{1}{T_i}\int^t e(s)\,ds + T_d\,\frac{de(t)}{dt}\right) \tag{39}$$

where the error e is the difference between the command signal u_c (the set point) and the process output y (the measured variable). K is the *gain* or *proportional gain*, T_i the *integration time* or *reset time*, and T_d the *derivative time* of the controller. The PID-controller was originally implemented using analog technology that went through several development stages, that is, pneumatic valves, relays and motors, transistors, and integrated circuits. In this development much know-how was accumulated that was embedded into the analog design. Today virtually all PID-controllers are implemented digitally. Early digital implementations were often a pure translation of (39), which left out many of the extra features that were incorporated in the analog design. In this chapter we will discuss the digital PID-controller in some detail. This is a good illustration of many practical issues in the implementation of control systems.

Modification of Linear Response

A pure derivative cannot, and should not be, implemented, because it will give a very large amplification of measurement noise. The gain of the derivative must thus be limited. This can be done by approximating the transfer function sT_d as follows:

$$sT_d \approx \frac{sT_d}{1 + sT_d/N}$$

The transfer function on the right approximates the derivative well at low frequencies but the gain is limited to N at high frequencies. N is typically in the range of 3 to 20.

In the work with analog controllers it was also found advantageous not to let the derivative act on the command signal. Later it was also found suitable to let only a fraction b of the command signal act on the proportional part. The PID-algorithm then becomes

$$U(s) = K\left(bU_c(s) - Y(s) + \frac{1}{sT_i}\left(U_c(s) - Y(s)\right) - \frac{sT_d}{1 + sT_d/N}\,Y(s)\right) \tag{40}$$

where U, U_c, and Y denote the Laplace transforms of u, u_c, and y. The idea of providing different signal paths for the process output and the command signal is a good way to separate command signal response (the servo case) from the response to disturbances (the regulator case). Alternatively, it may be viewed as a way to position the closed-loop zeros. There are also several other variations of the PID-algorithm that are used in commercial systems. An extra first-order lag may be used in series with the controller to obtain a high-frequency roll-off. In some applications it has also been useful to include nonlinearities. The proportional term Ke can be replaced by $Ke|e|$ and a dead zone can also be included.

Discretization

The controller described by (40) can be discretized using any of the standard methods such as Tustin's approximation or ramp equivalence. Because the PID-controller is so simple, there are some special methods that commonly are used. The following is a popular approximation that is very easy to derive. The proportional part

$$P(t) = K\Big(bu_c(t) - y(t)\Big)$$

requires no approximation because it is a purely static part. The integral term

$$I(t) = \frac{K}{T_i}\int^t e(s)\,ds$$

is approximated by a forward approximation, that is,

$$I(kh+h) = I(kh) + \frac{Kh}{T_i}\,e(kh)$$

The derivative part given by

$$\frac{T_d}{N}\,\frac{dD}{dt} + D = -KT_d\,\frac{dy}{dt}$$

is approximated by taking backward differences. This gives

$$D(kh) = \frac{T_d}{T_d + Nh}\,D(kh-h) - \frac{KT_dN}{T_d+Nh}\,\Big(y(kh) - y(kh-h)\Big)$$

This approximation has the advantage that it is always stable and that the sampled pole goes to zero when T_d goes to zero. Tustin's approximation gives an approximation such that the pole instead goes to $z = -1$ as T_d goes to zero. The control signal is given as

$$u(kh) = P(kh) + I(kh) + D(kh) \tag{41}$$

This approximation has the pedagogical advantage that the proportional, integral, and derivative terms are obtained separately. The other approximations give similar results. They can all be represented as

$$R(q)u(kh) = T(q)u_c(kh) - S(q)y(kh) \tag{42}$$

where the polynomials R, S, and T are of second order. The polynomial R has the form

$$R(q) = (q-1)(q-a_d) \tag{43}$$

The number a_d and the coefficients of the polynomials S and T obtained for different approximation methods are given in Table 3.

Incremental Algorithms

Equation (41) is called a *position algorithm* or an *absolute algorithm*. The reason is that the output of the controller is the absolute value of the control signal, for instance, a valve position. In some cases it is advantageous to move the integral action outside the control algorithm. This is natural when a stepper motor is used. The output of the controller should then represent the increments of the

Table 3 Coefficients in different approximations of the continuous-time PID-controller.

	Special	Tustin	Ramp equivalence
s_0	$K(1+b_d)$	$K(1+b_i+b_d)$	
s_1	$-K(1+a_d+2b_d-b_i)$	$-K\Big(1+a_d+2b_d-b_i(1-a_d)\Big)$	
s_2	$K(a_d+b_d-b_ia_d)$	$K(a_d+b_d-b_ia_d)$	
t_0	Kb	$K(b+b_i)$	
t_1	$-K(b(1+a_d)-b_i)$	$-K\Big(b(1+a_d)-b_i(1-a_d)\Big)$	
t_2	$Ka_d(b-b_i)$	$Ka_d(b-b_i)$	
a_d	$\dfrac{T_d}{Nh+T_d}$	$\dfrac{2T_d-Nh}{2T_d+Nh}$	$\exp\left(-\dfrac{Nh}{T_d}\right)$
b_d	Na_d	$\dfrac{2NT_d}{2T_d+Nh}$	$\dfrac{T_d}{h}(1-a_d)$
b_i	$\dfrac{h}{T_i}$	$\dfrac{h}{2T_i}$	$\dfrac{h}{2T_i}$

control signal, and the motor implements the integrator. Another case is when an actuator with pulse-width control is used.

To obtain an incremental algorithm the control algorithm is rewritten so that its output is the increment of the control signal. Because it follows from (43) that the polynomial R in (42) always has a factor $(q-1)$ this is easy to do. Introducing

$$\Delta u(kh) = u(kh) - u(kh-h)$$

we get

$$(q-a_d)\Delta u(kh+h) = T(q)u_c(kh) - S(q)y(kh)$$

This is called the *incremental* form of the controller. A drawback with the incremental algorithm is that it cannot be used for P- or PD-controllers. If this is attempted the controller will be unable to keep the reference value, because an unstable mode $z-1$ is canceled.

Integrator Windup

A controller with integral action combined with an actuator that becomes saturated can give some undesirable effects. If the control error is so large that the integrator saturates the actuator, the feedback path will be broken, because the actuator will remain saturated even if the process output changes. The integrator, being an unstable system, may then integrate up to a very large value. When the error is finally reduced, the integral may be so large that it takes considerable time until the integral assumes a normal value again. This effect is called *integrator windup*. The effect is illustrated in Figure 22.

There are several ways to avoid integrator windup. One possibility is to stop updating the integral when the actuator is saturated. Another method for *anti-windup* is illustrated by the block diagram in Figure 23. In this system an extra feedback path is provided by using the output of the actuator model and forming

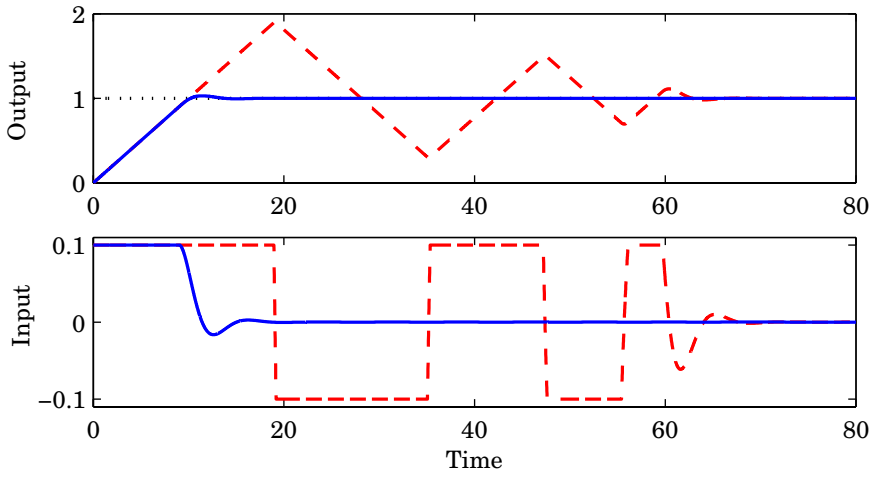

Figure 22 Illustration of integrator windup. The dashed/red lines show the response with an ordinary PID-controller with limitation of the control signal. The solid/blue lines show the improvement with a controller having antiwindup.

an error signal e_s as the difference between the estimated actuator output u and the controller output v and feeding this error back to the integrator through the gain $1/T_t$. The error signal e_s is zero when the actuator is not saturated. When the actuator is saturated the extra feedback path tries to make the error signal e_s equal zero. This means that the integrator is reset, so that the controller output is at the saturation limit. The integrator is thus reset to an appropriate value with the time constant T_t, which is called the tracking-time constant. Figure 22 shows the improved behavior with controllers having an antiwindup scheme.

Operational Aspects

Practically all PID-controllers can run in two modes: manual and automatic. In manual mode the controller output is manipulated directly by the operator, typically by push buttons that increase or decrease the controller output. The controllers may also operate in combination with other controllers, as in a cascade or ratio connection, or with nonlinear elements such as multipliers and selectors. This gives rise to more operational modes. The controllers also have

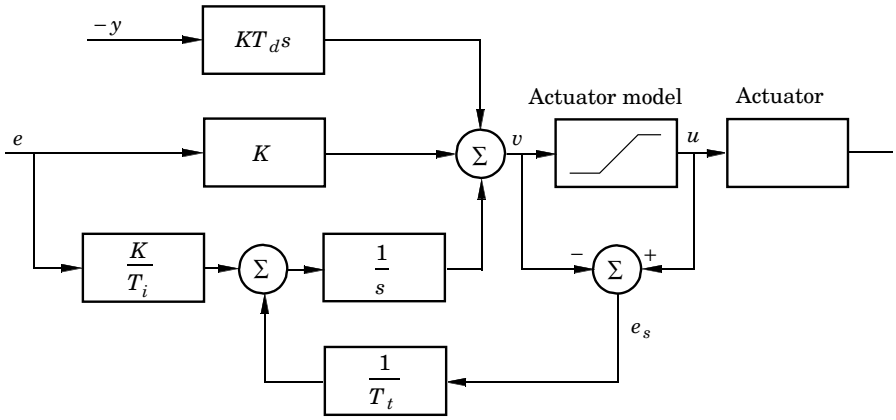

Figure 23 PID controller with antiwindup. The actuator output is estimated from a mathematical model of the actuator.

parameters that can be adjusted during operation. When there are changes of modes and parameters, it is essential to avoid switching transients.

Computer Code

A typical computer implementation of a discrete PID-controller is given in Listing 1. The discretization of the integral term is made using a forward difference. The derivative term is approximated using a backward difference. The programming language used in this example is an extension of Java.

The PID controller is implemented as a class and the signals, states, and parameters of the controller are represented as inner classes of the PID class. However, the code would look very similar also in other programming languages. In the code we assume the existence of a `waitUntil()` primitive that makes it possible to wait until an absolute time point. A primitive of this sort is normally available in most real-time kernels and languages. However, it is not a part of standard Java. A further discussion about how to implement periodic controller tasks will be found in Chapter 12.

The constructor `PID()` is called only once when an instance of the PID class is created. In a real system parts of these calculations have to be made each time the controller parameters are changed. The code given admits bump-less parameter changes if $b = 1$. When $b \neq 1$ the proportional term (`P`) is different from zero in steady state. To ensure bump-less parameter changes it is necessary that the quantity `P + I` is invariant to parameter changes. This means that the state `I` has to be changed as follows:

$$I_{\text{new}} = I_{\text{old}} + K_{\text{old}}(b_{\text{old}}u_c - y) - K_{\text{new}}(b_{\text{new}}u_c - y)$$

Tuning

If the sampling interval is chosen sufficiently short and if proper antialiasing filters are used the digital PID controller behaves essentially as an analog controller and the tuning rules for analog controllers can be used.

Summary

- The digital PID algorithm is a useful controller.
- Important to consider
  - Filtering of the derivative or high frequency roll off.
  - Setpoint weighting (poor man's two-degrees-of-freedom controller).
  - Antiwindup

Listing 1 Java code for PID-controller.

```
public class PID {

  private class Signals {
    double uc;    // Input: Set Point
    double y;     // Input: Measured Variable
    double v;     // Output: Controller output
    double u;     // Output: Limited controller output
  }

  private class States {
    double I = 0;     // Integral part
    double D = 0;     // Derivative part
    double yold = 0;  // Delayed measured variable
  }

  private class Parameters {
    double K;     // Proportional gain
    double Ti;    // Integral time
    double Td;    // Derivative time
    double Tt;    // Reset time
    double N;     // Maximum derivative gain
    double b;     // Fraction of set point in prop. term
    double ulow;  // Low output limit
    double uhigh; // High output limit
    double h;     // Sampling period
    double bi, ar, bd, ad;
  }

  private Signals signals = new Signals();
  private States states = new States();
  private Parameters par = new Parameters();

  public PID() {  // Constructor
    par.K = 4.4;
    par.Ti = 0.4;
    par.Td = 0.2;
    par.Tt = 10;
    par.N = 10;
    par.b = 1;
    par.ulow = -1;
    par.uhigh = 1;
    par.h = 0.03;
    par.bi = par.K*par.h/par.Ti;
    par.ar = par.h/par.Tt;
    par.ad = par.Td/(par.Td + par.N*par.h);
    par.bd = par.K*par.N*par.ad;
  }
```

```
  public double calculateOutput(double uc, double y) {
    signals.uc = uc;
    signals.y = y;
    double P = par.K*(par.b*uc - y);
    states.D = par.ad*states.D - par.bd*(y - states.yold);
    signals.v = P + states.I + states.D;
    if (signals.v < par.ulow) {
      signals.u = par.ulow;
    } else {
      if (signals.v > par.uhigh) {
        signals.u = par.uhigh;
      } else {
        signals.u = signals.v;
      }
    }
    return signals.u;
  }

  public void updateState(double u) {
    states.I = states.I + par.bi*(signals.uc - signals.y) +
               par.ar*(u - signals.v);
    states.yold = signals.y;
  }


  public static void main(String[] args) {
    double uc, y, u;
    PID pid = new PID();
    long time = System.getCurrentTimeMillis(); // Get current time
    while (true) {
      y = readY();
      uc = readUc();
      u = pid.calculateOutput(uc,y);
      writeU(u);
      pid.updateState(u);
      time = time + par.pid.h*1000;        // Increment time with h
      Thread.waitUntil(time);              // Wait until "time"
    }
  }

}
```

9. Pole-placement Design

In this chapter we will design two-degrees-of-freedom controllers based on pole-placement using input-output descriptions. The closed loop system will have the structure given in Figure 14. The process is described by

$$A(q)y(k) = B(q)(u(k) + v(k)) \tag{44}$$

where the degree of $B(q)$ is less than the degree of $A(q)$. Further, it is assumed that the polynomials $A(q)$ and $B(q)$ do not have any common factors and that the polynomial $A(q)$ is *monic*, i.e. it is normalized so that the coefficient of the term with the highest power in q is one. The signal $v(k)$ is an input disturbance to the process.

