# Feedback Control Using Computers

David M. Auslander

September 10, 2001

Note: This article is published in the *Encylopedia of Physics*, Academic Press

## Contents

# 1 Feedback Control

An engineered system is created according to a specification of what it should do. Many engineered systems involve the modulation of power in a manner that will meet those specifications. *Control* in the broadest sense refers to the means by which the power is modulated or manipulated. The art of engineering is to design systems that meet their specifications in the most cost effective way possible. What is considered possible in creating a specification is very much a product of whatever technological and methodological base is generally known at the time it is created. Control has traditionally been exerted by humans who adjust the application of power according to their observation of the evolving result. In systems where the specification is relatively loose or the system very slow, the human can exert control on an intermittent basis, directing attention elsewhere in the interim. Examples of this approach abound: a woodworker with a traditional lathe, a cook, heating a home with a wood or coal stove, driving a vehicle (automobile or horse-drawn), innumerable sports activities, etc. Some of these activities involve continuous application of control (more-or-less), such as driving, others are distinctly intermittent such as heating a home with a wood stove.

Prior to the 20th century, neither methodology nor technology existed for reproducing the human-mediated control process with machines, although both imagination (Jules Verne, for example) and a small number of notable examples (for example, the Watt steam engine speed governor) whetted the appetite. It is not a coincidence that virtually every book on automatic control cites the Watt governor as a historic example; there was not much else available. The 20th century saw the development of feedback control methodology and technology to the point that it has become a ubiquitous component in engineered systems. Feedback control is the explicit use of the observe, decide, act cycle without human presence. It is also called automatic control to recognize the replacement of the human controller with a machine of some sort. It was primarily applied to large-scale industrial applications in the early part of the century, power generation using speed control derived from the Watt governor, and process control using newly invented pneumatic technology.

Design methodology was weak in that period with most applications dependent on experience and experimentation to achieve satisfactory performance. Electronics and war fueled the mid century developments of feedback control, largely for military applications. That electronics proved to be a far superior technology for feedback control implementation motivated major developments in design methodology as well.

As computer technology developed sufficiently for its application to feedback control in the last third of the century, however, both methodology and technology for feedback control exploded. Computers were so far superior to previous technologies as the decision-making components that the entire engineering domain of feedback control was liberated from restrictions on the nature of computations that could be carried out. While economics still limit what can be accomplished in any given system design, the uncanny accuracy of Moore's Law — computing power will double every 18 months — motivates continuing widespread activity in design methodologies. It is the purpose of this chapter to highlight some of the critical factors needed for successful application of computers to feedback control. Beyond this article, there is an enormous instructional and research literature that can be used to provide insights into solution of a wide a variety of difficult problems.

## 1.1 Compensating for Ignorance

With perfect knowledge there would be no need for feedback control. It would be possible to know at all times exactly what input would cause the system to have the desired output. However, perfect knowledge is very expensive (infinitely expensive!) The beauty of feedback control is that, at modest cost, it can coax maximal performance from imperfect hardware. In other words, compensating for ignorance (or imperfection) through the observe, decide, act feedback cycle is generally much less expensive than attempting a better approximation at perfection in order to reach the same level of performance.

Imagine, for example, controlling the speed of a rotating disk driven by an electric motor. This situation is typical of large numbers of applications requiring speed control. In the absence of feedback successful control of this system would require a combination of knowledge of exactly how the device operates, knowledge of the load being placed on the device at all times, and knowledge of disturbances from the environment that might cause the speed to vary. "Exact" in this context is defined as sufficient to allow operation within the specified tolerance for variation of the speed from its specifications. This would, for example mandate knowledge of bearing operation and associated friction with changes in speed (the motor could be required to operate at various speeds or with a continuously varying speed) and changes in friction as a function of temperature and age of the machine. There would also have to be a means for knowing the precise characteristics of the load, which could change with device position (such as a robot) or as materials are picked up a dropped off. With the development of feedback control, control systems such as this that depend on prior knowledge and calibration have been called "open loop" control systems, with the term "closed loop" defined as synonymous with feedback control. Where the tolerance for speed control is very loose (as in a room-cooling fan, for example) this method might work satisfactorily and economically. As the tolerances are tightened (in a precision grinding application, for example), achieving the desired control accuracy would get very expensive with open loop control.

Why is feedback control for speed control a more economical solution? Primarily because achieving high accuracy with an open loop system puts very stringent demands on the manufacture of the components. They have to be made so that their performance is impervious to most environmental changes such as temperature, humidity, as well as age of the machine. It is usually almost impossible to predict such changes as must be done if they are larger than the control tolerance so that design-manufacturing techniques to minimize them are necessary — and expensive. In a feedback control system these types of stringent requirements apply only to the measuring instrument, in this example a speed measuring instrument. Other components, the motor, the mechanics delivering power to the target system, etc., can be manufactured to more relaxed tolerances.

Building the measuring instrument to tighter tolerances is much easier than building the power-delivery part of the system to the same tight tolerances. The primary reason is that measuring instruments do not have to deliver any significant amount of power — just enough to provide a readable signal to an amplifier. They can thus be much smaller and because they carry no load, are not subject to one of the main causes of speed change, load variation.

Thus, the use of feedback control is economically motivated: the main power delivery portion of the system can be built with short-term reproducibility that meets the performance specification; only the smaller, less expensive measuring instrument need have long-term accuracy meeting those specifications.

## 1.2 Why Feedback is Hard: Dynamics

Feedback control has its limitations. There is a fundamental limitation imposed by the instrument: the overall performance of the control system cannot be any better than that of the measuring instrument. There is another fundamental limit imposed by physical realizability: because no physical system can change state instantly, there is a limit to how fast a control system can respond to changes in either its command or to changes in disturbances. The "dynamic" behavior of a physical system describes how it changes in response to a change in its environment, purposeful change as when motor input power is changed, or unexpected change as when a "downdraft" causes sudden loss of altitude in a passenger airplane, much to the discomfort of the passengers. This is not to say that a control system with constant command and unchanging disturbances cannot maintain performance within specification indefinitely: that is "static" behavior because it does not depend on internal rates-of-change to meet the specification.

The behavior of a dynamic system depends on the history of what has been done to it. It is not enough to know what its current inputs are — even if the airplane has moved past the downdraft, it will take some time to bring it back to its previous flight state. This is the crux of the feedback control design problem: react too energetically to a deviation from the desired output and the future consequences may be more than was bargained for. React too timidly and it will take altogether too long for the system to get to where it should be (maybe never). The essence of the mathematical side of control engineering is to devise methods for doing the feedback process just right. The practical side of control engineering is in understanding when and how to apply the mathematics and in designing the environment in which these mathematical methods are embedded, software and hardware, so as to build the most effective possible control systems.

The conclusion is that *if not for cost considerations* it would always be better to use open-loop control than feedback to meet a given performance specification. Open-loop control can never cause an otherwise stable system to become unstable. However, engineering cannot be separated from economics. Feedback is so much more cost effective that its use is essential in many practical situations.

## 1.3 Feedback Control System Structure

The classical representation of a control system, the "block" diagram, focuses on the signals that are used specifically by the control system. The internal structure of the power-delivery system is not shown.

Figure 1 shows a block diagram for the speed control system. As is common in these diagrams, only the signals actually used by the control system are shown. Environmental disturbances are sometimes shown on block diagrams, but showing them accurately is complicated because their power relationship to the system is often different than that of the control input. The block labeled "power delivery system" usually includes the power amplifier ("drive" for a motor), the actuator, and the object to which power is being delivered.

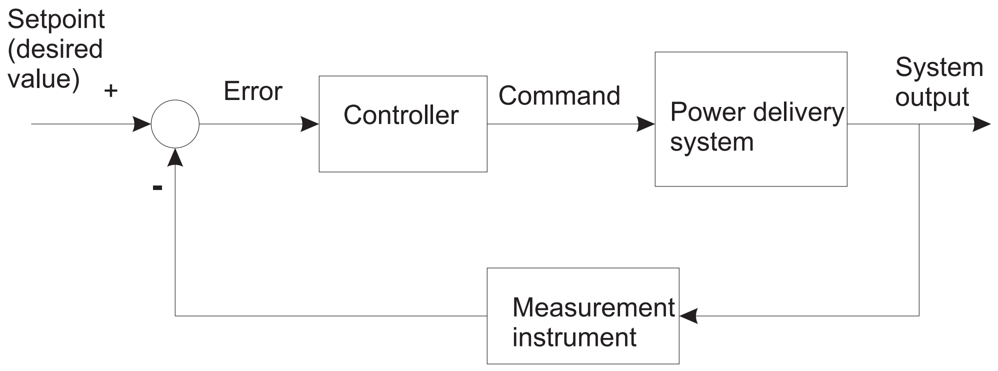

Figure 1: Block Diagram of a Speed Control System

Each of the lines is a directed signal, with the direction shown by the arrow, and represents a single, scalar quantity. In the physical system, these lines represent wires between amplified points, with the arrow going from the output of one amplifier to the input of another. Double or extra wide lines are sometimes used to show vector quantities. These are still directed, however, with each of the vector quantities usually representing a single amplified signal. In cases where one or more of the signals are delivered via a computer network, there is no direct connection to any single wire in the physical system since the network carries different signals at different times (see the next section for more detail on amplification).

## 1.4 Amplification and Isolation

Amplification, and the isolation afforded by amplification, is one of the major enabling technologies for feedback control (the other big one is digital computation). Generally speaking, amplification is the control (or modulation) of a large power flow with a smaller one. the amplifier input (often also called the command) is the low power signal. It normally modulates the flow of power from a source to target device.

There are two unique domains of amplification: power amplification and signal amplification. Of these, power amplification has been known for much longer and was essential for any kind of control system, open loop or simple feedback control systems. The Watt governor, for example used a steam valve to modulate the flow of steam to an engine and thus control its speed. Prior to Watt, the same steam valve was used to manually control the engine speed. A steam valve is a form of power amplifier. The power source is the boiler, which contains high pressure steam. Turning the valve stem operates some form of variable blockage that allows more or less steam to flow to the engine. The level of power needed to turn the valve stem is much lower than the power available in the flowing steam. Watt cleverly connected an instrument measuring the engine's speed to the speed valve so that as the speed increased the flow of steam was reduced, bringing the engine back closer to its desired speed. The instrument was a flyball, a vertical shaft that rotated with the main engine shaft. On it were a pair of weights hung from the top of the vertical shaft and connected to it so that they rotated with the shaft. As the shaft speeded up the balls would tend to fly outward. This outward motion was connected via a linkage to the steam valve so as to cause correct motion of the steam valve.

The Watt governor utilized the power amplifier, the steam valve, very well. But it suffered from lack of a signal amplifier. The power needed to operate the steam valve was substantial, even though much less than the power in the steam flow being modulated. That power was supplied from the main engine shaft, through the flyball measuring instrument. The flyball and its associated linkages thus played a triple role: measurement, computation, and power delivery to the steam valve. As a result, none of these functions could be optimized. All must be compromised by the need to meet all three requirements simultaneously.

## 2 Computers for Control

The vacuum tube, and the applications of electronics to signal and power amplification, opened the door for feedback control in which each of the system elements could be isolated and optimized. On the decision-making side, electronics could be used to build a wide variety of linear, filter-like functions. These could be combined to yield a huge variety of functions. As compared to the Watt governor, for example, the computational flexibility was vastly improved. The problem of "droop," the inability of the control system to drive the error to within its tolerance, even if neither the setpoint nor the disturbances are changing, is difficult or impossible to solve with the Watt governor. Using an isolated, electronic computing element, however, makes that problem relatively easy to solve (see "integral mode" control, below).

Flexible as it is, electronics as a computing media is severely limited. It is very effective at implementing linear operations (those representable by linear, ordinary differential equations) but it does not do well at nonlinear operations. Some nonlinear operations can be implemented by creative use of diodes — limits, function generators, peak holders, etc. — but these are difficult to work with and limited in scope.

Digital computers, on the other hand, have nearly unlimited computing complexity. If a way could be found to use them as the computing elements in feedback control systems that functionality could be optimized still further.

### 2.1 Computers and the Physical World

Using a computer as the computational element for control systems is a real challenge. First, the internal representation of numerical information is entirely abstract. That is, an internal coding is used that is arbitrary with respect to any physical realization of that number. Analog electronics used for control computation provided a relatively easy path in this regard because physical quantities are represented as voltages (or currents) in the computing circuit giving an easy analog to the external physical quantities they represent and easy conversion for input to and output from the computing circuit. There is no such easy path when a computer is used as the computing element in a feedback control system. Complex conversions must be done to connect the physical world to the computer and *vice-versa.*

To make matters worse, computers are also sampling devices. This is also fundamental to the nature of the beast. Computer architecture is based on the idea of a powerful processing unit that is capable of executing instructions it gets from a memory. Each instruction is executed in turn, with the central processing unit (CPU) devoting its full attention to that instruction. The result a is one-thing-at-a-time device so that control of a system happens by a series of discrete operations: observation, computation, actuation, with each of these operations requiring the execution of thousands of instructions. Thus control action is based on a momentary sample — the control object is ignored until the next time a measurement is made. This behavior is in contrast to standard electronic control systems in which all activities are carried out simultaneously. Modern computers contain some degree of parallelism, but not enough to change the sampling mode of behavior.

## 2.2 Signal Conversion

Measuring instruments designed for use in systems for which the human operator is providing the feedback control present their output information in human readable form, most commonly visual. Instruments designed for use in automatic feedback control systems (or in data logging systems) must present their outputs in forms that the control system can read. Because computers use electricity as their primary medium of operation, some form of electrical ouput is needed for measurements. The most common form of electrical output is an uncoded voltage, in which the instantaneous value of the voltage has a one-to-one functional relationship with the quantity being measured. Ideally, that relationship is linear, but linearity is not always achieved.

To be used in the computer, the voltage must be converted into a set of digital (binary) signals representing a binary integer (see Number Representations, Precision and Scaling, Section 2.5). This set of signals can then easily be copied into the computer's memory where it is usable as a "variable" by the control program. Generally, the value of the voltage scales linearly to the value of the integer. The device that does this is an "analog-to-digital converter (A/D or A-to-D or ADC)." While the details of A/D converter operation are not critical to this discussion, the general characteristics of the device are that it is fairly complex, consisting of digital logic and precision analog components, requires modest numbers of microseconds to complete a single conversion (which is slow on the time scale of a modern desktop computer), and typically have a conversion precision ranging from 8 to 16 bits (yielding resolutions from 1:256 to 1:65,536).

## 2.3 Discrete Control

Both numerical values and time are discrete in computer-based control. Numbers have binary digital signals as their basis and are thus normally finite precision. Time is discrete because the central processing unit can only handle one operation at a time. Both of these have severe consequences for feedback control. Finite precision arithmetic causes round-off error, which appears to the control system as noise. This is most severe in low-cost systems where economics forces the use of integer or fixed-point arithmetic.

Time discretization, however, is responsible for the most mischief. Because of the sampling nature of computer control, the control object is open loop most of the time. Disturbances that occur just after the computer has taken a sample are not detected until the next sampling instant. Control quality is improved as the sampling rate is increased, but faster sampling takes more computer power and is thus more expensive.

## 2.4 Computational Scale, Micro to Maxi

Personal computers have become a ubiquitous part of professional life — there is one on every desk to say nothing of the large numbers in homes. These computers are probably most people's major conscious image of a computer. Because applications tend to take advantage of the most computing power available, and because the cost of personal computers has dropped so much, the range of computing power is fairly narrow — there are not too many personal computers over five years old that are still in regular service.

The situation is very different with computers used for control of physical systems. These computers are normally embedded within a product and the economics are driven by the overall economics of the product. Thus, consumer products might impose limits of only $1 to $2 for the computer, while a machine used in a manufacturing process could allow for several thousand dollars or more for the control computer. Although all of these are computers and share the same general architecture, they impose vastly different constraints in terms of the types and amounts of computation that can be done within the time constraints of satisfactory control.

## 2.5 Number Representations, Precision and Scaling

How control computers do arithmetic has a significant effect on the control performance and on the productivity of the programmer writing the control software. In common applications such as spread sheets, simulation programs, stress analysis, etc., the users are isolated from the internal numerical representations because the programmers have made sure proper selections have been made. Programmers using common programming languages, C, C++, Java, Basic, etc., are given a wide choice of data types but modern computers are capable of handling all of them with minimal penalty. Given the scale of computers used for control, however, some choices may be precluded because the performance penalties in these cases are significant.

The major considerations is selecting internal computer data representations to use in a control calculation are: round-off error, dynamic range and computing efficiency. The easiest number representation to use for control calculations is scientific notation, variously called "real" or "floating point" in different computing environments. Computational scientific notation represents numbers by a mantissa and an exponent. The mantissa is a fixed-precision, fixed point number. It is multiplied by a base to the 'exponent' power to get the value of the floating point number. The mantissa is always normalized, meaning that its highest-order digit is always non-zero. In common notation, a scientific notation number is written as 1.2345*10$\hat{4}$. The part before the asterisk is the mantissa; the exponent is the part after the ˆ. The major differences between the common paper notation and computer notations are: 1) very few people pay much attention to the number of digits in the mantissa whereas that is critical in computer representations, and 2) in the computer the numbers are in base-2 (binary) format so the exponent raises 2 to a power. The major advantage to floating point numbers is that because the mantissa is always normalized, the calculational precision is independent of the magnitude of the number (except for extreme numbers). The disadvantage is that processing of floating point numbers is very complex and thus either expensive or slow (the usual trade-off with computers). Standard computer floating point data types give from 5 to 15 decimal digit precision in the mantissa.

Low cost applications cannot afford floating point (given Moore's Law, however, floating point can be applied to more applications every year). They are restricted to some form of integer arithmetic. The simplest form is where the numbers represent whole integers, usually signed. In this case, the precision of the calculation depends strongly on the magnitudes of the numbers involved. "Precision" is defined as the ratio of the number to the difference between the number and its nearest neighbor. Take the number 1,000, for example. Its nearest neighbors are 1,001 and 999. In either case, the associated precision is 1,000:1. On the other hand, take the number 2. Its precision is 2:1. The lower the precision, the larger the roundoff error in calculations. Roundoff errors appear as noise in computer control systems, so can severely affect the quality of control. Maximizing the overall precision of calculations requires that numbers be scaled so they are large enough to have reasonable precision but small enough to avoid overflow problems (which have catastrophic results). This adds programming complexity to the job of creating control software.

Fixed point numbers are somewhat more complicated than integers in that they allow for much more effective handling of quantities between zero and one. However, they suffer from the same precision problems as integers.

The summary: use floating point if at all possible! Extra hardware cost may pay off in quicker time-to-market and a more reliable product.

## 2.6 Control System Software: Real Time

For successful control of a physical system, the events noted on the timeline above (measure, compute, actuate) must happen at the correct times (within a specified tolerance). “Normal” software (word processors, numerical computation, spreadsheets, etc.) operate in ASAP mode. That is, the user would like the result as quickly as possible. Other than impatience if the operation takes too long, specific times are not a factor in standard software.

Software which must deliver results at specified times, or at specified time intervals after some external event, is designated ”real time“ software. The added time dimension, plus the fact that a number of control activities must overlap, makes control system software substantially more of a challenge to deal with than standard software of approximately equal complexity.

Real time software to implement feedback control has three major components: functions that service the instruments and actuators, a “main” section that implements the feedback algorithms, and the operator interface. Typically, the priorities of these components follow the same order — the instrument-actuator service is highest priority, the feedback next, and the operator interface is lowest. “Priority” for real time software refers primarily to the importance associated with a software module executing within a specified time of the event that triggers the module. That time interval is also called the “latency” specification for a software module. For control software, the general rule is that all software components must be activated within their latency specification, so that priority is based almost entirely on the latency specification. Although instruments and actuators fall into the highest priority class, there are very wide variations in latency requirements among different types. In most cases, the feedback control loops have similar latency requirements, so they can all be handled as a group. See Sections 3.4 and 3.3 for details on implementation of the feedback algorithm section. Finally, the operator interface must respond on human time scales, so usually has the loosest latency specifications. It is also common to implement the operator interface on a separate computer to simplify the real time structure. Operator interface concerns are very important to control system success. However, the subject is too vast to be treated here beyond this passing reference.