A general linear controller can be represented by

$$R(q)u(k) = T(q)u_c(k) - S(q)y(k) \tag{45}$$

where $R(q)$, $S(q)$, and $T(q)$ are polynomials in the forward-shift operator. The polynomial $R(q)$ is assumed to be monic. The control law (45) represents a combination of a feedforward with the pulse-transfer function $H_{ff}(z) = T(z)/R(z)$ and a feedback with the pulse-transfer function $H_{fb}(z) = S(z)/R(z)$ in Figure 14. To have a causal controller it is necessary that the degree of $R(z)$ is larger than or equal to the degrees of $S(z)$ and $T(z)$.

From (44) and (45) it follows that the closed loop system is

$$y(k) = \frac{B(q)T(q)}{A(q)R(q) + B(q)S(q)}u_c(k) + \frac{B(q)R(q)}{A(q)R(q) + B(q)S(q)}v(k) \tag{46}$$

The characteristic polynomial of the closed-loop system is

$$A_{cl}(z) = A(z)R(z) + B(z)S(z) \tag{47}$$

We now want to design the controller such that the closed loop system fulfills the following conditions:

1. The pulse-transfer function from u_c to y should be

   $$H_m = \frac{B_m}{A_m}$$

   with A_m stable and with $\deg A_m = \deg A$. Further, we want to have unit steady-state gain from u_c to y, which implies that $B_m(1) = A_m(1)$.

2. From (47) it follows that $\deg(AR + BS) \geq \deg A_m$. This implies that some dynamics in the pulse-transfer function from u_c to y has to be canceled. This dynamics must be stable and is denoted A_o.

3. The only way to remove open loop zeros in (46) is by cancellation. This implies that only stable zeros $B^+(q)$ can be removed. Introduce the factorization

   $$B(q) = B^+(q)B^-(q)$$

   To get a unique factorization it is assumed that $B^+(q)$ is monic. This implies that we require that the unstable zeros are kept in H_m, i.e.

   $$B_m(q) = B^-(q)\bar{B}_m(q)$$

   Compare the discussion of feedforward design in Chapter 7.

The conditions above are fulfilled if we choose $A_{cl} = A_o A_m B^+$ and the controller polynomials as

$$\begin{aligned} R &= B^+ \bar{R} \\ T &= \bar{B}_m A_o \end{aligned} \tag{48}$$

where $\bar{R}$ and S fulfill the *Diophantine equation*

$$A_o A_m = A\bar{R} + B^- S \tag{49}$$

The polynomial B^+ is here canceled compared with (47) This gives the closed loop system

$$\begin{aligned} y(k) &= \frac{B^+ A_o}{B^+ A_o} \cdot \frac{B^- \bar{B}_m}{A_m} u_c(k) + \frac{B^+}{B^+} \cdot \frac{B^+ B^- \bar{R}}{A_o A_m} v(k) \\ &= \frac{B_m}{A_m} u_c(k) + \frac{B\bar{R}}{A_o A_m} v(k) \end{aligned} \tag{50}$$

From (50) it follows that the factors $B^+ A_o$ are canceled in the pulse-transfer function from the reference signal to the output. The stable polynomial A_o is called the *observer polynomial*, since it can be interpreted as the dynamics of an observer that reconstructs the internal states of the process.

The dynamics of the transfer function from the disturbance v to the output is characterized by $A_o A_m$. If we want to separate the design for the servo and the regulator cases we need to have different responses from the reference signal and the disturbance. This essentially means that we want to replace A_m by A_c in the second term of (50). This can be obtained by modifying the controller to

$$\begin{aligned} R &= A_m B^+ \bar{R} \\ S &= A_m \bar{S} \\ T &= \bar{B}_m A_o A_c \end{aligned} \tag{51}$$

where $\bar{R}$ and S fulfill the Diophantine equation

$$A_o A_c = A\bar{R} + B^- \bar{S} \tag{52}$$

The closed loop system is now

$$\begin{aligned} y(k) &= \frac{BT}{AR + BS} u_c(k) + \frac{BR}{AR + BS} v(k) \\ &= \frac{B^+ A_o A_c}{B^+ (A\bar{R} + B^- \bar{S})} \cdot \frac{B^- \bar{B}_m}{A_m} u_c(k) + \frac{B^+ A_m}{B^+ A_m} \cdot \frac{B^- \bar{R}}{A\bar{R} + B^- \bar{S}} v(k) \\ &= \frac{B_m}{A_m} u_c(k) + \frac{B^- \bar{R}}{A_o A_c} v(k) \end{aligned}$$

So far we have only made formal calculations and determined the closed loop system that satisfies the specifications. We need to determine the conditions for when there exists a solution to our design problem.

The Diophantine Equation

The make the design it is necessary to solve the Diophantine equation and determine the controller polynomials. The fundamental mathematical problem is to understand the properties of the polynomial equation (52). It is named after Diophantus ($\approx$ A.D. 300), who was one of the original inventors of algebra. It has also many other names in literature, the *Aryabhatta's identity* or the *Bezout identity*. The existence of a solution will depend on the degrees of the polynomials and if there are common factors between the polynomials. The general Diophantine equation

$$AX + BY = C \tag{53}$$

has a solution if and only if the greatest common factor of A and B divides C. If one solution X_0, Y_0 exists there are $X = X_0 + QB$, and $Y = Y_0 - QA$, where Q is an arbitrary polynomial and is also a solution. There are thus many solutions to the equation, which can be generated as soon as one solution is obtained. There are, however, unique solutions to (53) such that $\deg X < \deg B$ or $\deg Y < \deg A$.

Solving the Diophantine equation for the coefficients in the X and Y polynomials is the same as solving a set of linear equations.

Causality Conditions

It follows from the analysis that there may be infinitely many solutions to the Diophantine equation. For the Diophantine equations that appear when solving the pole-placement problem, it is natural to introduce some constraints on the solution. The degrees of the polynomials $S(z)$ and $T(z)$ must be less than or equal to the degree of $R(z)$. If this is not the case the control signal at time k will depend on values of the measured signal and the command signal at times larger than k. We call this the *causality condition*, i.e.

$$\deg R \geq \deg T \quad \text{and} \quad \deg R \geq \deg S$$

Further, the closed loop system must have the same or longer time delay than the open loop system, which implies that

$$\deg A_m - \deg B_m \geq \deg A - \deg B = d$$

where d is the time delay of the process.

Summary of the Pole-placement Design Procedure

In the control problem it is natural to select the solution of (47) that gives a causal controller of lowest order. Since the process model must be causal it follows that $\deg B \leq \deg A$. Because the controller is also causal we have $\deg S \leq \deg R$. We will thus want to find the solution where the degree of S is as low as possible. From the condition for the unique solution of the Diophantine equation we have $\deg S < \deg A$. It follows that if $\deg A = n$ the minimum-degree solution corresponds to $\deg S = \deg R = \deg T = \deg A_o = n - 1$ and $\deg A_{cl} \geq 2n - 1$.

The minimum degree solution of the pole-placement design with $A_m = A_c$ is summarized by the conditions

$$\begin{aligned} \deg S &= \deg R = \deg T = n - 1 \\ \deg A_o &= \deg A - \deg B^+ - 1 \\ \deg A_m &= \deg A = n \end{aligned}$$

and the conditions (48) where $\bar{R}$ and S fulfill (49).

EXAMPLE 13—CONTROL OF A MOTOR
Consider the transfer function

$$G(s) = \frac{K}{s(s+1)}$$

which can be a model for a DC motor. Sampling the system gives the pulse-transfer function

$$H(z) = \frac{K(z-b)}{(z-1)(z-a)}$$

where

$$K = e^{-h} - 1 + h, \quad a = e^{-h}, \quad \text{and} \quad b = 1 - \frac{h(1-e^{-h})}{e^{-h} - 1 + h}$$

Notice that $-1 < b < 0$; that is, the zero is on the negative real axis, but inside the unit circle. It is first assumed that the desired closed-loop system is characterized by the pulse-transfer function

$$H_m(z) = \frac{z(1+p_1+p_2)}{z^2 + p_1 z + p_2}$$

It is assumed that the denominator corresponds to the sampling of the second order polynomial $s^2 + 2\zeta\omega s + \omega^2$ with $\omega = 1$ and $\zeta = 0.7$. The pulse-transfer function H has a zero $z = b$ that is not included in H_m. With the given specifications, it is necessary to cancel the zero $z = b$. The observer polynomial is chosen as

$$A_o(z) = 1$$

The following polynomial identity is then obtained.

$$(z-1)(z-a)r_0 + K(s_0 z + s_1) = z^2 + p_1 z + p_2$$

Equating coefficients of equal powers of z gives the controller parameters

$$r_0 = 1, \quad s_0 = \frac{1+a+p_1}{K}, \quad \text{and} \quad s_1 = \frac{p_2 - a}{K}$$

Further

$$T(z) = A_o(z)\bar{B}_m(z) = \frac{z(1+p_1+p_2)}{K} = t_0 z$$

The control law can be written as

$$u(k) = t_0 u_c(k) - s_0 y(k) - s_1 y(k-1) + b u(k-1) \tag{54}$$

A simulation of the step response of the system is shown in Figure 24(a). Notice the "ringing," or the "ripple," in the control signal, which is caused by the cancellation of the zero on the negative real axis. The ripple is not noticeable in the output signal at the sampling instants. It may, however, be seen as a ripple in the output between the sampling instants when the sampling interval is sufficiently long. The amplitude of the ripple in the output decreases rapidly as the sampling period is decreased.

Now assume that the desired closed-loop transfer function is

$$H_m(z) = \frac{1+p_1+p_2}{1-b} \; \frac{z-b}{z^2 + p_1 z + p_2}$$

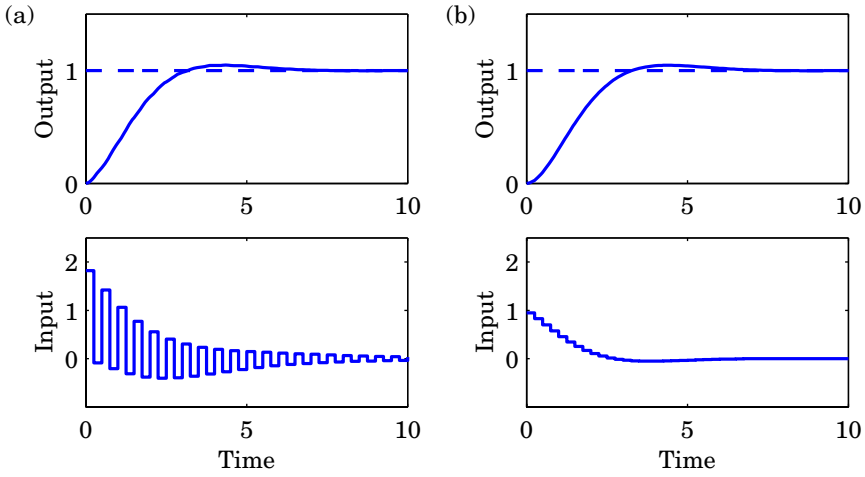

Figure 24 Step response of a motor with pole-placement control. The specifications are $\omega = 1$ and $\zeta = 0.7$. The sampling period $h = 0.25$. (a) The process zero is canceled; (b) The process zero is not canceled.

The process zero on the negative real axis is now also a zero of the desired closed-loop transfer function. This means that the zero does not have to be canceled by the regulator. The degree of the observer polynomial is

$$\deg A_o \geq 2 \deg A - \deg A_m - \deg B^+ - 1 = 1$$

Therefore, the observer polynomial should be of at least first order. A deadbeat observer is chosen with

$$A_o(z) = z$$

The minimal degrees of the polynomials R and S are then given by

$$\begin{aligned} \deg R &= \deg A_m + \deg A_o - \deg A = 1 \\ \deg S &= \deg A - 1 = 1 \end{aligned}$$

The Diophantine equation can then be written as

$$(z-1)(z-a)(z+r_1) + K(z-b)(s_0 z + s_1) = z^3 + p_1 z^2 + p_2 z$$

Further

$$T(z) = A_o \bar{B}_m = z\,\frac{1 + p_1 + p_2}{K(1-b)} = t_0 z$$

Notice that the controller is of the same form as (54). However, the coefficients are different. A simulation of the step response of the system is shown in Figure 24(b). A comparison with Figure 24(a) shows that the control signal is much smoother; there is no ringing. The response start is also a little slower, because A_o is of higher degree than when the process zero was canceled. □

Introduction of Integrators

To be able to eliminate constant input disturbances it is necessary that the controller contains an integrator. It is easy to introduce integrators in the controller

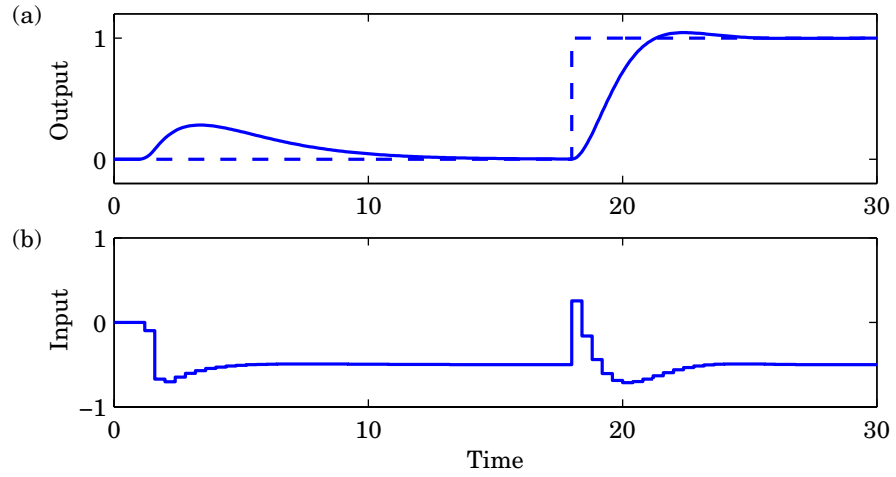

Figure 25 Control of the double integrator using the controller a pole placement controller with different responses for disturbances and reference signals. (a) Output (solid) and reference signal (dashed), (b) control signal.