The mechanism used to realize a software system having components with substantially different latency requirements is the “interrupt” facility that is part of all computer hardware. Interrupts are triggered by events external to the computer and cause it to switch its attention from one software section (or “thread”) to another. This switch takes place in a very short amount of time (normally less than a few microseconds). In this way, software modules with very short latency specifications can preempt activity by other software modules and respond very quickly to external events. The most common external event is a signal from a clock; others include switch closures, change in state of an electrical signal, etc. The computer’s operating system also uses the interrupt system to service standard peripherals such as the disk drive, keyboard and display.

The general principle in designing software for feedback control is to minimize the use of interrupts. Programs containing interrupts can be difficult to debug, harder to port, and require more maintenance. Assuming a computer doing nothing but feedback control (perhaps with a number of feedback loops active), all of the feedback loops could operate in non-interrupt mode as could any of the instruments and actuators that do not need interrupt servicing. Examples of instruments that do not need interrupts are analog instruments where the signal comes into the computer via an analog-to-digital converter, instruments with their own digital interface but that have substantial internal buffering, and on-off instruments. Likewise, on the actuator side, those that run from an analog command to a power amplifier, for example, do not require any interrupt servicing. On the other hand, a low-speed pulse width modulation (PWM) signal, which would be suitable for a large heater, would require interrupt servicing to maintain the accuracy of the PWM command.

Minimizing the use of interrupts requires only one major software design principle: keep all code non-blocking. All decisions based on external events either cause an action or not, but never wait. Examples of this style of programming are given in Sections 3.4 and 3.3.

### 2.7 Programmable Logic Controllers (PLC)

PLCs are a class of computer designed specifically for industrial use and, traditionally, programmed using ladder logic. PLCs were originally designed as a replacement technology for relay logic, a means of implementing Boolean and sequential logic using sets of contacts and solendoids (”relays”). As such, the original device was suitable for solving problems suitable for digital logic representation, with instruments and actuators using logic (on/off) for input and output. As its use expanded, facilities were added so that more general computation (using regular numerics rather than just logic values) could be mixed with the ladder programs. Eventually, a set of languages was established as a standard for these devices, ISO/IEC 1131 (International Organization for Standardization (www.iso.ch), International Electrotechnical Commission (www.iec.ch)). The language set includes ladder programming and an algorithmic language, as well as other programming means.

Implementing feedback control using sequential logic is beyond the scope of this article, so the reader is referred to other texts on PLC programming. Using the algorithmic programming facility of IEC 1131 is the same as implementing feedback control with any other programming language, so all of the material here is relevant.

## 3 Simulation

If the physical control object or a reasonable physical model or prototype of it is not available (for example, not yet built) or if experimentation with it would be either too expensive, too time-consuming, or too dangerous, some form of control algorithm design and tuning based on mathematical formulations can be done. There are two main ways to start this process: simulation or linear systems. The issue here is not excluding the use of one technique or another, but of having a general notion of how to start. By the time the process is over, if the problem is a difficult one, every relevant technique in the book will be thrown at it!

In the past (*i.e.*, when computing was very expensive) there was no question at all: the linear systems approach could produce results for many practical problems whereas the lack of generality of a simulation combined with the expense of getting results (which required a lot of computation) rendered it of little value except in the extreme. Computational cost, however, is no longer a serious limitation and the cost per unit of computation continues to decline. With low-cost computation available, the advantages of simulation can be considered: much more generality in the nature of mathematical models that can be used and few limits on the types of performance measures that can be applied.

Most control classes are taught from the assumption that the linear systems approach is the better way to start a control system design problem. However, the computing power now available, along with effective mathematical software tools, makes the simulation approach extremely attractive. Thus, simulation is introduced first here as the preferred means for an initial approach to a control system design or analysis problem.

## 3.1 Models of the Control Object

What is a simulation? It is a numerical result obtained by solving a set of equations that purport to describe the desired aspects of the system-of-interest's behavior. The numerical result can be viewed in some appropriate form by a human analyst, who would then make some decision and, perhaps, do another simulation. It could also be used in further numerical procedures that analyze the result and make decisions that could also result in the execution of another simulation. As computing power has advanced to the point of being able to run multiple simulations in an automated process of analysis or optimization simulation has risen in its importance as a primary design and analysis tool.

The "purports to" modifier in the above paragraph is a very important part of the mathematically-based process — simulation, linear theory, or any other approach. While this article mainly addresses issues in producing and using the simulation results, a separate and equally important part of the job is the verification that the model (*i.e.*, the set of equations) actually does describe the system-of-interest sufficiently well.

Most control objects to which computer control will be applied can be described with differential equations. Ordinary differential equations (ODEs) suffice in many cases; the discussion here will be limited to those cases. There are, however, important control problems where the control object is more properly described with partial differential equations.

$$\begin{array}{l} \frac{dx_1}{dt} = f_1(x_1, x_2, ..., x_n, t) \\ \frac{dx_2}{dt} = f_2(x_1, x_2, ..., x_n, t) \\ ... \end{array} \tag{1}$$

## 3.2 Numerical Analysis

Producing a simulation result requires a numerical solution to the set of ODEs representing the system. Control problems are almost universally initial condition problems, meaning that the values of the state variables are known at time $t = 0$. Since the system equations, equation 1, are generally nonlinear, only aproximate solutions can be produced. Using the fact that initial conditions are always known, the method of solution is to define a small, but finite, time increment, $\Delta t$, and then to approximate the values of the derivatives of all of the state variables during that time interval. There are many ways to do that, each having different properties in terms of the accuracy, stability and efficiency of the solution. The simplest approximation is that of Euler, in which the derivatives are computed at the beginning of the time interval, and assumed to be constant for the entire interval.

$$\begin{array}{l} x_1(t + \Delta t) = f_1(x_1(t), x_2(t), ..., t)\Delta t + x_1(t) \\ x_2(t + \Delta t) = f_2(x_1(t), x_2(t), ..., t)\Delta t + x_2(t) \\ ... \end{array} \tag{2}$$

This format is intuitive and easy to understand as it basically expands the finite difference definition of the derivative. Since everything is known at zero-time, the right-hand side is completely determined and the solution is easy to start. However, except in cases where neither computing time nor stability of the numerical solution are very important, the Euler solution method is rarely used.

Of the large number of possible methods, the most popular are the Runge-Kutta solutions. These are based on better approximations to the right-hand side obtained by expanding a Taylor series around the current point. Runge-Kutte solutions of various order are available; the fourth order version is probably the most popular giving a good compromise between complexity of coding and efficiency of the solution. Most popular implementations also use an adaptive step size. In addition to the approximation to the one step ahead solution an estimate of the error is also produced. The step size is then adjusted until the error estimate falls below specified bounds. This is particularly important for nonlinear problems where the appropriate step size changes as the solution proceeds.

The Euler and Runge-Kutta methods and others like them are "explicit" ODE solvers. They are simple to implement but have the property that the maximum allowable step size is determined by the fastest behavioral mode of the system being simulated *regardless of whether the details of the fast modes are important or not.* For systems containing modes with widely separated characteristic times, this can lead to very inefficient solutions. "Implicit" methods handle these "stiff" equation sets much more efficiently if the fast details are not important. However, they are much more complex to code and take much more computing time per step.

## 3.3 Simulation of Hybrid Systems

Digital control systems are hybrids. The control object exists continuously in time (thus the differential equation representation) while the controller is discrete and is only active for brief periods of time. The simulation software can recognize this situation by embedding a mini-event-manager into the main simulation loop.

The first part of the simulation program sets parameter values, initializes, state variables, etc. The working part of the program is the simulation loop. The main simulation loop has two parts: the event manager which controls execution of the discrete functions (those that the control computer would be doing) and the ODE solver for the continuous part of the system (normally the control object).

The event manager generally handles execution of one or more control (feedback) loops, data logging, external events that might affect operation, and when to terminate the simulation. On each pass through the event manager code associated with all relevant events is executed and each of those events sets the time at which it will again need attention. The ODE simulation section then solves the system equations for the control object up to the time of the next event. Any ODE algorithm can be used; the sample software uses both fixed step size Euler solvers coded inline with the event loop and Matlab solvers that use a separate file for the differential equation righthand sides.

Any of the contol simulation programs can serve as a template for this approach to control system simulation. The *BasicTank.m-BasicPtank.m* set of files simulates the control of liquid level in a single tank using the Matlab ODE solvers (which solver is used can be changed by just changing the function call name). *MassMotion.m*, on the other hand, uses an inline Euler solver to show the effect of different setpoint profiles on position control of an inertial load. The inline Euler solution is not as efficient or stable numerically but allows for the simulation to be completely contained within one file avoiding the necessity of transfer of parameters between the main file and the ode-file. Space limitations prevent inclusion of any of the actual code here, but examination of the files should display the simulation structure clearly.

## 3.4 Control Software Implementation

As noted in Section 2.6, the structure of the feedback control portion of the real time program was deferred until after the simulation structure was discussed. The biggest difference between the simulation software and the actual control software is that no simulation section is needed for the actual implementation — "nature" solves the ODE! The other significant difference is that time being real must be determined by an external clock rather than through an internal calculation. The program determines the "current" time by a call to a system utility that reads the computer's clock (the clock is external to the central part of the computer, the CPU, but is normally part of the hardware set that makes up a functioning computer.

Any number of control loops and other events can be handled with this structure. The performance restriction is, as is the case with all real time software, making sure that all control loops meet their latency restrictions. For feedback control loops, errors in when they run show up as noise in the control calculation. Large errors can affect stability and performance of the control loop. Since several control loops could conceivably be ready to run at the same time, the worst-case latency error is the sum of the execution times of all of the control loops. This can be calculated (based on computer performance figures) or measured experimentally and used to specify the computer speed needed to meet specifications for a given control configuration.

# 4 Basic Feedback Control

The dynamics of the control object is the largest source of difficulty in implementing control so most of the techniques devised to design controllers deal with dynamics and controller "tuning." Tuning refers to the process of setting parameter values so as to achieve satisfactory performance. In the digital control world, a "controller" is a section of software that implements an algorithm that takes the measurement of the control object's output as its input and produces as its output the actuation signal. In subsequent discussions, the term "control algorithm" will be used rather than "controller."