(45) by postulating that the polynomial R contains the factor $z-1$. The Diophantine equation (49) then becomes

$$A_o A_m = A(z-1)\bar{R} + B^- S$$

Further the orders of the observer and S polynomials have to be increased by one to get a solution.

EXAMPLE 14—CONTROL OF THE DOUBLE INTEGRATOR

Consider the double-integrator plant and assume that there is a process disturbance in the form of an unknown constant that is acting on the process input. The pole placement controller is designed such that the controller has an integrator and that the A_c is chosen to correspond to a sampled second order system with with the $\omega_c = 0.5$, the damping $\zeta_c = 1$, and $h = 0.4$. The A_m is chosen with $\omega_m = 1$ and $\zeta_m = 0.7$. Figure 25 shows the control of the double integrator when using the pole-placement controller designed using (51) and (52) and with an integrator in the controller. There is first an input load disturbance at time $t = 1$, and then a change in the reference value to the model at $t = 18$. The model is designed to be twice as fast as when the disturbance rejection is designed. The simulation shows that the regulation and servo problems can be separated and given different dynamics. □

Summary

- Pole-placement can be obtained using a two-degree-of-freedom input-output controller.
- The design requires the solution of a set of linear equations when solving the Diophantine equation.
- The design requires that the user can specify the desired closed loop transfer functions.
- Zeros outside the unit circle or to the left of the imaginary axis should not be canceled.

- Integrators are easily introduced.
- The servo and regulator problems can be separated.
- The effects of aliasing can be reduced by imposing the condition $S(-1) = 0$, this implies that the controller gain is zero at the Nyquist frequency.

10. Optimization Based Design

In the previous chapters we have discusses the PID and pole-placement controllers. The design of these controllers have either been based on on-line (heuristic) tuning or specification of the closed loop pulse-transfer function. In these designs we have to compromise between the speed of the response and the size of the control signal. In this chapter we will discuss two approaches to the design based on optimization criteria.

Linear Quadratic (LQ) Design

The process is assumed to be linear, but it may be time-varying and have several inputs and outputs. The design problem is formulated as to minimize a criterion, which is a quadratic function of the states and the control signals. The resulting optimal controller is linear. The system is described by

$$x(k+1) = \Phi x(k) + \Gamma u(k) \tag{55}$$

where $x(0)$ is given. We may also allow the matrices Φ and Γ to be time-varying. The problem is now to determine the control sequence $u(0)$, $u(1)$, ..., $u(N-1)$ that minimizes the loss function

$$\begin{aligned} J = \sum_{k=0}^{N-1} \Big(& x^T(kh) Q_1 x(kh) + 2x^T(kh) Q_{12} u(kh) \\ & + u^T(kh) Q_2 u(kh) \Big) + x^T(Nh) Q_0 x(Nh) \end{aligned} \tag{56}$$

The loss function can also be defined in continuous time and then sampled to get a loss function of the form (56).

Using the idea of *Dynamic Programming* introduced by Richard Bellman the solution is obtained by minimizing the loss function backwards. The minimum control sequence is given by

$$u(k) = -L(k)x(k) \tag{57}$$

where

$$L(k) = \Big(Q_2 + \Gamma^T S(k+1)\Gamma \Big)^{-1} \Big(\Gamma^T S(k+1)\Phi + Q_{12}^T \Big) \tag{58}$$

and where $S(k)$ is the solution of the Riccati equation

$$\begin{aligned} S(k) = \Phi^T S(k+1)\Phi + Q_1 - & \Big(\Phi^T S(k+1)\Gamma + Q_{12} \Big) \\ & \times \Big(\Gamma^T S(k+1)\Gamma + Q_2 \Big)^{-1} \Big(\Gamma^T S(k+1)\Phi + Q_{12}^T \Big) \end{aligned} \tag{59}$$

with the end condition $S(N) = Q_0$. The optimal linear quadratic controller is a linear feedback from the states of the system. Compare (24). The condition for the existence of a unique control strategy is that Q_0 is positive semidefinite and that $Q_2 + \Gamma^T S(k)\Gamma$ is positive definite. The solution of the discrete-time Riccati equation can only be made by hand for very simple problems. Efficient numerical algorithms are, however, available for the solution of the Riccati equation.

Notice, that the system (55) is allowed to be multivariable and time-varying. The resulting controller (57) is time-varying, even if the system is time-invariant.

In many situations it is sufficient to use the stationary solution. The stationary solution can be obtained by iterating (59) or by solving the algebraic Riccati equation

$$S = \Phi^T S\Phi + Q_1 - \left(\Phi^T \bar{S}\Gamma + Q_{12}\right)\left(\Gamma^T S\Gamma + Q_2\right)^{-1}\left(\Gamma^T S\Phi + Q_{12}^T\right)$$

The LQ-controller has several good properties. It is applicable to multivariable and time-varying systems. Also, changing the relative magnitude between the elements in the weighting matrices means a compromise between the speed of the recovery and the magnitudes of the control signals. Using the stationary solution for S in (57) and (58) guarantees a stable closed loop system if the the weighting matrices fulfill the positivity conditions stated above and if there exists a positive definite steady-state solution to the algebraic Riccati equation.

How to Find the Weighting Matrices?

When using optimization theory, the weighting matrices in the loss function should ideally come from physical arguments. There are, however, few situations where this is the case. Linear quadratic control theory has found considerable use even when this cannot be done. In such cases the control designer chooses a loss function. The feedback law is obtained directly by solving the Riccati equation. The closed-loop system obtained is then analyzed with respect to transient response, frequency response, robustness, and so on. The elements of the loss function are modified until the desired result is obtained. It has been found empirically that LQ-theory is quite easy to use in this way. The search will automatically guarantee stable closed-loop systems with reasonable margins.

It is often fairly easy to see how the weighting matrices should be chosen to influence the properties of the closed-loop system. Variables x_i, which correspond to significant physical variables, are chosen first. The loss function is then chosen as a weighted sum of x_i. Large weights correspond to small responses. The responses of the closed-loop system to typical disturbances are then evaluated. The relative weights between state variables and control variables are usually found by trial and error.

Sometimes the specifications are given in terms of the maximum allowed deviations in the states and the control signals for a given disturbance. One rule of thumb to decide the weights in (56) is to choose the diagonal elements as the inverse value of the square of the allowed deviations. Another way is to consider only penalties on the state variables and constraints on the control deviations.

EXAMPLE 15—LQ-CONTROL OF THE DOUBLE INTEGRATOR

Consider the double integrator and use the sampling period $h = 1$. Let the weighting matrices in (56) be

$$Q_1 = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \quad \text{and} \quad Q_2 = \begin{pmatrix} \rho \end{pmatrix}$$

The influence of the weighting can now be investigated. The stationary feedback vector has been calculated for different values of ρ. Figure 26 shows the output and the control signal for some values. When $\rho = 0$, which means there is a penalty only on the output, then the resulting controller is the same as the deadbeat controller. When ρ is increased, then the magnitude of the control signal is decreased. The parameter ρ is now, together with the sampling period, the design parameter.

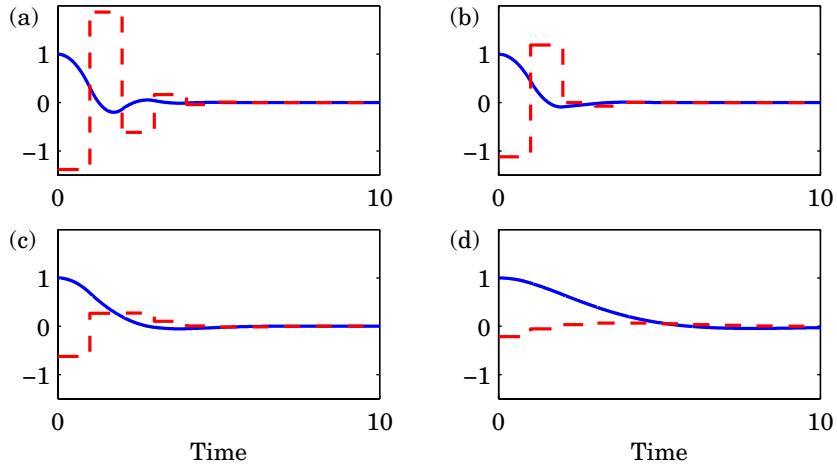

Figure 26 Linear quadratic control of the double-integrator plant for different weightings, ρ, on the control signal. The initial value of the state is $x(0) = [1 \quad 0]$. The output y (solid/blue) and the control signal u (dashed/red) are shown. (a) $\rho = 0.015$, (b) $\rho = 0.05$, (c) $\rho = 0.5$, and (d) $\rho = 10$.

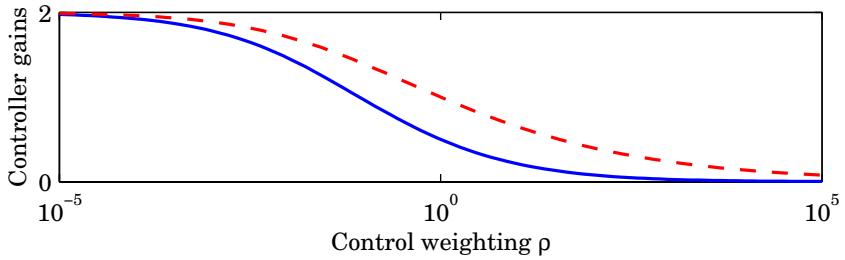

Figure 27 Linear quadratic controller for the double integrator. The stationary gains l_1 (solid/blue) and l_2 (dashed/red) of the feedback vector $L = [l_1, \ l_2]$ for different values of the control weighting ρ.

Figure 27 shows the stationary L vector as a function of the control weighting ρ. When ρ increases the gains go to zero and there will be almost no feedback. □

Kalman Filters and LQG Control

When using the linear quadratic controller the full state vector has to be measured. From the measurements of only the outputs it is, however, possible to obtain estimates of the states using the *Kalman filter*. Let the process be described by

$$\begin{aligned} x(k+1) &= \Phi x(k) + \Gamma u(k) + v(k) \\ y(k) &= Cx(k) + e(k) \end{aligned} \tag{60}$$

where v and e are discrete-time Gaussian white-noise processes with zero-mean value and

$$\begin{aligned} \mathrm{E}\, v(kh)v^T(kh) &= R_1 \\ \mathrm{E}\, v(kh)e^T(kh) &= R_{12} \\ \mathrm{E}\, e(kh)e^T(kh) &= R_2 \end{aligned}$$

From measurements of the inputs and outputs the Kalman filter gives the one-

step-ahead estimates of the states, compare (37),

$$\hat{x}(k+1) = \Phi\hat{x}(k) + \Gamma u(k) + K(k)\left(y(k) - C\hat{x}(k)\right)$$

where

$$\begin{aligned} K(k) &= \left(\Phi P(k) C^T + R_{12}\right)\left(R_2 + CP(k)C^T\right)^{-1} \\ P(k+1) &= \Phi P(k)\Phi^T + R_1 \\ &\quad - \left(\Phi P(k) C^T + R_{12}\right)\left(R_2 + CP(k)C^T\right)^{-1}\left(CP(k)\Phi^T + R_{12}^T\right) \\ P(0) &= R_0 \end{aligned}$$

Consider the loss function $\mathrm{E}J$ where J is given by (56). The optimal controller that minimizes this loss function is given by

$$u(k) = -L(k)\hat{x}(k \mid k-1)$$

where $L(k)$ still is given by (58). The resulting controller consists of a linear feedback from the estimated states. This implies that the computation of L is separated from the computation of K. The Linear Quadratic Gaussian (LQG) controller is very flexible since it can be used for multivariable and time-varying systems. One criticism of LQG-control is that an accurate full-order model of the process must be available. Most physical processes are of high order. However, for control purposes it is often sufficient to use a low-order approximation.

$\mathcal{H}_\infty$ Design

Classical control design methods are based on shaping of Bode or Nyquist curves using graphical design methods. Many methods have been proposed and much research efforts have been given to find systematic and computational feasible methods for loop shaping. The developed methods are quite mathematical, but good computational tools have been developed, which have made the methods feasible to use. The basic idea is to find a controller for a multivariable system such that the sensitivity and complementary sensitivity functions together with the transfer function from the measurement noise, or equivalently from the reference signal, to the control signal are as small as possible in some sense. For simplicity, we will discuss the single input single output case. The relevant transfer functions are, see (25),

$$\begin{aligned} T &= -H_{ze} = \frac{H_p H_c}{1 + H_p H_c} \\ S &= H_{ye} = \frac{1}{1 + H_p H_c} \\ H_{ue} &= -\frac{H_c}{1 + H_p H_c} \end{aligned} \tag{61}$$

These pulse-transfer functions should be “small”. The system can be represented as in Figure 28, where the the controller to be designed also is included. The system has the input e and the outputs

$$\zeta = \begin{pmatrix} W_u H_{ue} \\ -W_T T \\ W_S S \end{pmatrix} e = H_{\zeta e} e$$

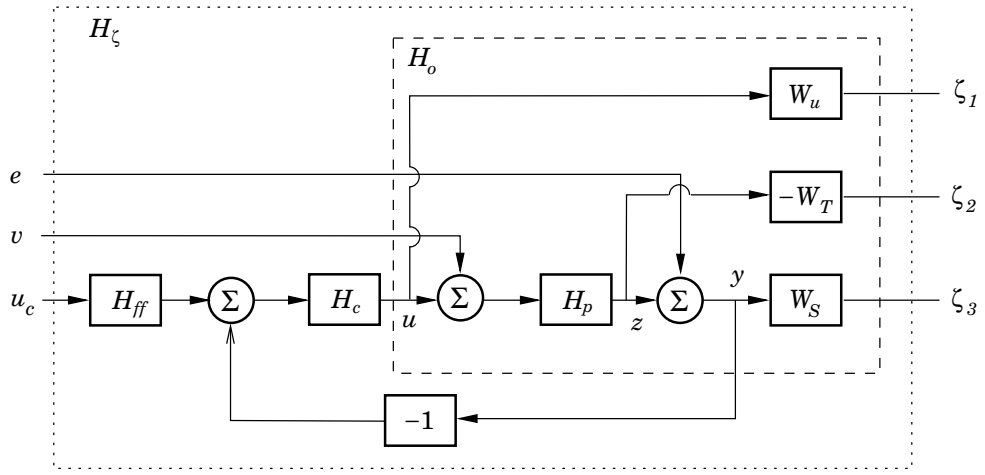

Figure 28 Block diagram of expanded system for loop shaping.

where W_u, W_T, and W_S, are weighting functions that are used to determine the frequency ranges where the transfer functions (61) should be small. There are several ways in which the size of the transfer functions can be measured. One way is to use the $\mathcal{H}_\infty$-norm, i.e. to find, compare (26),

$$\|H_{\zeta e}\|_\infty = \max_\omega |H_{\zeta e}(e^{i\omega})|$$

In the multivariable case the right hand side is replaced by the maximum singular value of $H_{\zeta e}(e^{i\omega})$. The design problem is now defined as to find controllers that satisfy

$$\|H_{\zeta e}\|_\infty < \gamma \tag{62}$$

If this is satisfied it can be shown that

$$\begin{aligned} |W_S(e^{i\omega})S(e^{i\omega})| &< \gamma \quad \forall\omega \\ |W_T(e^{i\omega})T(e^{i\omega})| &< \gamma \quad \forall\omega \\ |W_u(e^{i\omega})H_{ue}(e^{i\omega})| &< \gamma \quad \forall\omega \end{aligned}$$

Notice that these equations are more restrictive than (62).

The solution to the $\mathcal{H}_\infty$ problem is obtained by starting with the open loop system H_o marked in Figure 28 and using the algorithm

1. Choose the transfer functions W_u, W_T, and W_S.
2. Assume a value of γ.
3. Solve a Riccati equation defined by a state space representation of H_o and γ. If no solution exists go to Step 2 and increase γ and if a solution exists use this or go to Step 2 and decrease γ to investigate if a better solution exists.
4. Investigate the properties of the closed loop system and go back to Step 1 if the specifications are not fulfilled.
5. Investigate if the order of the controller can be reduced without influencing the properties of the closed loop system.

There are heuristic rule for choosing the frequency weightings W_u, W_T, and W_S. For instance, if we want the system to be able to eliminate ramp disturbances, i.e. to have a frequency roll-off of -2, then W_S should contain two integrators. The orders of the filters will increase the order of the Riccati equation that has to be solved and it is thus advantageous to have as simple weighting filters as possible. The Riccati equation in the solution is build up from a state space representation of the open loop system H_o with its weighting filters and γ is included as a parameter.

The resulting controller can be interpreted as an observer and a linear feedback from the estimated states. I.e. the controller has the same structure as the LQG-controller discussed earlier in this chapter. The order of the open loop system and the orders of the frequency filters determines the order of the controller. This is regarded as a drawback of the design method. However, the resulting controller can often be reduced without too much deterioration in the performance of the closed loop system, see Step 5 in the algorithm.

The $\mathcal{H}_\infty$ design method is developed for continuous-time as well as for discrete-time systems. The theory and and choice of the weighting filters are easier to understand for the continuous-time case. A practical way of determining a discrete-time $\mathcal{H}_\infty$ controller is to make the design in continuous time and discretize the resulting controller using the methods discussed in Chapter 6.

Summary

- Controllers can be designed based on optimization methods.
- In the Linear Quadratic Gaussian controller there is a separation between the estimation of the states of the process and the computation of the state feedback gains.
- The design parameters of the Linear Quadratic Gaussian controller are the sampling period and the weights in the loss function.
- The $\mathcal{H}_\infty$ controller determines a controller that keeps the maximum gain of the transfer function below a certain value for all frequencies.
- The computations to derive the $\mathcal{H}_\infty$ controller have to be iterated to find the frequency filters and the minimum value of the norm. There are, however, good numerical tools for the computations.
- The order of the $\mathcal{H}_\infty$ controller normally has to be reduced.

11. Practical Issues

In the previous chapters we have discussed how to design different types of sampled-data controllers. Independent of the design method the resulting controller is a discrete-time system that has to be interfaced with the physical world outside the computer. In this chapter we will discuss some of the implementation issues. We need to discuss the realization or representation of the controller and the influence of finite precision in the A-D and D-A conversions and the calculations. Further, problems with actuator limitation will be discussed.

Controller Implementation and Computational Delay

The design of a controller for the sampled-data system results in a controller that can be represented in state-space form

$$\begin{aligned} x_c(k+1) &= Fx_c(k) + Gy(k) + G_c u_c(k) \\ u(k) &= Cx_c(k) + Dy(k) + D_c u_c(k) \end{aligned} \tag{63}$$

where $x_c(k)$ is the state of the controller. The controller (63) can be interpreted as an observer combined with a feedback from the observed states of the process. The controller can, however, also be a realization of any dynamical controller that has been designed. If the controller is designed in input-output form then the it can be written as

$$R(q)u(k) = T(q)u_c(k) - S(q)y(k) \tag{64}$$

In this form the states of the controller are the old inputs, outputs, and reference signals that have to be stored to compute the control signal. This implies that (64) normally is not a minimal representation of the controller. The number of states is in a direct implementation $\deg R + \deg S + \deg T$, while the minimal representation will have $\deg R$ states. One reason to have a non-minimal representation can be that the inputs, outputs, and reference signals have to be stored for other reasons, such as data analysis. Since the complexity of the controller, in most cases, is quite low then the extra storage of some variables does not have any practical consequences.

When discussing the implementation of the controller we can without loss of generality choose the form (63). An algorithm for the controller at time k consists of the following steps:

1. A-D conversion of $y(k)$ and $u_c(k)$.
2. Computation of the control signal $u(k) = Cx_c(k) + Dy(k) + D_c u_c(k)$.
3. D-A conversion of $u(k)$.
4. Update the state $x_c(k+1) = Fx_c(k) + Gy(k) + G_c u_c(k)$.

Notice that the implementation of the control algorithm is done such that the control signal is sent out to the process in Step 3 before the state to be used at the next sampling interval is updated in Step 4. This is to minimize the *computational delay* in the algorithm. It is possible to further reduce the amount of computations between the A-D and D-A conversions by observing that the term $Cx_c(k)$ can be precomputed, which gives the following algorithm

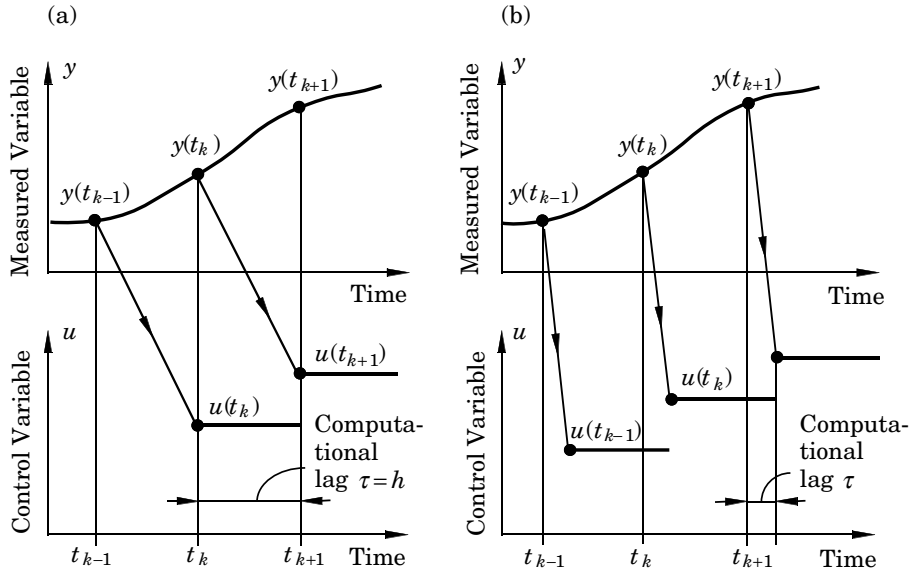

Figure 29 Two ways of synchronizing inputs and outputs. In (a) the signals measured at time t_k are used to compute the control signal to be applied at time t_{k+1}. In (b) the control signals are applied as soon as they are computed.

1. A-D conversion of $y(k)$ and $u_c(k)$.
2. Computation of the control signal $u(k) = z(k) + Dy(k) + D_c u_c(k)$.
3. D-A conversion of $u(k)$.
4. Update the state $x_c(k+1) = Fx_c(k) + Gy(k) + G_c u_c(k)$ and precompute $z(k+1) = Cx_c(k+1)$ to be used at the next sampling instance.

Compare Listing 1 of the computer code for the PID controller in Chapter 8. For a single-input-single-output controller this results in two multiplications and two additions that have to be performed between the conversions.

Depending on the real-time operating system and the conversion times in the A-D and D-A converters the computational delay, see Figure 29, may vary from sample to sample. This will be even more pronounced if the control algorithm also includes iterations or optimization steps. One way to reduce the variation of the delay is to introduce a full sample interval delay in the controller, i.e. to allow $u(k)$ to be a function of the process output and the reference signal up to time $k-1$, see in Figure 29(a). The computational delay is now more deterministic but unnecessarily long, which normally is not good for the performance of the closed loop system. The delays and timing are also discussed in Chapter 13.

Controller Representation and Numerical Roundoff

Even if (63) and (64) are general forms for implementation of the controller there is a degree of freedom in the choice of the states of the controller. Different state representations have different numerical properties. Assume that we want to implement the digital filter or controller

$$y(k) = H(q^{-1})u(k) = \frac{b_0 + b_1 q^{-1} + \cdots + b_m q^{-m}}{1 + a_1 q^{-1} + a_2 q^{-2} + \cdots + a_n q^{-n}}\, u(k)$$

The filter $H(q^{-1})$ can be implemented in different ways

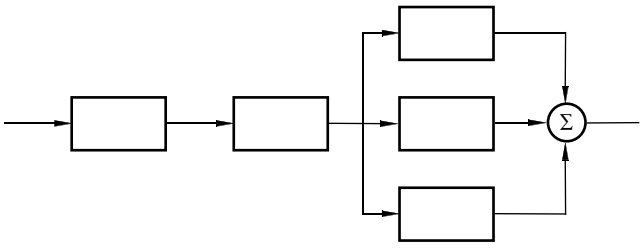

Figure 30 Illustration of series and parallel form of implementation of a digital filter where each block is a system of first or second order.

- Direct form
- Companion form
- Series or parallel form
- δ-operator form

The *direct form* is the non-minimal representation

$$y(k) = \sum_{i=0}^{m} b_i u(k-i) - \sum_{i=1}^{n} a_i y(k-i)$$

This and all other representations that directly are using the polynomial coefficients a_i and b_i are very sensitive for numerical errors if the polynomials have multiple roots close to the unit circle. Small perturbations in the coefficients or in the numerics can then lead to instabilities in the computations. Better realizations are series or parallel forms illustrated in Figure 30, where each block represents a first or second order filter.

The poles of a sampled-data system are approaching one when the sampling interval is decreased. This implies that there will be a cluster of poles that are close to the stability boundary when the sampling interval is short. To avoid numerical difficulties in this case it is possible to make a short-sampling-interval modification. From (8) it follows that the matrix F in (63) is close to a unit matrix and that the vectors G and G_c are proportional to the sampling interval for short sampling intervals. This implies that there can be several orders of magnitude in difference between F, G, and G_c. The state equation in (63) can be rewritten into the equivalent form

$$x_c(k+1) = x_c(k) + (F-I)x_c(k) + Gy(k) + G_c u_c(k) \tag{65}$$

The last three terms of the right hand side can now be interpreted as an correction of the state that are of the same magnitude. The correction is added to the previous state. The modification introduced in (65) is similar to the introduction of the δ-operator which is defined as

$$\delta f(kh) = \frac{f(kh+h) - f(kh)}{h}$$

The δ-operator can be interpreted as a forward difference approximation of the differential operator $p = d/dt$.

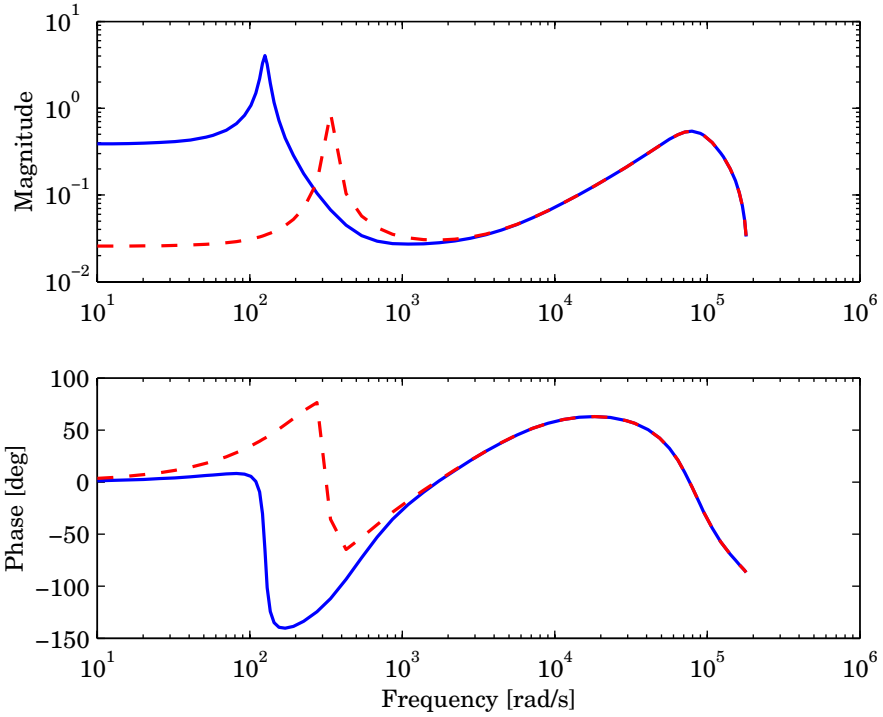

Figure 31 Bode plot of the desired controller (full/blue) for a DVD player. A 24 bits implementation of the controller coincides with the desired controller. The Bode plot for a 16 bits implementation (dashed/red) of the coefficients is also shown.