## 4.1 Small Signal Behavior: Tracking, Disturbances

The most fundamental behavior of a control system is exhibited by its response to small changes in either the setpoint (desired value) or in a disturbance. The behaviors that are important to observe are, for the short-term characteristics — is the response monotonic or does it oscillate, does the oscillation persist or die away — and in the long term how much change there is in the output for a given change in either the setpoint or disturbance (ideally, the long-term change should be equal to the setpoint change and be unaffected by disturbances).

## 4.2 Equilibrium, Stability and State

These three terms from system theory are important in understanding and characterizing the behavior of control systems. The "state" of a system is characterized by a set of variables (not surprisingly called state variables) such that if at any instant the values of those variables are known and the future inputs to the system are known then the complete future behavior behavior of the system is determined. Classes of systems to which this characterization can be applied are called "state-determined systems." The state variables provide information about the independent energy storage modes in the system. The particular set chosen is not unique, since any linearly independent sum of state variables is an equally valid state variable. The necessary number of state variables is unique.

A system is at "equilibrium" if for a constant input none of its states is changing. Thus, an equilibrium point in the state space is a point at which the rates-of-change for all of the state variables are zero (the state-space is the space for which each state variable is an axis). For example, a tank with liquid coming into it from above and liquid draining from the bottom can be characterized by a single state variable – liquid level or volume of liquid in the tank would be the most common choices. It is at an equilibrium point in its (one-dimensional) state space when the inflow is exactly balanced by the outflow.

"Stability" refers to what happens when an equilibrium is perturbed. A stable system will return to its equilibrium. In the tank example, if a bucket of water is suddenly thrown into the tank, the water level will rise above its equilibrium value. After a while, however, the level will go back to the equilibrium. Such a system is stable, in this case asymptotically stable because it returns to the equilibrium exponentially. An inverted pendulum, on the other hand, which is carefully balanced in its upright position will never return to the equilibrium position if it is given a little push.

An understanding of stability is of critical importance in control system design because the addition of a controller to a system which is otherwise stable can cause unstable behavior. A controller can also be used to stabilize an otherwise unstable system such as an inverted pendulum.

## 4.3 On/Off Control

By far the easiest control algorithm is on/off — if the output value of the control object is above the setpoint turn the actuation to its minimum value (which is often off) otherwise turn it full on. This scheme also matches the common situation is which the actuation device is only capable of running in these modes. On/off actuators are usually considerably less expensive than those with a full range of operating values. Heating and cooling equipment, for example, often works this way. Figure 2 shows the response of a simulated single tank, liquid level control system using on/off control. The discrete nature of the digital control shows up in the graph of inlet flow (actuation) which holds the same value from one sampling instant to the next.

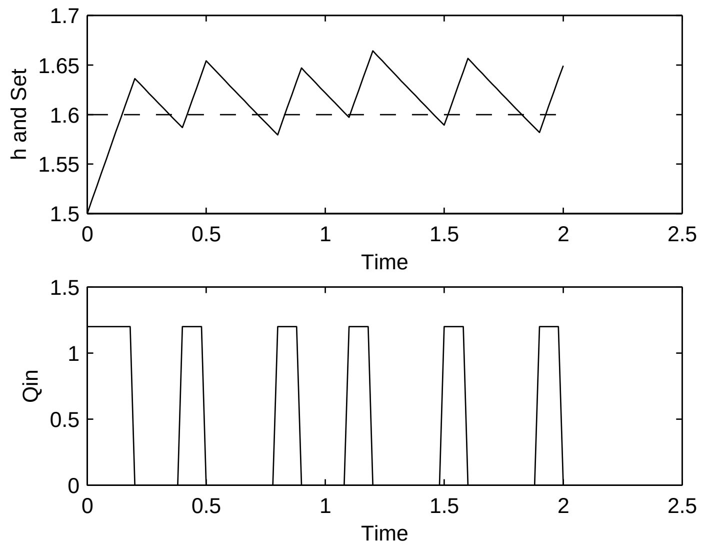

Figure 2: On/Off Control, Simple Tank

The result is mixed. The output (liquid level) stays near the setpoint, but keeps oscillating as the actuator goes from full on to off. Thus, this kind of control is satisfactory where the tolerance on the output is fairly weak (as it is, for example, in a home heating and/or cooling system which uses this method).

## 4.4 Proportional Control

The simplest control algorithm beyond on/off, and one that is intuitively appealing, is to make the controller output proportional to the error,

$$m = k_p \varepsilon \tag{3}$$

where, $\epsilon = r - y$ is the error, $r$ is the reference (setpoint) value, $y$ is the output of control object and $m$ is the controller command (actuation) output. To be a bit more precise, the control object output is not directly known, only its measurement, $\tilde{y}$, is actually available to the controller. This detail is often ignored in preliminary analysis because the instrument is commonly of much better quality than any other parts of the system so its imperfections can be ignored, at least initially.

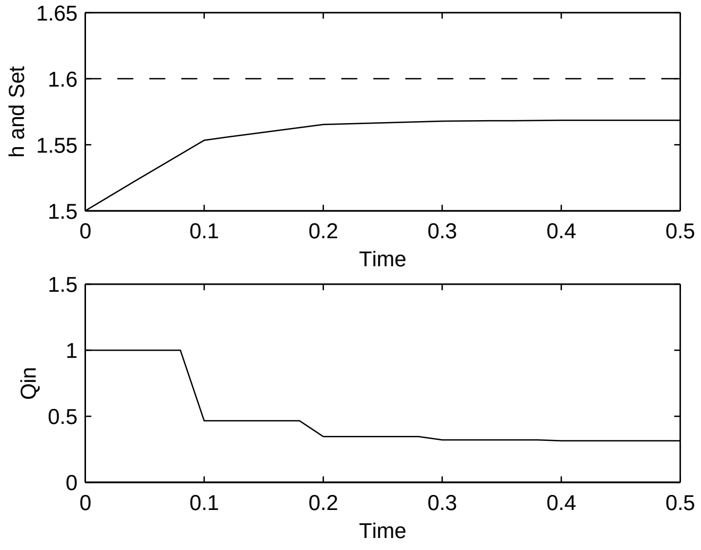

Figure 3: Proportional Control, Simple Tank

Figure 3 shows the response for the same liquid level control system with proportional control. The behavior is much smoother than with on/off control, but, in this case, the output never quite reaches its desired value.

## 4.5 Proportional, Integral, Derivative Control (PID)

The most notable problem with using proportional control of liquid level is that the output never quite reaches the desired level (steady-state offset). This can be corrected by adding an additional mode to the control based on the integral of the error, the "I" mode. In this case the basic controller equation becomes,

$$m = k_p \varepsilon + k_i \int \varepsilon \, dt \tag{4}$$

This classic form of the PI control law cannot actually be realized with a computer controller because of its discrete operation. Instead, a summation as an approximation to the integration in usually used.

Figure 4 shows that the addition of integral action solves the problem of steady state offset, with little change otherwise in the behavior.

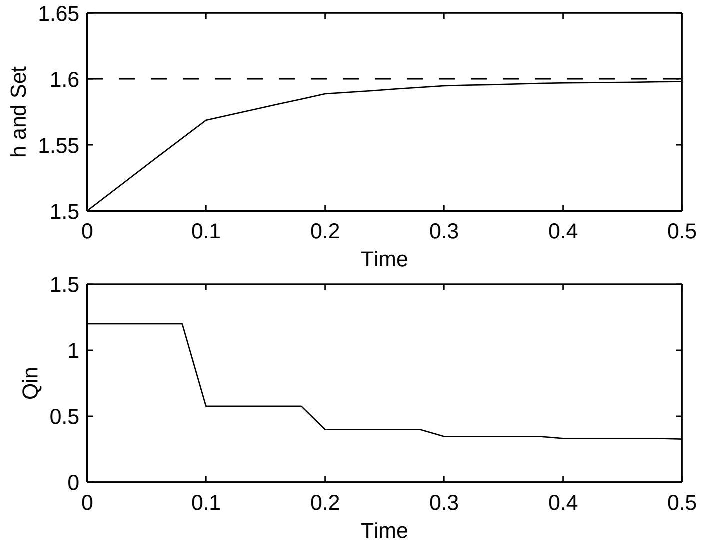

Figure 4: PI Control, Simple Tank

## 4.6 PID Tuning

Finding the best spot in the three-dimensional PID parameter space can be a daunting enterprise. A variety of techniques are available to accomplish this end. In many cases, the field tuning is the preferred environment. With the actual system built and available for experimentation and the control algorithm selected and implemented, the controller parameters are then selected based on experiments performed on the actual system. The problem with a purely experimental method is that even with only three parameters to tune, a full search of the space is rarely practical. A set of guidelines can reduce the dimensionality of the search to a manageable level. A simplified set of rules for hand-tuning a PID control that reduces the three-dimensional search to a series of one-dimensional searches is:

1. With $k_d$ and $k_d$ set to zero, start with a very low P-gain ($k_p$) and increase it slowly until performance starts to deteriorate. Back off to a bit below this point. If the performance is satisfactory, the search is probably finished (additional control modes may improve the performance a bit, but at the expense of added complexity and possibly, robustness.

2. Determine whether the response obtained with the final P-gain suffers more from problems of oscillation and instability or from problems of steady-state error.

3. If stability problems dominated the search for a P-gain, proceed to tune the D-gain; if steady-state error problems predominate, proceed to tune the I-gain.

4. D-gain: Increase $k_d$ slowly and see if the performance improves. If it doesn't, keep the D-gain at zero. If it does, continue until the best performance is obtained. At this point, it may be possible to increase the P-gain and, possibly the I-gain to improve performance still further. Proceed to I-gain tuning if that has not yet been done and the steady-state error is still too large.

5. I-gain: Increase $k_i$ slowly to eliminate the steady-state error. When the best performance has been obtained, try reducing the P-gain a bit, and then retune the I-gain to try for better performance. Proceed to D-gain tuning if that has not been done yet.

This procedure assumes that the controller sampling interval has already been chosen. Although there can be some interaction with the tuning procedure, any choice of sampling interval based on performance will always select for faster sampling. Thus, by the time the controller tuning starts, the sample interval has probably been reduced to as low a value as is practical.

A system consisting of two tanks connected by a pipe allowing flow in either direction has more difficult dynamics than the single tank system used as an example above. Figure 5 shows a hand-tuned result, using the above rules, for small signal behavior of a two tank system. As compared to the single tank system, the use of derivative control is necessary to achieve reasonable stability, and the best (eyeballed) behavior still has some oscillation in it.