EXAMPLE 16—NUMERICAL SENSITIVITY

We will illustrate the numerical sensitivity with a simulation of the tracking in a DVD player. The purpose with the control system is to follow the track on the disk despite disturbances due to the eccentricity of the tracks. The disturbance can be regarded as a sinusoidal disturbance that the controller has to eliminate. The model for the tracking is a fourth order system with quite well damped resonance. The sampling rate is 60 kHz. Figure 31 shows the Bode plot of the desired controller in two cases. The controller is on Jordan form and coefficients are implemented with 16 and 24 bits, respectively. The resulting Bode plot of the 16 bits implementation is shown as a dashed/red line while the 24 bits implementation coincides with the desired response (full/blue line). Figure 32 shows the control error for the two implementations. It is seen that a high accuracy of the coefficients is needed to eliminate the sinusoidal disturbance. □

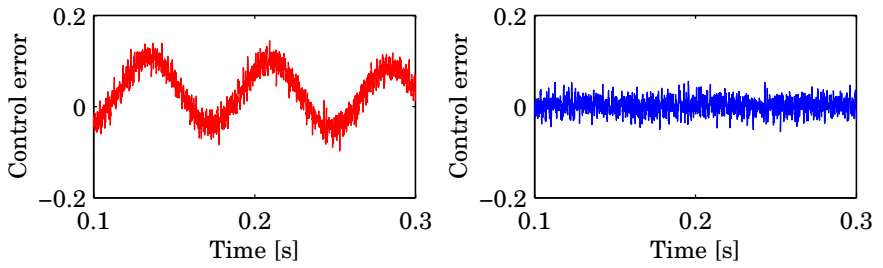

Figure 32 The control error in a simulation of the tracking in a DVD player. The controller coefficients are implemented with 16 (left) and 24 (right) bits, respectively.

A-D and D-A Quantization

The previous section illustrated how numerical quantization can influence the performance of the system. With double-precision the arithmetic computations are typically done with a resolution of 64 bits. The A-D and D-A conversions are done with much less accuracy and will have a larger influence on the performance. The resolutions of the A-D and D-A converters are typically 12–16 bits and 8–12 bits, respectively. It is better to have a good resolution of the measured signal, i.e. at the A-D converter, since a control system is less crucial for a quantized input signal. Compare the situation where on-off control is used. With such a crude input quantization it is still possible to get a reasonable closed loop performance.

Nonlinearities, such as quantizations, will normally introduce oscillations or limit cycles in the system. The magnitude of the oscillations will decrease with the quantization step. It is normally difficult to make an exact analysis of the influence of the converter resolutions even if crude results can be obtained using the method of describing function. The final choice of the resolutions has to be based on simulations or crude estimates.

Sampling Period Selection

In the discussion of the design of the controllers we have stated that the sampling interval is a design parameter in a sampled-data system. There are several issues that influences the selection of the sampling interval

- Response to reference signals (servo case)
- Influence of disturbances (regulator case)
- Computational load on the computer
- Digital filtering of signals

When we are considering the servo performance of the closed loop system, i.e. reference signal changes, it is possible to have quite long sampling periods. Comparatively low sampling rates can be used in control systems, compared with signal processing applications. The reason for this is that the processes to be controlled are normally of low pass character and the time constants are normally longer than the closed loop response time. The output is then mainly influenced by the pulse area over the sampling interval and is insensitive to the pulse shape. A second reason for being able to use long sampling intervals in the servo case is that the changes in the reference signal are synchronized with the sampling instants.

In the regulator case we want to eliminate the influence of disturbances acting on the system. The disturbances are not synchronized with the clock in the computer. This means that the reaction of the closed loop system will be very different if the disturbance starts just before a sample or just after a sample. In the latter case the disturbance will influence the process over a full sampling interval before any control actions can be taken to reduce the influence of the disturbance. To eliminate different types of disturbances it is necessary to increase the sampling rate for the regulator case compared with the servo case. Based on experience and simulations a common rule for the selection of the sampling period is to choose

$$\omega h = 0.1 \text{ to } 0.6$$

where ω is the desired natural frequency of the closed loop system. This implies that we are sampling up to 20 samples per period of the dominating mode of the closed loop system. It is important to note that the choice of the sampling period should be related to the desired properties of the closed loop system. When an analog controller is redesigned as in Chapter 8 the sampling period should be chosen according to (36).

The computers today are very fast and there should not be any problems with the computing capacity. However, with the fast computers we also want to control faster systems, which implies that the load on the computer still may be an issue. To get as many control loops as possible into the computer it may be necessary to decrease the sampling period. The load on the computer and dynamic scheduling of the tasks in the computer may otherwise introduce jitter in the sampling and even lost samples. See Chapter 13.

In connection with the discussion of the aliasing problem we found that it is always necessary to introduce antialiasing filters. The bandwidths of these filters should be tuned to the sampling rate such that the attenuation is sufficiently high above the Nyquist frequency. It is, however, difficult to retune the analog filters when the sampling period is changed. One solution is to sample the signals fast using a fixed analog filter tuned for the fast sampling period. A digital filter can then be constructed that removes the signals with frequencies above the Nyquist frequency and the filtered signal can be sampled with the desired sampling rate. The digital filters can easily be redesigned when the sampling rate is changed.

Saturations and Windup

In the discussion we have assumed that the processes to be controlled are linear systems in state-space or input-output form. There are, however, some common nonlinearities that should be considered. These are input saturations, rate limiters, hysteresis, and backlash. Input or control signal saturation are very common since the control signal can only be of a certain magnitude. For instance, a valve can only be fully open or fully closed and a motor can only rotate with a maximum speed in one or the other direction. The performance of the closed loop system can be very much influenced of the saturation when large inputs are demanded, especially if the controller has an integrator. This is called *integrator windup* and was discussed for the PID controller in Chapter 8.

The influence of the nonlinearities can be reduced by allowing only very cautious changes in the control signal, which results in conservative design of the controllers. The nonlinearities can also be taken into account by using nonlinear optimization techniques, but this results in quite complicated design methods. For input saturation there is a simple and effective way of reducing the effect of the combined integrator and saturation. Consider a controller in the input-output form (64). It is assumed that the controller contains an integrator, i.e. the R-polynomial contains the factor $(q-1)$. A block diagram of the controller is given in Figure 33(a). When the signal e is not equal to zero then the integrator will force the signal v to increase in magnitude even if the input to the process u already has saturated. This implies that the increase in v has no effect on the output of the process. The process will behave as an open loop system with a maximum or minimum input signal. The idea is now to stop the integrator from increasing its magnitude when the control signal saturates. Adding $A_{aw}(q)u(k)$, where $A_{aw}(q)$ is a stable polynomial, to both sides of (64) gives

$$A_{aw}(q)u(k) = T(q)u_c(k) - S(q)y(k) + (A_{aw}(q) - R(q))u(k)$$

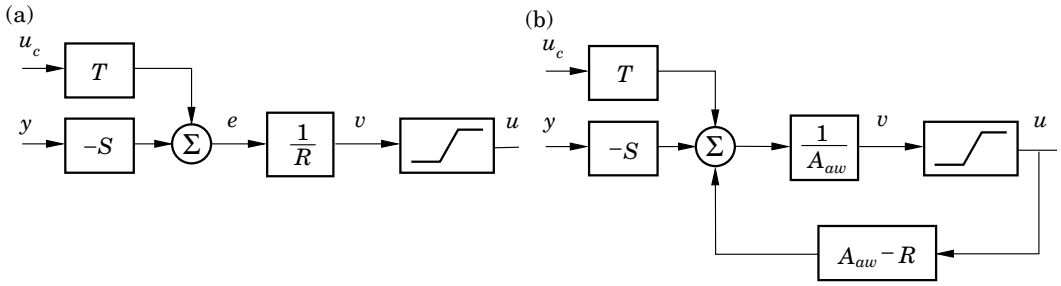

Figure 33 Block diagram of (a) the controller (64) and (b) the modification in (66) that avoids windup.

A controller with *antiwindup* compensation is then given by

$$\begin{aligned} A_{aw}(q)v(k) &= T(q)u_c(k) - S(q)y(k) + (A_{aw}(q) - R(q))u(k) \\ u(k) &= \text{sat}(v(k)) \end{aligned} \tag{66}$$

where $\text{sat}(v(k))$ is the saturation of the signal $v(k)$. A block diagram of this controller is shown in Figure 33(b). Compare Figure 23. The controller (66) is equivalent to (64) when the control signal is not saturating. When the feedback is broken due to the saturation then the signal v will not increase in magnitude since A_{aw} is a stable polynomial.

EXAMPLE 17—ANTIRESET WINDUP

Consider the double integrator plant representing the disk arm drive. A controller with integrator is designed for the process. Figure 34 shows the performance of the closed loop system with and without the antireset windup modification. With saturation but without antireset windup the output will have a large overshoot since the integrator in the controller will continue to increase after the saturation. The output need to be above the reference signal for some time to decrease the integrator value. With the antireset windup modification

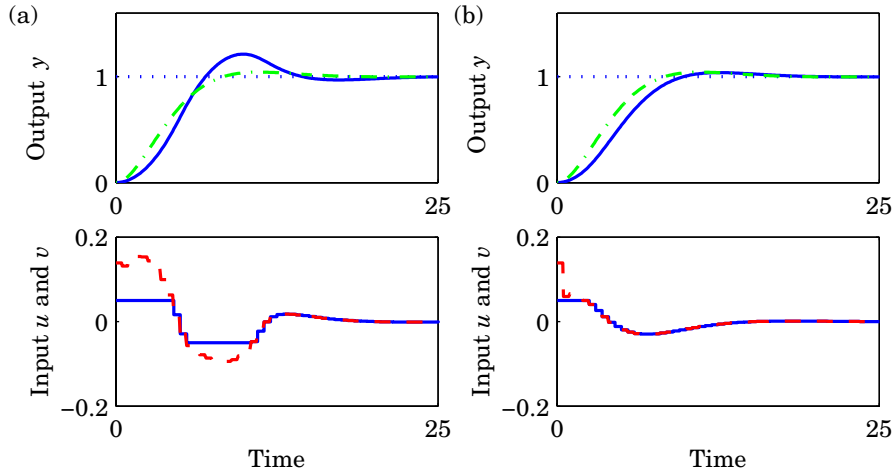

Figure 34 Control of the double integrator with a controller (a) without and (b) with antireset windup given by (66). The upper curves show the output (full/blue) and the behavior when there is no saturation (dash-dotted/green). The lower curves show the unsaturated input v (dashed/red) and the saturated input u (full/blue).

the integrator is modified such that the unsaturated signal v approaches the saturated input u. □

The example clearly shows that a simple modification of the controller improves the performance of the closed loop system when there are saturations in the control signal.

Summary

- Minimize the computational delay.
- Use an implementation that is insensitive for numerical errors, especially when the sampling interval is short.
- Select the sampling frequency in relation to the desired closed loop performance.
- It is utmost important to include antireset windup in the controller.

12. Real-time Implementation

Sampled-data control systems are implemented in different ways. Many are embedded systems where the computer is one component in a larger system. The systems are often implemented in microprocessors using a real-time operating system or kernel. Examples are systems in automobiles, robots, and home electronics. In process and manufacturing industries the systems have to be very flexible, since the number of control loops and their parameters often change. These control applications are often implemented using a real-time operating system with extensive graphical user interfaces. Often the systems are programmable allowing the end-user to program control applications using special, often graphical, domain-specific programming languages, e.g., the languages in the IEC 61131-3 standard.

Real-time Systems

A *real-time system* has to respond to external signals within a finite and specified period of time. For the real-time system it is important that the results are correct and that they are delivered within pre-specified times or deadlines. The system responds to events that can be periodic such as sampled-data control systems, or non-periodic such as alarms, operator interactions, or communication events. Several types of real-time systems have been defined. A *hard real-time system* is a system where it is absolutely imperative that the response occurs within the required deadline. Many hard real-time systems are *safety-critical* where life or serious damage is at risk if the deadlines are not met. Examples of hard real-time systems are aircraft fly-by-wire systems, vehicle brake-by-wire systems, and air traffic control systems. A *soft real-time system* is a system where the deadlines are important but where the system will still function correctly if deadlines are occasionally missed. Examples of soft real-time systems are web browsers, telephone exchanges, and interactive graphical user interfaces.

A common misconception is that real-time systems are equivalent to high-speed computations. This is not true. The important issue is that the real-time system should execute at a speed that matches, and makes it possible to fulfill, the timing requirements of the external system. In practical cases this, of course, means that execution speed is very important. It is, however, not this issue that makes real-time systems special.

During the last two decades real-time systems, in particular hard real-time systems, have emerged as a rapidly emerging research area within computer science. As motivating examples of hard real-time systems, sampled-data control systems are commonly used. In some cases control systems have hard deadlines, e.g., control of open-loop unstable systems. In many other cases control systems are not hard real-time systems, since slight variations in the sampling period or computational delay will in most cases only result in minor variations in the behaviour of the closed control loop that are well within the performance specifications. Nevertheless, the timing in the system and the interaction with the operator is crucial and must obey strict time constraints.

Virtually all real-time systems are inherently concurrent. The reason is that the real world is parallel. In any complex system a lot of events can occur at the same time. The work that has to be done to service the different events constitutes the set of tasks that have to be performed. It is also quite common to have multiple controller tasks within the same system, for example, different flow, pressure, level, and temperature control loops in process control applications and different position, velocity, and force control loops in mechatronics appli-

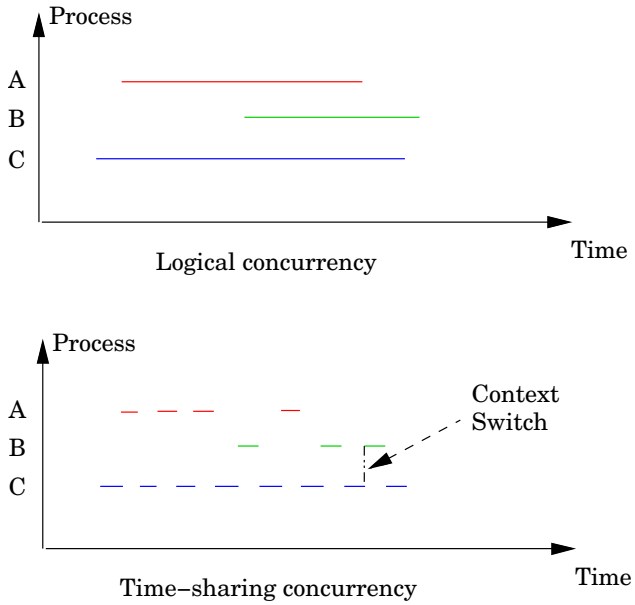

Figure 35 Logical and time-sharing concurrency

cations. The different controllers typically have different sampling intervals. In some cases the controller tasks are relatively independent of each other. In other cases the controller tasks need synchronization and communication, for example, in cascade controllers.