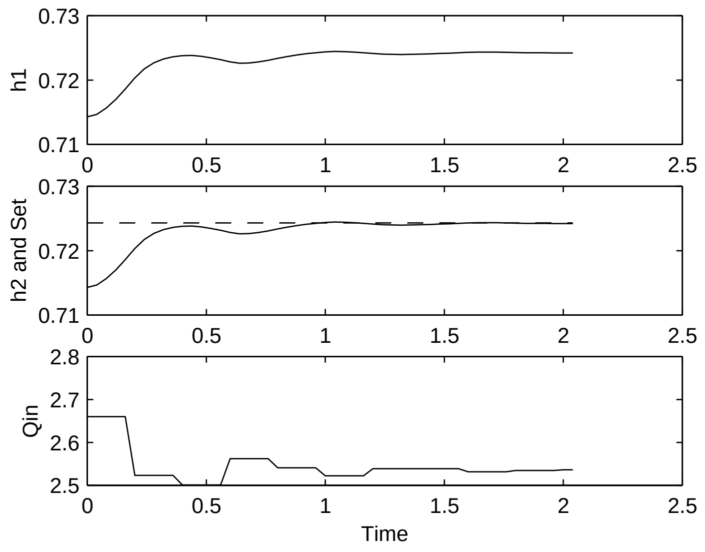

Figure 5: PID Control of Two-Tanks, Hand Tuned

## 4.7 Controller Tuning By Optimization

While simulations are often used to hand-tune controllers, that process can be automated. The computing load to do that is quite high, but is not unreasonable for many problems. The most direct method is to define a scalar performance measure for the control system and then apply an optimization technique to find the controller parameter set that gives the best value of that performance measure.

The simplest performance criterion is the integral (or sum) of the squared error. This is a pure output-based criterion in that it does not weight the input effort at all. There are many variants of this that are in use, but the squared error will do for an example. Performance criteria that also weight the input effort can be used where energy is a consideration (as in a system that carries its own fuel) or where smoothness is important as well as how quickly the error is reduced. Although the definition of a quantitative performance index seems to add a degree of objectivity beyond the "eyeball" method so often used in hand tuning, the procedure can actually be more arbitrary than one might imagine. Significantly different results can be obtained for different performance criteria, all of which seem perfectly reasonable, different operating conditions (setpoint change, disturbance), even different length of simulation run.

The hand-tuned control system of Figure 5 was optimized based on an error-squared performance criterion. The result is shown in Figure 6. The results are very similar. In fact, the performance measure for the hand-tuned case is J=1.19; the optimizer was only able to reduce it to 1.09 (the hand tuning was done by eye without reference to any quantitative index). The optimizer used was the *fminsearch()* function of Matlab. It uses the Nelder-Mead nonlinear, unconstrained simplex method. This method isn't necessarily the most efficient, but it tends to be quite robust. Several starting points, including the hand-tuned gains, were used to give some confidence that there were not any other peak points that were missed (multi-modal space). The optimized result had gains very similar to the hand-tuned gains, except for the integral gain which was nearly doubled.

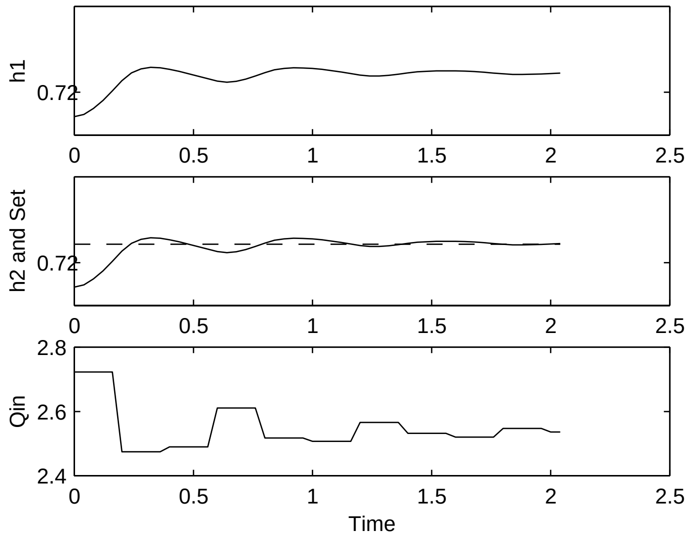

Figure 6: PID Control of Two-Tanks, Optimization Tuned

Because much of the early part of the response necessarily has a large error, a common variant of the error-squared criterion is to multiply error-squared by time. This tends to weight the latter part of the response more heavily than the earlier part. A tuning optimization on this basis was also done, but did not result in a significantly different behavior.

## 4.8 Disturbance Rejection

The optimization in the section above was done for a change in setpoint. The purpose of feedback control is to insulate a system from changes in its environment. Setpoint change is only one possibility of many, although different from most in that it is changed purposely. The term "disturbance rejection" generally refers to how well the feedback control system can maintain the desired setpoint when external conditions change. A case in point for the two-tank system is when the inflow is not what is expected. This could happen, for example, if a calibrated valve is used with no feedback measurement of the actual flow. In that case, a change in properties of the liquid (temperature, density, viscosity, etc.) could cause the flow to be different from the expected calibration. The difference between the actual and expected flow rates appears as an external disturbance.

Figure 7 shows the response to such a disturbance. The controller tuning used is the optimum tuning that was determined above for a change in setpoint.

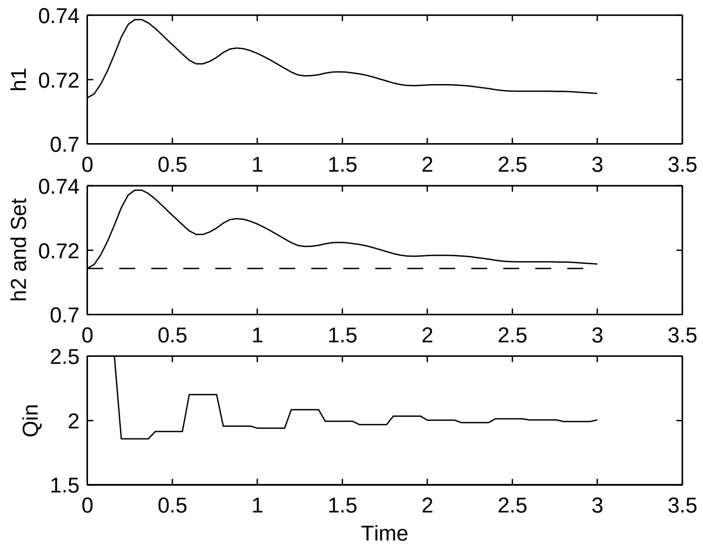

Figure 7: Disturbance Rejection, Two-Tank System, Original Tuning

Running the optimizer again for the disturbance rejection case resulted in a very different gain set. As the results in Figure 8 show, a much higher integral gain was used to bring the tank height back towards the setpoint very quickly, but at the expense of more oscillatory behavior. This case is a good illustration of the dilemma of tuning: there is no one "best" tuning. What is best will depend entirely on what conditions of operation and performance measures are used.

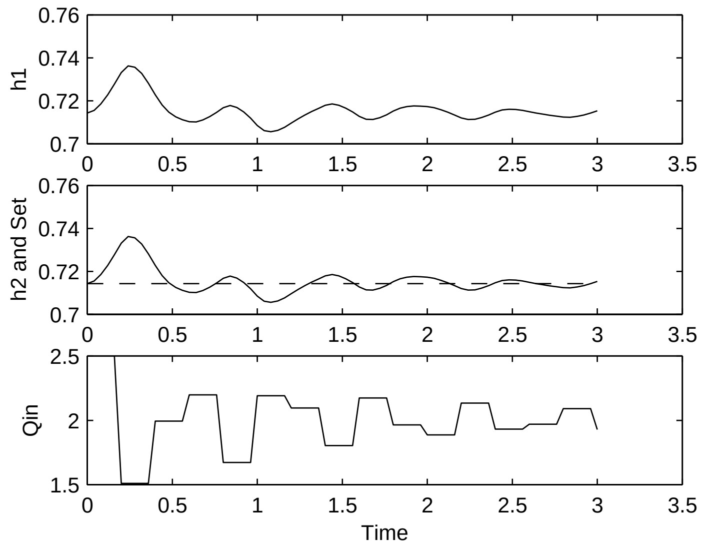

Figure 8: Disturbance Rejection, Two-Tank System, Retuned for Disturbance

## 5 Control Implementation in the Real World

The material presented thus far gives the basics of control, but doesn't solve very many real problems. Probably the most restrictive aspect has been the assumption of small-signal behavior; real world problems often require large changes in operating variables. This section examines some of these issues in an effort to bring the methodology closer to what is needed to build successful control applications.

### 5.1 Rule #1: Keep the Error Small

To some extent, the small signal domain can be forced. As long as the controller error is small, even if the system is in the midst of a large change, many of the advantages of small signal behavior are retained. The controller error comes about from two factors: changes in setpoint, which can be limited, and changes in disturbances, which are exogenous and cannot be controlled. Although the system operator may request a large change in setpoint, the whole change does not have to be passed to the controller at once. Imposing a gradual setpoint change, within the physical limits of the actuator-control object to respond, keeps the error very small during the change. If properly designed, the speed of the change can be almost as fast as other methods of dealing with large setpoint changes, and even as fast or faster in some cases.

### 5.2 Setpoint Profiling

There are two approaches to profiling the setpoint so as to keep the behavior within the physical limitations of the control object and thus keep the error small: 1) use program logic to specify certain functions that the setpoint will follow during changes (*e.g.*, constant slope) or, 2) pass the setpoint through a linear, low-pass filter to "soften" its behavior. The former approach (program logic) will be used here, primarily because it matches the actual physical limitations that actuators have better than the linear filter. Motors tend to have speed or current limits, pumps have flow rate limits, heaters have power limits, etc. These types of limitations generally impose limits on such geometric properties as slopes. A linear filter, on the other hand, because of its linear properties scales its response to have the same shape regardless of the size of the change. If the filter parameters are set so that the maximum slope matches the physical limitation, the reponse will be too sluggish for smaller changes. Also, the program logic based methods have an identifiable "end" to the setpoint change process, which could be useful in coordination with other parts of the system; linear filters usually have asymptotic behavior. An advantage to using linear filters for softening is that the behavior of the entire control system can be analyzed using linear system theory, which cannot be done when program logic is used for softening.

The use of setpoint profiling is illustrated for the case of motion control — moving an inertial load from one point to another (file *MassPosition.m*). For the purposes of this problem, it is assumed that perfect measurements exist for the position and velocity of the mass (note: with a change of names, this simulation would apply equally to rotary positioning of a load with angular inertia). The control structure uses an interior control loop for the velocity control and an exterior loop for the position control. Each of these loops has just a P (proportional) control. This structure is called "cascade" control (see Section 5.8).

Figure 9 shows the small signal behavior of this system with nominal hand tuning of the P gains for the velocity and position loops. These same controller gains are used for all examples in this section. The response shows a small overshoot and smooth behavior. The applied force stays within its limits.

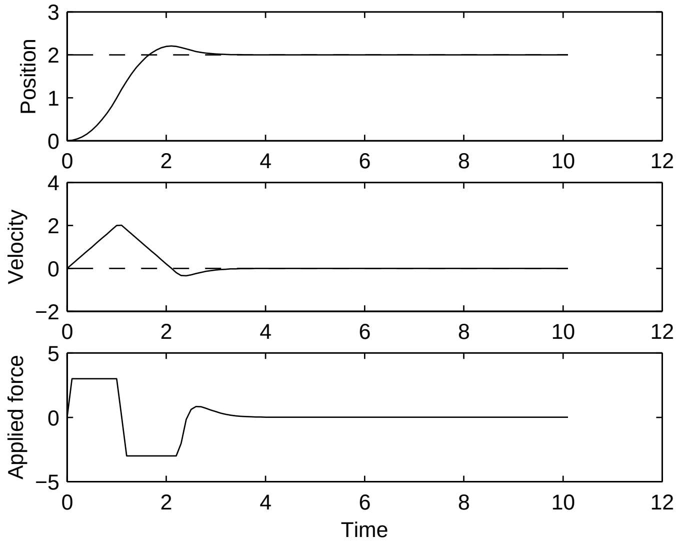

Figure 9: Positioning a Mass: Small Move, No Profile

When the distance to be moved is increased, the force reaches its limit immediately because of the large initial error. Figure 10, using exactly the same controller gains as in Figure 9, shows an unsatisfactory response. The overshoot is large and the position has a slowly converging oscillation about the setpoint. Throughout this whole period, the force is either at its maximum or its minimum, behaving more like an on-off control than a proportioning control.

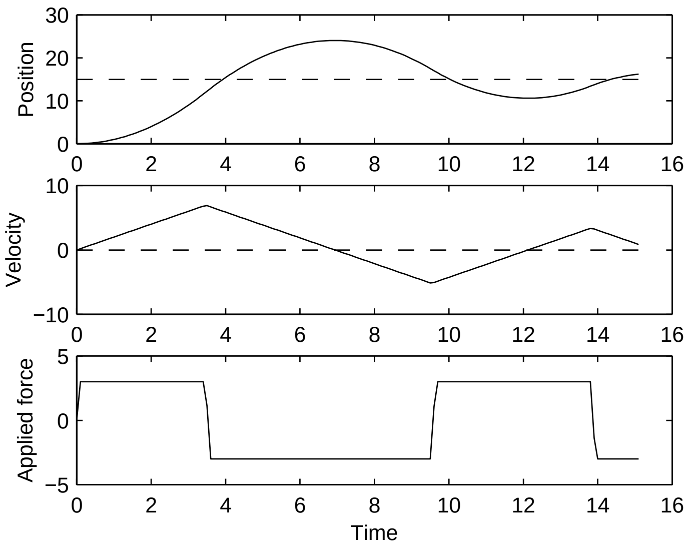

Figure 10: Positioning a Mass: Large Move, No Profile

Figure 11 shows the behavior for a move of the same length but using a trapezoidal setpoint profile with the same controller gains. There is almost no overshoot and the response settles to its final position very quickly. Because of the setpoint profile, the error is always small and the applied force stays within its limits for the entire move. The trapezoidal profile has three sections: constant acceleration, constant velocity, constant deceleration. It is built around the two primary physical limitations of motor-driven systems: maximum acceleration due to current limitations and maximum velocity due to bearing and strength limitations.

An even smoother behavior can be obtained by using an S-shaped acceleration zone. The corners between the acceleration zones and the constant velocity zone in the trapezoidal profile cause sharp changes in the force. If the mechanism includes a structure with any flexibility, for example, these sharp changes could cause undesirable vibration. There are several ways to do this. Figure 12 shows the system behavior with a sinusoidally based profile. Note that it is substantially smoother than the response using the trapezoidal profile, although at the expense of a somewhat longer time to reach the final position. This profile is built using the same maximum acceleration, cruise velocity and controller gains as were used for the trapezoidal profile.

## 5.3 Actuator Saturation

One consequence of not keeping error small is that the controller will ask for actuation output that cannot be attained, causing actuator "saturation," as was seen in the large move done without a setpoint profile in the previous section. There can be adverse effects on the actuator itself as well as issues associated with the performance of the control system if saturation occurs. The adverse effects range from damage to the actuator (for example, demagnetizing a motor) to changes in behavior during saturation such as actuator sticking where a certain amount of time or signal size is required to get it out of the saturation condition.

While an actuator is saturated, the feedback control is, in effect, deactivated. In many cases,

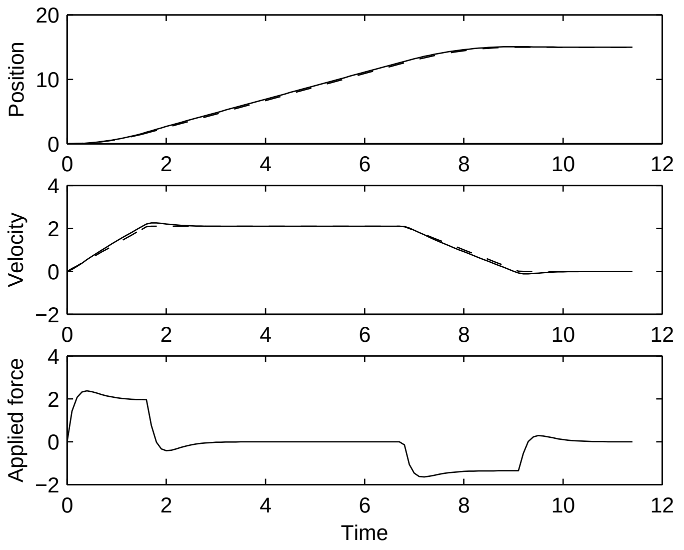

Figure 11: Positioning a Mass: Large Move, Trapezoidal Profile

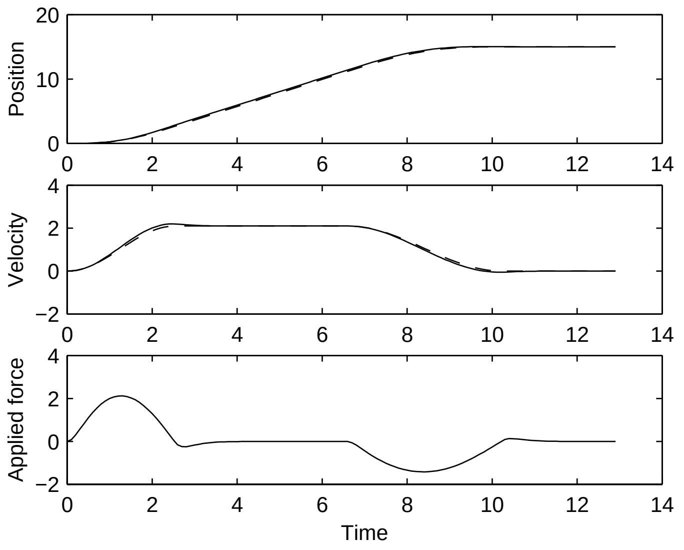

Figure 12: Positioning a Mass: Large Move, Sinusoidal Profile

the only problem caused by this is that the response time is lengthened and the feedback controller becomes active again when the saturation condition ends. If the system is open-loop unstable, however, the feedback control is required for stabilization. While the actuator is in saturation, this stabilizing influence cannot be exerted. In this case, the effect would be catastrophic and it would be very difficult or impossible to regain control.

## 5.4 Integrator Windup

A particular, well-known side effect of actuator saturation is integrator windup, which is a particular problem in digital control because of the large dynamic range of internal number representations. Briefly, what happens is that when the actuator goes into saturation, because of a large error, the integrator continues to accumulate. The integrator may have already become fairly large because the error is large. As long as the acutator stays saturated it cannot respond to larger controller output values. The integrator, however, seeing the large error, keeps adding more and more to its accumulated value at each sample time. In computer control using some form of floating point number, there is no practical limit to how large that number can get.

Sooner or later, the error starts decreasing. However, the integrator term is by that time so large that the controller maintains the actuator in its saturated state. At some point, the error which reach and just cross zero. Until that time, the integrator cannot even begin to decrease. Thus, at the time the error is crossing zero, the controller is in a state at which it is still commanding maximum actuation output and will continue to do that for quite some time. The error thus changes sign, but continues in the same direction at a very high rate-of-change until the integrator actually comes down to a reasonable value. The net result of this: the controlled variable oscillates wildly. A completely unacceptable behavior. This is *not* a function of controller tuning. A controller with excellent small-signal behavior can exhibit integrator windup.

The two tank system of Figure 6 was tuned on the basis of small-signal behavior. Figure 13 shows the same system, but with a large change in setpoint and no setpoint profiling. It doesn't bear any resemblance to the small signal response at all because saturation is a nonlinear effect so the shape of the response is not preserved as the size of the setpoint change. This response is a relatively mild case of integrator windup, but the windup can be seen clearly in the figure by noting that the level of the liquid in the second tank crosses the setpoint, but the controller remains saturated at its maximum value for a long time after that as the integrator slowly winds down.