Implementation Techniques

If we have a massively parallel computer system then it is, at least theoretically, possible to assign one CPU to each task and to perform the execution associated with the events truly in parallel. However, when we have a limited number of CPUs or, in the common case, when we only have one CPU, we must in some way multiplex or interleave the execution of the different tasks onto the single CPU in such a way that the deadlines of all tasks still are met. This can be done using several techniques. The most primitive alternative is to let the programmer manually interleave the periodic tasks into a single sequential program, known as a *cyclic executive*. This very easily results in complex code that is difficult to understand and change. A better alternative is to automate the construction of the cyclic executive. This is what is done if off-line static scheduling techniques are used. The resulting system is time-driven. A similar approach is used within the *synchronous programming* languages, e.g., Esterel, Lustre, and Argos, which are compiled to a sequential finite state machine format. The advantage with static execution schedules is the high level of temporal determinism achieved. A disadvantage is that the resulting systems are quite rigid and the schedule typically has to be modified every time the control system is modified. This can only be used for embedded real-time systems that remain fixed for long periods of time.

For more dynamic applications it is necessary to use event-driven scheduling. In *concurrent programming* a real-time kernel or real-time operating system (RTOS) uses multiprogramming to multiplex the execution of the tasks on a single CPU. Hence, the available CPU capacity is shared between the different tasks. Ideally, this is done in such a way that the timing constraints of the the different tasks all are met, i.e., from an external, or logical, point of view it

appears as if the tasks execute truly in parallel. In reality the CPU switches between the processes. These switches are called *context switches*. This type of concurrency is also known as *time-sharing concurrency*. The situation is shown in Figure 35. The advantage with the concurrent approach is the flexibility that can be achieved. For example, new tasks can be added to the system at run-time. The concurrent approach is also what is dominating in industry and a large amount of commercial hardware and software systems are available. The main disadvantage is the nondeterminism. The event-driven execution makes it impossible to know which task that is executing at any given time. It also makes it difficult to guarantee that the timing constraints of the tasks are met. However, recent real-time scheduling analysis techniques have somewhat alleviated this situation making it possible to, at least theoretically, formally guarantee that the tasks meet their time constraints.

Concurrent Programming

In a real-time kernel or RTOS each task is represented by a process or thread. The term *process* is used when the tasks have separate address spaces and the term *thread* is used when the tasks reside within the same address space, making it easy to share code and variables. Here we will use the term process to cover both cases.

A process can be in one out of three internal states: running, ready, or blocked. On a single-CPU system only one process can be running at a time. Processes that are ready have access to all resources that they need to execute except the CPU. The ready processes are often stored in a queue called `readyQueue`, according to their priority. The running process is first in the queue, since it has highest priority. The processes that are blocked are waiting for something, e.g., sleeping for a certain time or waiting for some shared resource to become available. A context switch takes place every time the kernel changes the running process, for example, when the current running process decides to wait (changes to the blocked state) or when a process with higher priority than the running process becomes ready. During the context switch the context of the running process, e.g., the values of the CPU registers, is saved and the context of the process that will become the new running process is restored. In a preemptive real-time kernel the context switch takes place immediately, whereas in a non-preemptive kernel the running process may continue to execute until it reaches a preemption point where a context switch may occur or until it voluntarily releases the CPU.

The priority of a process reflects the importance of the time demands of the process. The higher priority the process has the more severe are its time demands. Normally, the priorities are fixed numbers that are assigned off-line. An alternative to priorities is to use deadlines. Here, each process has an associated deadline. Using *earliest deadline first (EDF) scheduling* the kernel always executes the process which is closest to its deadline. Hence, the deadline can be viewed as dynamic priority that changes as time proceeds.

Synchronization and Communication

Processes normally need to access different common resources. In many cases these accesses need to be done under mutual exclusion, i.e., only one process at a time may access the resource. Examples of resources can be external input-output (IO) units and, if the processes reside in a shared address space, non-reentrant code and accesses to shared variables. Shared variables are commonly used to implement communication between different threads. The sections in

code where shared variables are accessed are known as *critical sections*. A number of different synchronization schemes have been developed, e.g., semaphores, monitors, and mutexes. A process that needs access to a shared resource becomes blocked if a higher priority process currently holds the resource.

A real-time control system must communicate with the external environment. This is done through external hardware interfaces such as A-D and D-A converters, serial and parallel ports, and different bus interfaces. The computer control system may also communicate with the human users of the system through the keyboard, mouse, and screen.

The communication with the external interface is made either on request (polling) or using interrupts. Using *polling* the program repeatedly checks if the hardware interface has new data to report or not, or if the hardware interface is finished sending data or not. If the system has many external units a solution based on polling can become very inefficient. Using *interrupts* it is the external device that informs the system that some significant event has occurred, e.g., when new data is available. The interrupt is generated by the external device when the event has occurred. Through hardware support the interrupt will asynchronously interrupt the execution of the system and the execution will continue in an interrupt handler where the execution that is needed to service the event is performed. When the interrupt handler is finished the execution continues at the place where the interrupt occurred in the currently executing process, unless a context switch was initiated from the interrupt handler.

Clock interrupts are of special importance in a real-time system. The clock interrupts are used to keep track of time, something which is necessary for processes that need to wait a certain time or execute at a regular frequency. Clock interrupts are generated from the hardware clock of the CPU every *tick*. The tick value determines the smallest time resolution of the kernel. Processes that are waiting for a specific time are stored in a time-sorted queue, often called `timeQueue`. In the clock interrupt handler the processes that have finished waiting are extracted from `timeQueue` and inserted into `readyQueue`. This may lead to a context switch. Instead of having clock interrupts every tick it is common to use a hardware timer unit. The hardware unit is set up to generate a clock interrupt every time a process needs to be woken up.

In order for an operating system to qualify as a real-time operating system several criteria must be fulfilled. To make use of the available scheduling theory it is important that the kernel provides worst-case bounds on the execution times of the different kernel primitives. Context switch times and interrupt handling need to be fast. Sufficient number of distinct priority levels, support for preemptive scheduling, and the availability of a real-time clock with high enough granularity are further requirements.

Periodic Controller Tasks

Periodic controller tasks can be implemented in different ways. To begin with we will assume that a controller task is represented by a single execution unit, e.g., a thread. A few modern real-time operating systems provide operating support for scheduling periodic tasks. In this approach the task is implemented as a procedure or method that is registered with the kernel to execute at a certain period. It is then the responsibility of the RTOS to periodically call this procedure. Similar features are provided by the recent Real-Time Java standard. In most RTOS periodic tasks are instead implemented as self-scheduling tasks, that themselves contain calls to timing primitives.

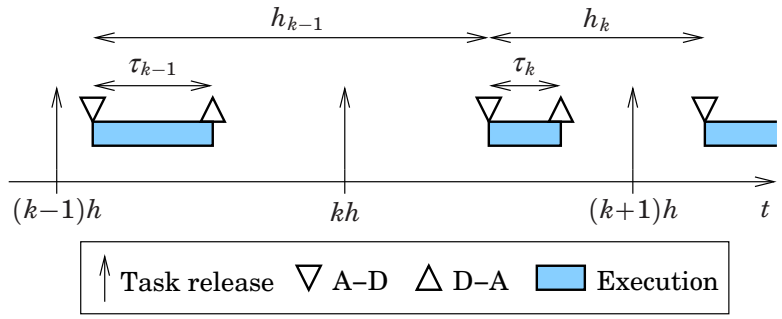

Figure 36 Control loop task model

A real-time kernel normally provides two types of primitives that a task can use to wait a certain time. A relative wait statement delays the calling task for an interval relative to the current time. These primitives are known as *delay*, *sleep*, or *waituntil*. An absolute wait statement instead delays the calling task until an absolute point in time. These primitives are known as *delayuntil*, *sleepuntil* or *waituntil*. The absolute delay primitives are more powerful than the relative delay primitives in the sense that it possible to implement a relative delay using an absolute delay primitive but not the opposite way around. The absolute delay statements are also better for implementation of self-scheduling controller tasks.

When implementing periodic control loops it is important that they execute at equidistant time intervals. The following definitions of the critical timing parameters of a control loop will be used in the sequel, see Figure 36. The *task release time* is the time when the task is woken from its sleep. The sampling interval h is the time interval between A-D conversions. The computational delay or input-output latency τ is the time interval between the A-D and the D-A conversions. The sampling jitter is the variation in sampling interval and the computational delay jitter (input-output latency jitter) is the variation in τ. Ideally the sampling jitter should be zero and the computational delay should be jitter-free. If the computational delay is small it can normally be ignored in the control design. If it is jitter-free it can be compensated for off-line in the design.

There are several reasons for jitter in sampling interval and control delay. Some examples are data-dependent execution times, suspensions by higher priority tasks, blocking caused by lower priority tasks, communication delays in networked control loops, and jitter caused by the hardware and software platform used.

The simplest way of implementing a periodic controller loops is as follows:

```
while (true) do {
  executeController();
  sleep(h);
}
```

This approach does not work. Since the execution time of `executeController` is not taken into account the actual sampling period will always be larger than h. The approach also does not take variations in the execution time into account.

A slightly better approach is the following:

```
while (true) {
  start = getCurrentTime();
  executeController();
```

```
  end = getCurrentTime();
  sleep(h - (end - start));
}
```

The idea here is to take the execution time into account by only waiting for the time that remains of the sampling interval. However, if the task is suspended after it has recorded the end time but before the start of the sleep, the approach will not work.

A better solution is achieved by using an absolute wait primitive as follows:

```
while (true) {
  start = getCurrentTime();
  executeController();
  nexttime = start + h;
  sleepUntil(nexttime);
}
```

This solution is better but it still has problems. The main problem is that all delays that occur between `sleepUntil` and `start = getCurrentTime();` are not taken into account. Therefore it is better to only measure the current time once, before the start of the periodic task:

```
nexttime = getCurrentTime();
while (true) {
  executeController();
  nexttime = nexttime + h;
  sleepUntil(nexttime);
}
```

If we assume that `sleepUntil` has the semantics that an attempt to sleep until a point in time that already has elapsed is equivalent to a no operation this solution will also try to catch up if it occasionally takes longer time to execute `executeController` than the sampling interval. Hence, a too long sampling interval will be followed by a too short sampling interval in such a way that the average sampling period is equal to h. The value of this behaviour for control loops is questionable. An alternative approach is to simply accept an occasional too long sampling interval and then try to execute at the desired sampling interval from then on. This behaviour is obtained by reseting `nexttime` to the current time inside `sleepuntil` every time an overrun occurs.

To reduce the computational delay it is common to split the execution of the controller into two parts, the `calculateOutput` part and the `updateState` part and to perform the D-A conversion immediately after the execution of `calculateOutput` as discussed in Chapter 11. The pseudo-code is shown below:

```
nexttime = getCurrentTime();
while (true) {
  AD_conversion();
  calculateOutput();
  DA_conversion();
  updateState();
  nexttime = nexttime + h;
  sleepUntil(nexttime);
}
```

To further reduce the computational delay the priority of the task can be raised within `calculateOutput`. The four different parts of the algorithm (A-D conversion, `calculateOutput`, D-A conversion, and `updateState`) can be viewed as subtasks. To improve the determinism of the IO operations the A-D and D-A conversions are sometimes performed in hardware or in the interrupt handler.

Scheduling

For hard real-time systems it is crucial that the timing requirements always are met. Hence, it is necessary to perform an off-line analysis (that is performed before the system is started) that guarantees that there are no cases in which deadlines are missed. If deadlines could be missed then the system should not be started at all.

In scheduling theory we assume that we have events that occur and require computations. Associated with an event is a task that executes a piece of code in response to the event. The events could be periodic, sporadic, or aperiodic. A sporadic event is non-periodic but have a maximum frequency that guarantees that only the task associated with one event instance is active at a time. An aperiodic event has an unbounded release frequency and could have many active associated tasks.

Each event has an associated deadline. This is an upper bound on the allowed time taken to execute the piece of code associated with the event. Each event also has a required computation time. This is the worst-case CPU time it takes to execute the piece of code associated with the event in the absence of other competing tasks. Obtaining this time can in practice be quite difficult. The problem is known as *execution time estimation* or *program timing analysis*.

Two main approaches to execution time estimation exist: measurements and analysis. When measuring execution times the code is compiled and run with measuring devices (a logical analyzer) connected. A large set of input data is used and the longest execution time measured is defined as the required computation time for the code. The main problem with this approach is that it can be overly optimistic. There are no guarantees that the longest execution time really has been encountered. The main reason for this is that execution times are data dependent and an exhaustive testing using all combinations of input data is impossible in practice. Another problem are nondeterministic features of modern CPUs, e.g., caches and pipelines.

The aim of execution time analysis is an automatic tool that takes the source code and formally correctly decides the longest execution time. Although this has been an active research area for some time now, still very few timing analysis tools are available. The main problem with the approach is that it can be overly pessimistic. The longest execution time calculated by the method can be much longer than the actual execution time. This is specially the case if the code contains many branches. Since scheduling analysis requires a worst-case bound this approach is the only that is formally correct. A problem with timing analysis is that the result is compiler dependent. Hence, the analysis must be performed on machine code. Loop iterations and recursions typically require the programmer to annotate the code with worst case bounds on the number of loops or recursions. Another problem is dynamic memory allocation. The time taken to allocate dynamic memory is often unknown. For example, memory allocation may invoke garbage collection.

A general problem with execution time analysis is that the rate at which computer architects introduce new hardware features increases faster than the increase in the number of features that can be handled by the analysis. Hence,

the gap in speed between general purpose hardware and hardware, which safely can be used for hard real-time applications, increases rather than decreases. Due to the lack of analysis tools the dominating approach in industry is to use measurements and then to add an appropriate safety margin to the measured worst case time.

The most commonly used scheduling approach is *preemptive fixed priority scheduling*. This is used in the majority of the commercially available RTOS and real-time kernels. For this type of scheduling a considerable amount of theory is available that can be used to decide whether a given task is schedulable or not. In the *rate monotonic analysis* priorities are assigned to tasks monotonically with task rate. A task with a shorter period is assigned a higher priority.

Under the following assumptions a simple sufficient schedulability result holds for the case of rate-monotonic task priority assignment:

- only periodic tasks with unique periods,
- tasks may not suspend themselves or be blocked by other tasks due to synchronization,
- the task deadlines equal the task periods,
- the real-time kernel is "ideal" in the sense that that context switches and interrupt handling take zero time

The result says that, for a system with n periodic tasks, all tasks will meet their deadlines if the total CPU utilization U of the system satisfies the bound

$$U = \sum_{i=1}^{i=n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1), \tag{67}$$

where C_i is the worst-case execution time of task i, T_i is the period of task i, and n is the number of tasks.

The utilization U determines the CPU load. For example, if we have only one task with a period of 10 ms and with a required computation time of 10 ms then the CPU will spend all its time executing that task, hence, the utilization will be 100%. If the CPU utilization is larger than the bound in (67) the task set may still be schedulable. As $n \to \infty$, the utilization bound $\to 0.693$. This has led to the simple rule-of-thumb that says that

"If the CPU utilization is less than 69%, then all deadlines are met".