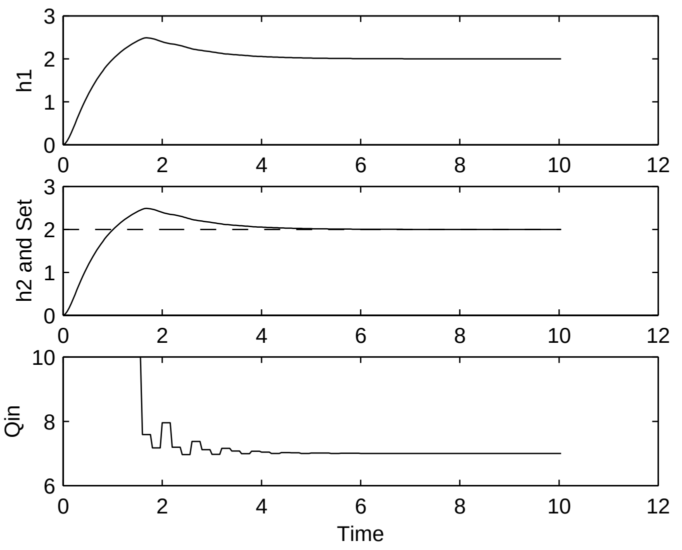

Figure 13: PID Control of Two-Tanks, Large Setpoint Change, No Windup Protection

As demonstrated in Section 5.2, setpoint profiling can be used to retain small signal characteristics by keeping the error small. For the case of saturation, another (and simpler) method is to explicitly prevent the integrator windup but otherwise leave the control algorithm intact. The method used in this case to prevent windup is to freeze the value of the integrator as long as the controller remains in saturation. This takes only a couple of lines of code to implement. The result, even for this mild case of integrator windup, is dramatic. Figure 14, using this windup protection method, settles to the setpoint in approximately half the time taken by the system with no windup protection.

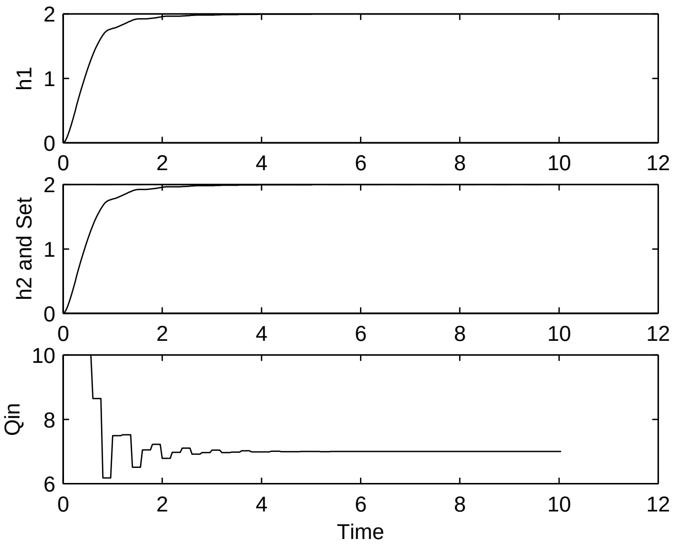

Figure 14: PID Control of Two-Tanks,Large Setpoint Change, With Windup Protection

## 5.5 Noise and Aliasing

The world of digital computing is unique: there is no noise! That is why computer programs can be run over and over again, giving identical results each time. The design concept behind computing, digital, synchronous logic, assures that this state-of-affairs remains true within practical bounds (practical enough to base banking operations on getting correct results!) Control computer interact with the physical world. The interaction with the outside world is asynchronous so breaks one of the basic rules by which noise-free operation is assured. Many of the signals used for control also originate as analog signals. Analog signals characterize the value of the quantity represented with a voltage (or current) value. Unlike a digital signal, whose information content is determined by the number of bits and is fixed, an analog signal's information content is determined by the magnitude of the noise mixed in with the signal. *All* analog signals contain noise.

A problem unique to digital control arises under certain circumstances as an interaction between the noise present in an analog signal and the control computer's sampling process. For illustration, we can imagine that the "noise" is a purely sinusoidal signal. If the sampling is at a frequency much lower than the frequency of the noise signal, the noise appears in the sampled signal at full amplitude, but at a frequency much lower than its original frequency. This is illustrated in Figure 15, where the solid line is the original (analog) signal and the dashed line is the sampled version.

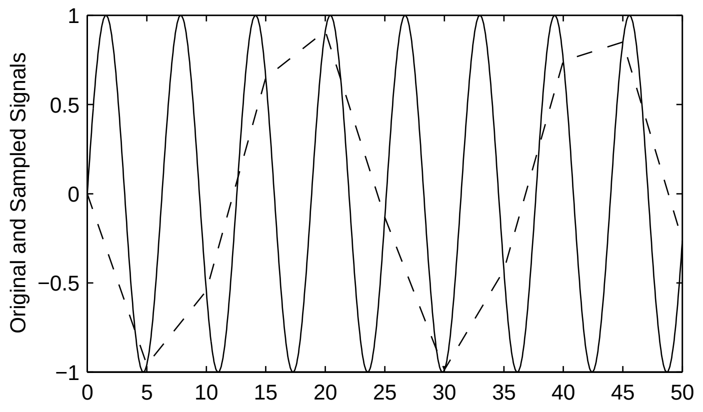

Figure 15: Original Analog Signal and Aliased Sampled Signal

Once aliasing occurs, the standard method of removing noise, spectral filtering, can no longer be used because the noise signal is now in the same frequency range as the information. For those used to working with analog controllers and signal conditioning equipment, aliasing is counter-intuitive since most analog equipment has natural low-pass filtering properties so that high frequency noise tends to disappear in many cases and is easily filtered in others.

There is no easy solution to aliasing. The two relatively difficult solutions are to add an analog, low-pass filter between the noise source and the sampling device or to sample at a high enough rate so the noise can be removed after sampling. Both of these add considerable complexity to the control system. Another approach is to avoid the problem entirely by using digital instruments such as position encoders (incremental or absolute).

## 5.6 Computational Delay

Any delay in utilizing feedback information causes deterioration in the quality of control, up to and including instability. The classical example of this is using a television camera on the moon to show the operation of a vehicle or machine to an operator on earth. Because of the speed of light limitations on transmission speed, it takes over a second for the television signal from the moon to reach earth. The operator is thus seeing what was happening a second and some ago, not what is happening at the time the image is being viewed. A decision is made on an action (for example, with a joy stick) but it takes another second and a fraction for that signal to get back to the moon. If the system on the moon being controlled is very slow, the three seconds or so of total delay won't cause much of a problem. If it is fast moving however, the operator will be playing a losing game by always commanding actions that were appropriate for the time the signal was generated, but are no longer appropriate by the time the signal actually reaches the actuator. Most human operators adopt a wait-and-see strategy when faced with this type of situation. It works, but the overall performance is vastly slower than it would be in the absence of the delay.

The time a computer takes for computation between reading the instrument signal into the computer and generating the actuation output introduces such a delay. The worst case of such delay occurs when a small computer is dedicated to the control of a single loop and takes the full sample time to complete its computation. This introduces a full step delay into the system.

For the two tank system treated above, Figure 16 shows that introduction of a full step delay in the small-signal (using the gains determined by the optimizer) results in an unstable and thus completely unacceptable behavior (note that in the original case the simulation was built with the imlicit assumption that the computing time was a small fraction of the sample time and thus did not introduce any significant delay). In such circumstances, the controller must be retuned and will only be able to sustain substantially lower gains and thus more sluggish response.

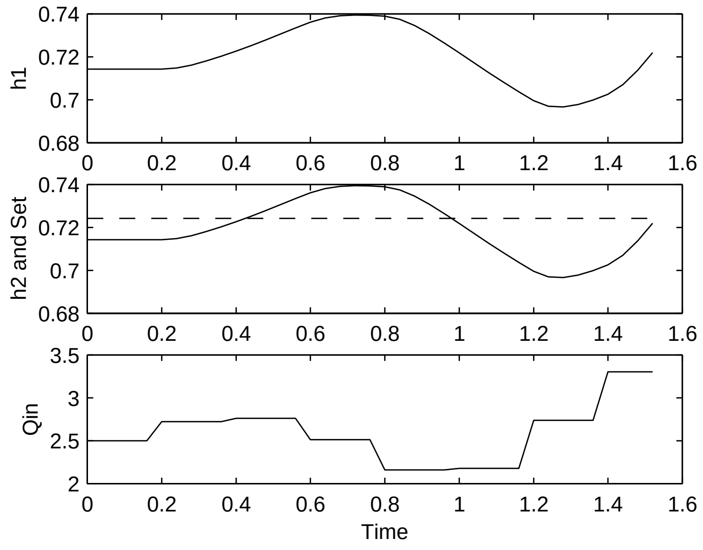

Figure 16: Two Tanks System: Full Step Delay

## 5.7 Using Knowledge: Feedforward

"Knowledge is power" so don't waste it! While the emphasis thus far has been on feedback control as a means to compensate for ignorance, in many cases there is quite a lot of knowledge as to how the system operates. That knowledge can be used as a feedforward signal in addition to the command signal from the feedback controller. If it is reasonably good, the control object's output will get fairly close to its desired value just with the feedforward. The feedback will then have much less work to do since the degree of ignorance is much less.

The two tank system offers two examples of potential use of feedforward. Knowing the setpoint, the input flow rate that will be needed for steady-state maintenance of that height in the second tank can be computed and added to the controller output. This does not change the observed behavior very much, but it does allow for controller gain tuning that nearly eliminates the integral gain. This makes the controller considerably more stable and robust as long as external unmeasured disturbances are not so large as to need the higher I-gain. On the other hand, if the flow disturbance described in Section 4.8 could be measured, it could be compensated for with feedforward almost instantaneously, reducing the system excursion due to that disturbance to almost nothing. A measured flow disturbance in the second tank would fall in the middle since the feedforward flow correction could be applied well before a significant change in the level of the second tank would be detected.

## 5.8 Multiple Measurements: Cascade Control

Control systems with one measurement and one actuation point (SISO for single-input, single-output) are very common, but do not by any means encompass the full universe of feedback control systems. A full treatment of multiple-input, multiple-output (MIMO) systems is beyond the scope of this article, but there is a widely used control structure that deals with many systems having several measurements but only a single actuation (these might be called MISO but that acronym does not seem to be used). More measurements than actuations is common because the cost of measurement is generally much lower than the cost of actuation.