The rate-monotonic scheduling theory is quite mature. Schedulability tests that are both sufficient and necessary have been developed. Results are available that allow relaxations of the quite rigid assumptions above. For example, scheduling results are available that take blocking due to synchronization into account, the deadlines must not equal the task periods, context switches and interrupt handling can be taken into account. The scheduling theory can also be applied to distributed systems where the communication uses the *Controller Area Network (CAN)* protocol. CAN is very common in the automotive industry.

Formal analysis methods are also available for EDF scheduling. With the same assumptions as in the sufficient rate monotonic result above the following necessary and sufficient condition holds for schedulability:

$$U = \sum_{i=1}^{i=n} \frac{C_i}{T_i} \leq 1 \tag{68}$$

If the CPU utilization U of the system is not more than 100% then all deadlines will be met.

The main advantage with EDF scheduling is that the processor can be fully utilized and still all deadlines can be met. More complex analysis exists that loosens many of the assumptions above. Another advantage is that it is often more intuitive to assign deadlines to tasks than it is to assign priorities. Assigning priorities requires global knowledge about the priorities of all the other tasks in system, whereas assigning deadlines only requires local knowledge. Another difference between priority and deadline-based scheduling concerns the behaviour during overruns, when a task has executed longer than its allowed maximum time and the task set no longer is schedulable. A priority-based approach will favor high-priority tasks over low-priority tasks, with the possible consequence that low-priority tasks may not execute at all; they are starved. Using deadline-based scheduling the available CPU time will be distributed among the tasks in a more fair way. Depending on the application this may or may not be a desired feature. The major drawback with deadline-based scheduling is the lack of commercial products supporting it.

The scheduling theory described above is based on the worst-case scenario that all the periodic tasks are released simultaneously, at the so called *critical instant*. If the task set is schedulable for this case it can be shown that it is also schedulable for all other cases. If offsets are introduced among the tasks that prevent the critical instant from occurring, the schedulability of the task set increases. A number of scheduling results have also been derived for subtask scheduling. Here, a task is divided into serially executed subtasks each characterized by a worst-case execution time, a priority, and a deadline. This model matches the subtask view of a controller task very well.

The rate monotonic approach has gained a lot of attraction. It has been adopted by several major companies, e.g., IBM, NASA, and European Space Agency, and the theory has influenced the development of real-time programming languages, e.g., Real-Time Java, and standards, e.g., Real-Time POSIX and UML. The original results were based on a set of rather restrictive assumptions and requirements. Most of these have subsequently been removed or relaxed. To apply the theory it is essential that the estimated worst-case computation requirements are known for all tasks. As discussed before it is difficult to obtain these numbers. If a formal approach is used the bounds obtained are often very pessimistic. This means that the worst-case execution times are much larger than those that occur in practice. Also, the different factors used in the analysis are also all worst-case numbers. This means that the whole analysis is based upon a succession of worst-case assumptions. Therefore, there is a large risk that many task sets will be considered unschedulable by the analysis although they would run perfectly well in reality. Another objection against the approach that sometimes is raised is that with the current low price on hardware it is just as feasible to simply add another CPU to the system if there is even the slightest risk that some of the timing requirements are not met. However, for many embedded systems this argument is not true.

Summary

- A time-driven implementation maximizes the temporal determinism.
- Deterministic execution of controller tasks in an event-driven system requires a real-time operating system or kernel.

- Absolute timing primitives are needed to implement self-scheduling controller tasks correctly.
- Using scheduling theory it is possible to formally guarantee that the time constraints of real-time tasks are met, provided that the worst-case execution times are known.
- Timing compensation can make it possible to use COTS components for implementing real-time control systems.
- The ratio between the sampling periods and the scheduling of different tasks are important in multi-rate systems.

13. Controller Timing

The rapid increase in average-case execution speeds and decrease in cost for commercial off-the-shelf (COTS) components, e.g., processors and operating systems, make them interesting platforms also for real-time control systems, at least for non-safety-critical applications, even if hard worst-case guarantees cannot be obtained. To use these components it is important to understand how jitter in sampling interval and computational delay effect control performance.

Consider the general linear continuous-time plants described by

$$\begin{aligned}\frac{dx(t)}{dt} &= Ax(t) + Bu(t) + B_v v(t)\\ y(t) &= Cx(t) + e(t)\end{aligned}$$

and controlled by a general linear discrete-time controllers on the form

$$\begin{aligned}x_c(kh+h) &= \Phi_c x_c(kh) + \Gamma_c y(kh)\\ u(kh) &= C_c x_c(kh) + D_c y(kh)\end{aligned}$$

when sampling intervals and computational delays are given by probability density functions. If a quadratic cost function

$$J = \lim_{t\to\infty} \frac{1}{t} \mathop{\mathrm{E}}_{v,e}\left\{ \int_0^t \left(x^T(s)Q_1 x(s) + u^T(s)Q_2 u(s)\right) ds \right\}$$

is used as a performance indicator it is possible to analytically evaluate the performance for of the closed loop system. This is done by sampling the plant and the cost function, formulating the closed-loop system (a jump-linear system), and computing the stationary covariance of the states. These calculations can be packaged in Matlab.

EXAMPLE 18—PERFORMANCE ANALYSIS

Consider the inverted pendulum process shown in Figure 37. The equations of motion are given by

$$\frac{d^2\theta}{dt^2} = \omega_0^2 \sin\theta + u\,\omega_0^2 \cos\theta$$

where $\omega_0 = \sqrt{g/l}$ is the natural frequency of the pendulum.

Using the state vector $x = [\,\theta \quad \dot{\theta}\,]^T$ and linearizing around the upright equilibrium gives the state-space model

$$\begin{aligned}\frac{dx}{dt} &= \begin{bmatrix} 0 & 1 \\ \omega_0^2 & 1 \end{bmatrix} x + \begin{bmatrix} 0 \\ \omega_0^2 \end{bmatrix} u\\ y &= [\,1 \quad 0\,]\, x\end{aligned}$$

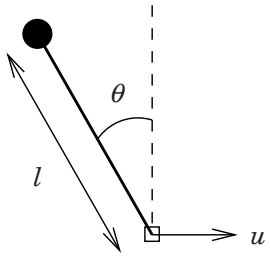

Figure 37 Inverted pendulum

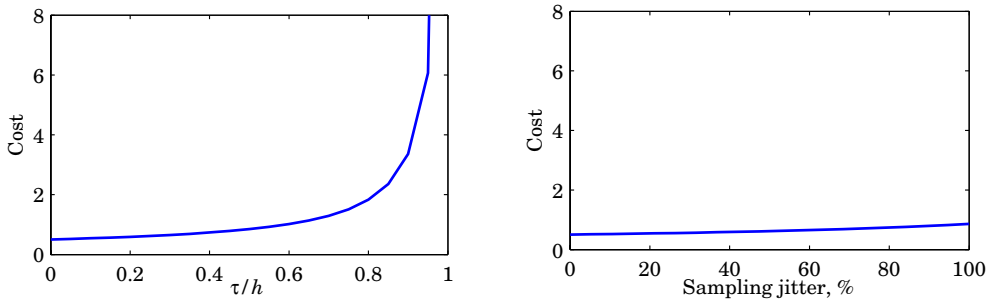

Figure 38 Cost as a function of computational delay (left) and as a function of sampling interval jitter (right).

For this model a discrete state-feedback controller $u = -Lx$ is designed using pole-placement.

The cost $J = \mathrm{E}\,\theta^2$ as a function of the normalized computational delay τ is shown to the left in Figure 38. The performance degrades as the delay increases. The loop becomes unstable ($J \to \infty$) when $\tau \to h$. To the right in Figure 38 the same cost function is shown as a function of the sampling interval jitter. Here, the performance degrades only slightly as the jitter level increases. □

The traditional approach in the real-time control community is to design and implement the systems in such a way that the temporal determinism is maximized. An alternative approach is to view the temporal non-determinism implied by COTS components as an uncertainty, similar to a plant uncertainty, or a disturbance and then design the control systems to be robust against the uncertainty or actively compensate for the temporal disturbances on-line. This also opens up for more flexible system architectures where the controllers and the implementation platform negotiate on-line about the access to shared resources such as CPU time and communication bandwidth. In this approach the control performance can be regarded as a *Quality-of-Service (QoS)* parameter.

A reason why COTS components in many cases can be used for control system implementation is the inherent robustness of most feedback control schemes. However, in many cases better performance can be obtained of if the controller is allowed to actively compensate for the jitter. A prerequisite for this is that the implementation platform provides the means to measuring time with sufficient resolution.

Compensation for sampling interval jitter can be performed in several ways. A simplistic approach is to keep the sampling interval, h, as a parameter in the the design and to update this every sample. For a sampled control system this corresponds to resampling the continuous-time system. For a discretized continuous-time design the changes in sampling interval only influence the approximation of the derivatives.

EXAMPLE 19—SAMPLING JITTER COMPENSATION

Consider PD control of a DC servo. The goal of the control is to make the servo position, $y(t)$, follow the reference position, $u_c(t)$, as closely as possible. Let the servo be described by the continuous-time transfer function

$$G(s) = \frac{1000}{s(s+1)}.$$

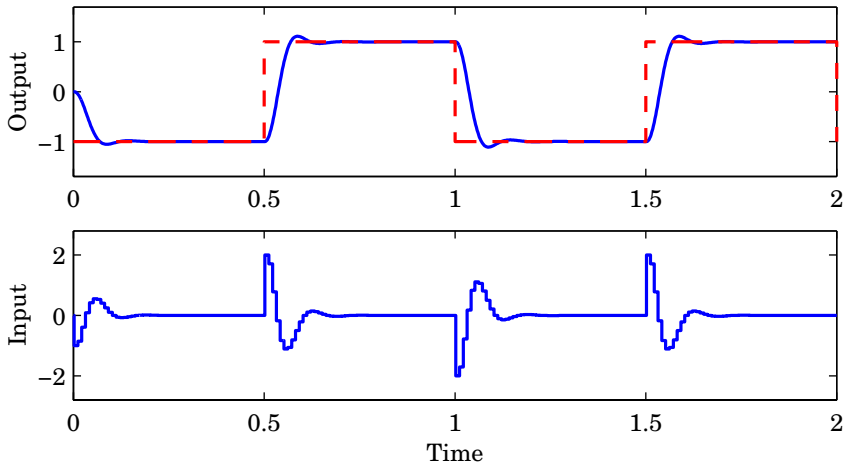

Figure 39 Output (solid/blue) and reference signal (dashed/red) are shown in the upper plot and the control signal in the lower plot. When no sampling jitter is present, the control performance is good.

A good implementation of the PD controller, which includes filtering of the derivative part, is, compare Chapter 8,

$$\begin{aligned} P(kh) &= K(u_c(kh) - y(kh)), \\ D(kh) &= a_d D(kh-h) - b_d(y(kh) - y(kh-h)), \\ u(kh) &= P(kh) + D(kh), \end{aligned}$$

where $a_d = T_d/(T_d + Nh)$, $b_d = KT_dN/(T_d + Nh)$.

A nominal sampling period of $h = 10$ ms is chosen, and the PD controller is tuned to give a fast and well-damped response to set-point changes. The resulting parameters are $K = 1$, $T_d = 0.04$, and $N = 30$. The parameters a_d and b_d are normally pre-calculated, assuming that the sampling interval is constant.

A first simulation of the closed-loop system, where there is no jitter in the sampling interval, is shown in Figure 39. The controller behaves as expected, and the performance is good. A second simulation, where the actual sampling interval varies randomly between $h_{min} = 2$ ms and $h_{max} = 18$ ms, is shown Figure 40. The sampling jitter causes the controller to repeatedly take either too small or too large actions. The resulting performance is quite poor. This is especially visible in the control signal. Finally, the controller is redesigned to compensate for the jitter. This is done by measuring the actual sampling interval and recalculating the controller parameters a_d and b_d at each sample. Figure 41 shows that this version of the controller handles the sampling jitter well. □

Similar techniques as in Example 19 can also be used to compensate for parts of the computational delay, particularly the part of the delay between the A-D conversion and the start of the control calculations. This delay can be a communication delay in a networked control loop where the sensor is located on a different node than the controller. The delay can also be the delay from the release time of the task to the start of the calculations of the task when the A-D conversion is performed in the interrupt handler. A possibility here is to use observer techniques to estimate what the values of the measured signals are at the start of the controller calculations.

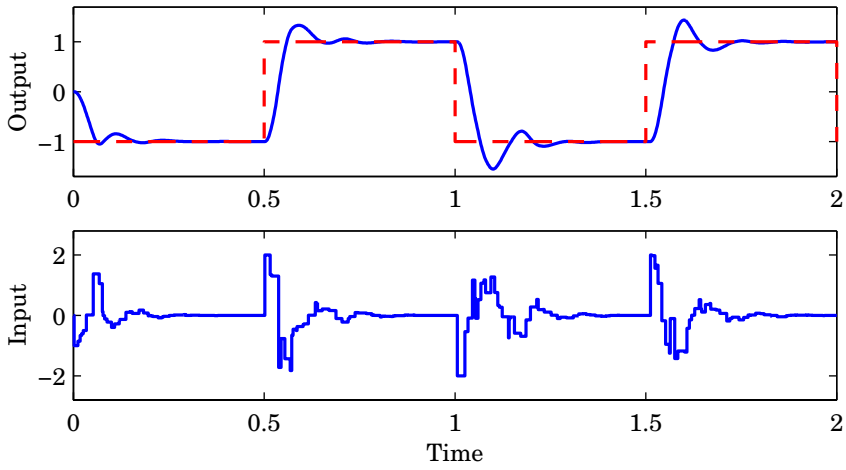

Figure 40 Same as Figure 39, but with sampling interval jitter. Sampling jitter causes the control performance to degrade.

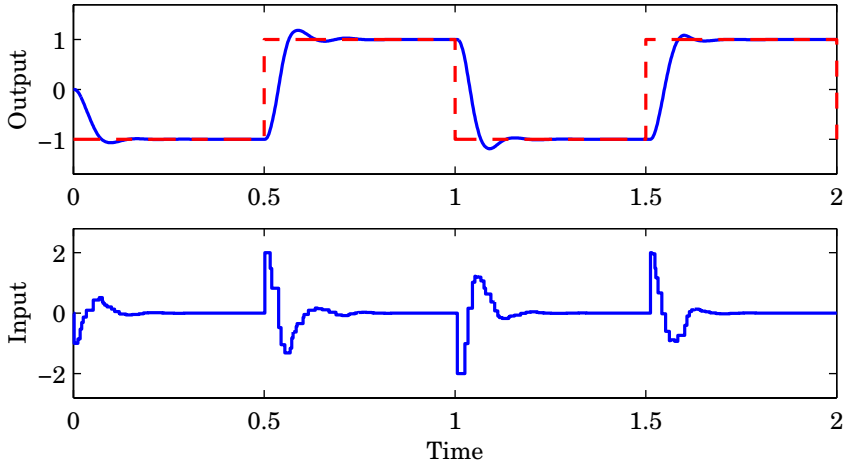

Figure 41 Same as Figure 40, but with compensation for the sampling jitter.

The possibility for controllers to compensate for timing variations can also be used as a way to increase flexibility. Assume that the computer contains a set of controller tasks. The number of controller tasks and their execution time bounds may change over time. The latter can be due to too optimistic execution time bounds causing occasional overruns, or due to different operation modes in the controller. As the workload changes, the scheduler may then adjust the task attributes, e.g., the periods, of the controller tasks in order to optimize global control performance under the constraint that the task set should remain schedulable.

A prerequisite for this type of on-line integration of control and scheduling is that it is possible to make an integrated off-line design of control algorithms and scheduling algorithms. Such a design process should allow an incorporation of the availability of computing resources into the control design. This is an area where, so far, relatively little work has been performed.

A scheduler that on-line adjusts task attributes in order to optimize control performance or QoS can be interpreted as a controller in itself, a *feedback*

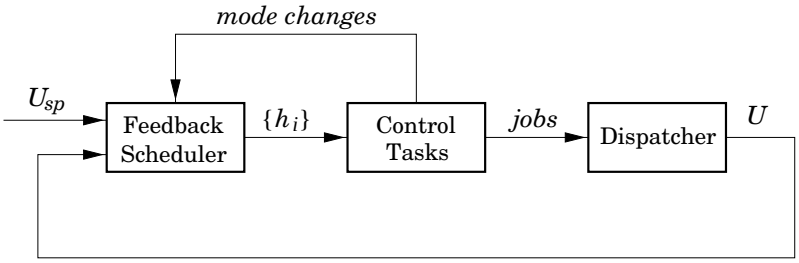

Figure 42 Feedback scheduler structure where U_{sp} is the set-point for the CPU utilization.

scheduler. Important issues that then must be decided are what the right control signals, measurement signals, and set-points are, what the control structure should be, and which process model that may be used. The block diagram of a feedback scheduler is shown in Figure 42.

The goal of the scheduler is to keep the CPU utilization, U, at a desired value. To do this it adjusts the sampling frequencies of the controller tasks. Feedforward is used to compensate for mode changes. A feedback scheduler can reduce the workload required by a controller task in two ways: by increasing the sampling interval or by decreasing the maximum allowed execution time for the task. The latter would be a possibility for controllers that can be expressed on iterative form, i.e., controllers that monotonously improve the control performance as a function of their allotted execution time.

The communication between different nodes in the control system may also cause delays. In most cases these delays are small compared with the time scale of the processes, but poorly choices of the sampling rates in, for instance, multi-rate systems, can create long delays in the transfer of information between different parts of the system.

EXAMPLE 20—DELAYS IN MULTI-RATE COMMUNICATION NETWORKS

Consider a simple distributed control and communication system as shown in Figure 43. The system consists of a controller node and an IO unit which are connected over a field-bus. The process outputs are measured and the control signals are applied through the IO device. The communication to and from the control node can be made over a local area network or a field-bus.

In the IO module the output of the process is A-D converted, and the result is copied into memory cell $m1$. Synchronized with this is the D-A conversion which writes the content of $m4$ to the process. Both these events occur every h_2 time unit, however, they may not necessarily be completely synchronous, that is there may be a phase difference between the input and the output. In a similar way, both the write bus, transferring values from $m1$ to $m2$, and the read bus, transferring values from $m3$ to $m4$, share the same timing, although there may

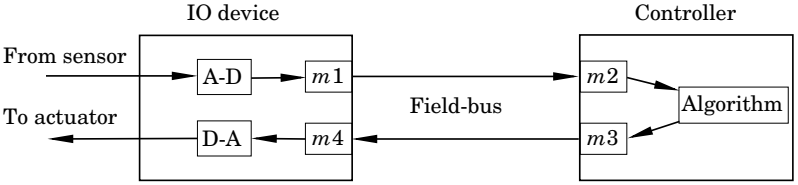

Figure 43 Control over a layered communication net.

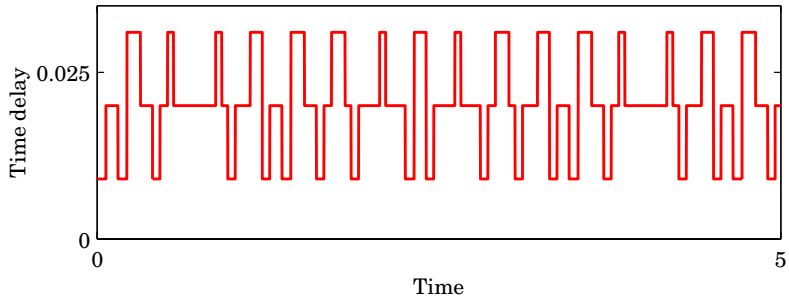

Figure 44 Time delays due to multi-rate sampling in a layered communication network. The system in Figure 43 is simulated with $h_0 = 50$, $h_1 = 12$, and $h_2 = 11$. The phase shift between the A-D and D-A converters is $\theta_1 = 7$ and in the field bus it is $\theta_2 = 9$.

also be a phase difference between the two. The sampling period is h_1. Finally, the controller is sampling and writing back to the field-bus with a sampling period of h_0.

Figure 44 shows the time history of the total delay from the A-D to the D-A converter from a simulation of the system in Figure 43. The total delay can be surprisingly long compared with the sampling interval of the controller even if the converters and the field bus have short sampling periods. The reason is that the ratio of the sampling intervals are not integers. This implies that the sampling times are "sliding" with respect to each other. The choice of the sampling intervals and the scheduling of the different tasks are thus very crucial in multi-rate systems. □

Summary

- It is important to assure good timing between the different processes of the control system.
- Timing compensation can make it possible to use COTS components for implementing real-time control systems.
- The ratio between the sampling periods and the scheduling of different tasks are important in multi-rate systems.

14. Research Issues

Computer control systems is quite a mature field. There are, however, two important research problems that do not have satisfactory solutions: multi-rate sampling and sampling of nonlinear systems. There are several approaches to control of multi-rate systems. They are all based on the same idea; to find a basic sampling period h such that all rates are integer multiples of h. It is also assumed that all samplings are synchronized. The system is then resampled with period h. This leads to a linear system of high dimension. The difficulty with this approach is that the dimension of the system changes drastically with minor changes in the sampling rate. Sampling of nonlinear systems is another problem which has not been solved satisfactorily, this is serious because it requires a lot of ad hoc approaches when applying the powerful theory of nonlinear systems. The standard approach is to sample fast and to use simple difference approximations. It seems reasonable that varying sampling rates should be used when dealing with nonlinear systems. A fruitful approach may be to use results from numerical integration of ordinary differential equations. Recent results where control theory has been used to construct new algorithms for step length adjustment in algorithms for integration of differential equations is an indication that it may be very fruitful to combine the fields.

The continued advances of computing and communication will also have strong impact on computer control systems as has been indicated in Chapters 12 and 13.

Acknowledgments

We would like to express our acknowledgments to many colleagues and researchers, who have given us much insights into the area of computer-controlled and real-time systems. Over the years our research on computer-controlled and real-time systems has been supported from the agencies Nutek/Vinnova and TFR/VR. This support has been instrumental in the development of our knowledge in these areas. Thanks also to Bo Lincoln for providing the example on the DVD player and to Anton Cervin for providing the timing compensation and performance analysis examples. A special thank to Prentice Hall Inc. for the permission to use original and/or modified figures from the book Åström and Wittenmark (1997): *Computer-Controlled Systems* in this paper (original figures are: 3, 9, 13, 14, 15, and 29 and modified figures are: 2, 4, 5, 6, 7, 10, 11, 12, 16, 17, 18, 19, 22, 23, 26, 27, and 34).

Notes and References

The basic theory of computer-controlled systems is found in Åström and Wittenmark (1997), Franklin *et al.* (1998), and Middleton and Goodwin (1990). General design methods for continuous-time and discrete-time control can be found in the previous references and in Glad and Ljung (2000), Goodwin *et al.* (2001), and Zhou and Doyle (1998). Analysis, tuning rules and implementation of PID controllers are presented in Åström and Wittenmark (1997) and Åström and Hägglund (1995).

The complex issue of frequency analysis of sampled-data systems is discussed in Araki and Ito (1993), Chen and Francis (1995), and Yamamoto and Khargonekar (1996).

Multirate systems and sampled-data control of nonlinear systems are discussed, for instance, in Kranc (1957), Crochieve and Rabiner (1983), and Monaco and Normand-Cyrot (2001).

The lecture notes Årzén (2001) and the books Shin and Krishna (1997), Buttazzo (1997), Liu (2000), and Burns and Wellings (2001) give extensive information on different aspects on controller implementation issues, real-time systems, and real-time scheduling.

There are also websites that contains material on sampled-data control. For the book Åström and Wittenmark (1997) material for a course is available at `http://www.control.lth.se/ccs` and for Goodwin *et al.* (2001) at `http://csd.newcastle.edu.au/control/index.html`.

At `http://www.engin.umich.edu/group/ctm/digital/digital.html` Matlab tutorials for digital control are available.

A dictionary where many of the terms in instrumentation and control are defined is found at `http://www.iica.org.au/dict/quikref.htm`.

The Matlab files that are used to generate all the simulations in this paper are available at `http://www.control.lth.se/ccs/ifacfiles.zip`.

Bibliography

Araki, M. and Y. Ito (1993): "Frequency-response of sampled-data systems I: Open-loop considerations." In *Preprints of the 12th IFAC World Congress*, vol. 7, pp. 289–292. Sydney.

Årzén, K.-E. (2001): *Real-Time Control Systems*. Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Åström, K. J. and T. Hägglund (1995): *PID Controllers: Theory, Design, and Tuning*. Instrument Society of America, Research Triangle Park, North Carolina.

Åström, K. J. and B. Wittenmark (1997): *Computer Controlled Systems*, 3rd edition. Prentice Hall, Upper Saddle River, N.J.

Burns, A. and A. Wellings (2001): *Real-Time Systems and Programming Languages: Ada 95, Real-Time Java and Real-Time POSIX*, third edition. Addison-Wesley.

Buttazzo, G. C. (1997): *Hard real-time computing systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, Boston, MA.

Chen, T. and B. A. Francis (1995): *Optimal Sampled-Data Control Systems*. Springer-Verlag, London.

Crochieve, R. E. and L. R. Rabiner (1983): *Multirate Digital Signal Processing*. Prentice Hall, Upper Saddle River, N.J.

Franklin, G. F., J. D. Powell, and M. L. Workman (1998): *Digital Control of Dynamic Systems*, 3rd edition. Prentice Hall, Upper Saddle River, N.J.

Glad, T. and L. Ljung (2000): *Control Theory — Multivariable and Nonlinear Methods*. Taylor and Francis, London.

Goodwin, G. C., S. F. Graebe, and M. E. Salgado (2001): *Control System Design*. Prentice-Hall, Upper Saddle River, N.J.

Kranc, G. M. (1957): "Input-output analysis of multirate feedback systems." *IRE Trans. Automat. Control*, **AC-3**, pp. 21–28.

Liu, J. W. (2000): *Real-Time Systems*. Prentice Hall, Upper Saddle River, N.J.

Middleton, R. H. and G. C. Goodwin (1990): *Digital Control and Estimation. A Unified Approach*. Prentice-Hall, Upper Saddle River, N.J.

Monaco, S. and D. Normand-Cyrot (2001): "Issues on nonlinear control." *European Journal of Control*, **7**, pp. 160–177.

Shin, K. G. and C. M. Krishna (1997): *Real-Time Systems*. McGraw-Hill, New York.

Yamamoto, Y. and P. Khargonekar (1996): "Frequency response of sampled-data systems." *IEEE Trans. Automat. Control*, **AC-41**, pp. 166–176.

Zhou, K. and J. C. Doyle (1998): *Essentials of Robust Control*. Prentice-Hall, Upper Saddle River, N.J.

About the Authors

Björn Wittenmark was born in Växjö, Sweden, in 1943. He obtained the M.Sc. degree in Electrical Engineering in 1966 and the Ph.D. degree in Automatic Control in 1973, both from Lund Institute of Technology. Since 1989 he is a Full Professor at the Department of Automatic Control at Lund Institute of Technology, Lund, Sweden. His main research interests are in the fields of adaptive control, sampled-data control, and process control. He has written numerous papers in these areas and is the coauthor and editor of eight books.

He has written the books *Computer Controlled Systems* and *Adaptive Control* (both coauthored with Karl J. Åström). Wittenmark became a Fellow of IEEE in 1991 and is a member of the Royal Physiographic Society since 1997.

Karl J. Åström was educated at The Royal Institute of Technology in Stockholm. During his graduate studies he worked on inertial guidance for the Research Institute of National Defense in Stockholm. After working five years for IBM in Stockholm, Yorktown Heights and San Jose he was appointed Professor of the Chair of Automatic Control at Lund University in 1965, where he built the Control Department from scratch. Åström has held many visiting appointments in Universities in Europe, USA and Asia. Åström has broad interests in automatic control including, stochastic control, modeling, system identification, adaptive control, computer control and computer-aided control engineering. He has supervised 44 PhD students, written six books and more than 100 papers in archival journals. He has several patents one on automatic tuning of PID controllers jointly with T. Hägglund has led to substantial production in Sweden. Åström is a member of the Royal Swedish Academy of Engineering Sciences (IVA) and the Royal Swedish Academy of Sciences (KVA) and a foreign member of the US National Academy of Engineering, the Russian Academy of Sciences and the Hungarian Academy of Sciences. Åström has received many honors among them four honorary doctorates, the Quazza Medal from IFAC, the Rufus Oldenburger Medal from ASME, the IEEE Field Award in Control Systems Science and the IEEE Medal of Honor.

Karl-Erik Årzén was born in Malmö, Sweden, in 1957. He obtained the M.Sc degree in Electrical Engineering in 1981 and the Ph.D. degree in Automatic Control in 1987, both from Lund Institute of Technology. Since 2000 he is a Professor at the Department of Automatic Control at Lund Institute of Technology, Lund, Sweden. His main research interests are in the fields of real-time systems, real-time control, and discrete event control.