When the output of one component of a system becomes the input to the next a "cascade" is said to be present. Examples of this are the power amplifier providing power to a motor or the pump providing input flow to a tank. In these cases, it is tempting to add an instrument to the intermediate output, measuring the current output of the amplifier or the flowrate from the pump.

The motion control example used in Section 5.2 took advantage of two measurements, position and velocity, with a single actuation, force applied to the mass. A cascade control structure was utilized with velocity control as the inner loop and position control as the outer loop. Most motion systems also have a control loop inside the amplifier so that the amplifier command controls the current applied to the motor. The two tank system could also use cascade control if the liquid height in the first tank were to be measured as well as the liquid height in the second tank. In that case, the liquid height control for the first tank would be the inner loop. This structure could be important if there were a need to make sure the first tank did not overflow since the single loop structure does not put any limits on height of the liquid in the first tank.

## 5.9 Gain Scheduling

When tuning a feedback control system that has a large operating range, it is often impossible to find a set of controller parameters that will operate successfully across the entire range. Tunings that are always stable may be excessively sluggish at some parts of the operating range, and those that give lively performance may be unstable or highly oscillatory at other operating points. Probably the most dramatic example of this situation is in the control of high performance supersonic aircraft. The dynamic behavior changes so much as the plane goes from subsonic to supersonic that it is essentially impossible to use the same controller for both situations. Even in one domain or the other, the dynamic behavior remains a very strong function of altitude.

A solution to this problem is to schedule the gains so that they take on correct small-signal tunings at different parts of the operating range. Computer-based controllers offer the ability to store large tables of gains, and, most importantly, to carry out the transition computations that give a "bumpless" transfer from one gain set to another. Bumpless transfer assures that the control output does not make a sudden change when the gain set changes. If a pure P-control were being used, for example, as the gain changed the output would also change. The presence of an integral action, even if the I-gain is zero, allows the integrator value to be used to balance the output changes from other control modes.

# 6 Design Based on Linear Models

If the real world is entirely nonlinear (and it is) why study linear system analysis? Because,

1. Small-signal behavior can often be adequately captured with linear models, and
2. Linear analysis is incredibly powerful when it is applicable.

## 6.1 The Magic of Linearity

Systems that are linear obey additive superposition. In qualitatitive terms, that means that if the response to one particular input signal is known and so is the response to another, the response to the sum of the two input signals is the sum of the corresponding outputs. In practice, this means that the solutions to all linear problems are expressed in series form, all using the *same* set of series: sines, cosines and exponentials (which are all part of the same series family).

Because the sinusoidal-exponential family is so well understood, behavioral characteristics of linear systems can be determined parametrically, without having to perform full simulations. The origin of most of the linear system material pre-dates computers so is designed to extract the maximum amount of information about a system with minimum computation.

Linear system analysis for control is covered in a large number of texts as well as in thousands of research papers and articles. For that reason, only a few highlights will be pointed out here. The reader can refer to the extensive literature for details.

## 6.2 Linearization

Linear models can be constructed from set of nonlinear differential equations, from simulations of those equations, or from experiments with the actual system. In all cases, a linear model is created that describes the system behavior near a specific operating point. When this process starts with data from experiments with the actual system, it is usually called system identification, but the end result is the same: a linear model.

When starting with a set of first-order (nonlinear) differential equations the object is to find the local slopes for the relationship between each of the system (state) variables in each equation and the rate of change of the corresponding state variable. This relationship can be expressed in matrix form; the matrix of local slopes is called the "Jacobian" matrix.

This procedure yields the classic "state space" form of the linear system equations for a control object,

$$\frac{d}{dt}\mathbf{x} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}; \quad \mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \tag{5}$$

where $\mathbf{x}$ is a vector of the system state variables, $\mathbf{u}$ is a vector of system inputs, $\mathbf{y}$ is a vector of system outputs, and $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ and $\mathbf{D}$ are matrices of (constant) coefficients. The sizes of the coefficient matrices depend on the number of states, inputs and outputs.

While the equation is often written in this form in texts, for real systems this should be viewed as an incremental description of the behavior near an equilibrium point. The state, input and output variables represent changes from the equilibrium values. This form is the basis for a large number of design and analysis methods based on linear algebra and calculus.

## 6.3 Transfer Functions

The most compact expression of a linear model is as an n-th order differential equation. The matrix format often has many elements that are zero and a large number of relatively simple elements. The n-th order form compacts all of the $n^2$ elements of the matrix format to $n$ elements. The transfer function form originates as a Laplace transform, but is visually identical to the n-th order differential equation format. The Laplace transform origin legitimizes manipulations using transfer functions as polynomials that make it very easy to combine linear elements and get a single transfer function for the combined system.

Transfer functions are freely convertable to state space matrix form and *vice versa*. The transfer function format is unique to a given physical system, but the state space format is not. A new set of state variables can be constructed from any linearly independent sum of other state variable sets. Like the state space format, a transfer function represents the incremental behavior near an equilibrium point.

"Classical" control design and analysis is based on the transfer function format. It focuses on the design of single-input, single-output (SISO) control systems. Transfer functions represent SISO systems very well, even for high order, complex dynamics.

## 6.4 Difference Equations

The state space and transfer function formats describe the continuous-time portion of the system, normally the control object. Computer controlled systems, however, are hybrids: the controller is discrete-time and the control object is continuous time. The discrete-time portion of the system is described by difference equations rather than the differential equations that describe the continuous-time portion. The discrete-time portion of the system description has two portions: a part that has been converted from the continuous-time model and a part for the controller. The first part is converted from a linear model, so will also be linear. The second part, the controller, is specified by the system designer. It is "linearized" by omitting the nonlinear parts (such as saturations).

The state space format for the discrete time equations is very similar in appearance to the continuous time version,

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{F}\mathbf{x}(k) + \mathbf{G}\mathbf{u}(k) \\ \mathbf{y}(k) &= \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \end{aligned} \tag{6}$$

The coefficient matrices are sometimes given different names to avoid confusion, and sometimes given the same names to indicate similarity of function. The index $k$ is the sample time counter.

Because the controller is normally the "open" part of the design, it is customary to convert the entire system to the discrete-time domain. There are a number of methods for converting the continuous time part to discrete-time. If the equations can be solved (or the transforms inverted) the conversion can be done exactly ("exactly" means that the responses will match at the sampling instants; nothing is said about the behavior between samples). There are also several approximations that are simpler to use, but only valid for specific ranges of sampling intervals.

## 6.5 Discrete-Time Transfer Functions

The Z-transform domain serves the same role for discrete-time systems that the Laplace domain does for continuous-time: it legitimizes the use of transfer functions to combine transfer functions for components and get a transfer function for overall system behavior. However, there are not as many design methods based on the Z-transform as there are for the Laplace transform based transfer function. As with the Laplace transfer function, conversions can be made back and forth to the matrix format with the same uniqueness properties.

## 6.6 Equilibrium, Stability and Eigenvalues

Equilibrium, as defined above, becomes a very simple matter for linear systems: in most cases there is only a single equilibrium. In some cases there are an infinite number of equilibrium points, but, even with these systems, with the application of feedback control the final system normally has only one equilibrium point. It is very convenient to transform the state variables for the linear model of a system so that this equilibrium point is at the origin of the state space, so the equations are often written in this form. Because of the linear properties, this transformation has no effect on the results.

The other property of linear systems is that the stability properties at this equilibrium point are global – that is, they extend throughout the state space. The linear model, of course, only has a limited domain of applicability, so this result can be interpreted as extending the stability properties at the equilibrium point to the full range of the linear model's validity.

A major use of linear systems theory has been to parametrically define stability boundaries. Since unstable performance of a feedback control system is absolutely unacceptable, this boundary, expressed in terms of system parameters (such as controller gains), separates the region of unacceptable from the region of might-be-OK.

With modern computing tools, it is possible to learn much more than just whether a system is stable or not. Linear dynamic systems, discrete or continuous time, are described by a set of numbers characterizing their behavior. There are n of these "eigen" or "characteristic" values for a linear system, where n is the number of state variables. Examination of the set of values indicates whether a system will be stable or not, oscillatory or not. The numerical values give information about oscillatory frequency or speed of response. For systems of modest size (that is, modest number of state variables) computing these values is simple and straightforward in engineering computation systems such as Matlab or in Fortran, C, etc.

## 6.7 Pole Placement Design

Since the eigenvalues of a linear system reveal so much about how a system will behave, it is natural to build a design method around placing them in desired locations. "Pole placement" (poles are another name for eigenvalues) is a means of doing just that. Inm some cases, the controller gains that will achieve the desired placement can be computed directly, in others optimization methods can be used to find the relevant gains.

## 6.8 Linear Quadratic Optimal Control

If the control system model with multiple measurements and a single actuator is carried to its extreme, there will be a measurement for every state variable. This represents the extreme because any additional measurements will not add any new dynamic information. Added measurements can be used for reliability or noise-reduction purposes, but from the perspective of linear controller design they do not add anything.

Feedback control algorithms can be designed to take advantage of all of these measurements. Linear quadratic optimal control (LQR for linear quadratic regulator) arises out of the much more general optimal control field. In general, an optimal control formulation will give the open loop input that is needed to optimize some specified performance of a dynamic system (it is closely related to dynamic programming). In the particular case of a quadratic performance index combining the square of the error and square of the actuation, the solution to the optimal control problem is a feedback control where the measurements used for the feedback are all of the state variables. In this formulation, each of the state variable is multiplied by a gain and the results are summed to get a single actuation value. The result of the LQR formulation is the set of gains, based on the relative weighting of the error and actuation in the performance index. The nice feature of LQR control as compared to pole placement is that instead of having to specify where n eigenvalues should be placed a set of performance weightings are specified that could have more intuitive appeal. The result is a control that is guaranteed to be stable (to the extent, of course, that the model actually describes the real, physical system).

It should be noted that the particular form of the weighting function falls out of the optimal control theory. Although it does strike most people as a reasonable performance index, that is only coincidence. If any other index is desired and indirect design method such as a numerical optimizer must be used to achieve the design.

It is also possible to LQR if the number of measurements is less than the number of state variables. This is accomplished through the use of an "estimator," a formulation that uses the linear system model to estimate all of the states based on measurement of only some of them. The derivative action in the PID control is a very primitive form of estimator that adds one estimated state variable to the measured variable that is used for SISO control.