

## Exercise 1

# Introduction to Control Systems – Simulation

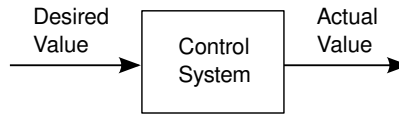
This exercise will take two weeks. It is an introductory lab that will show you many things about control systems engineering but presented in a not too technically encumbered fashion. Later in the course you will see these things again, but covered in a deeper fashion to take advantage of the mathematical competence you will have gained in the course. The goals of this lab are:

1. To introduce you to control systems in general, in a hands-on fashion, to two possible configurations, and to many of the tools one uses to design a controller.
2. To introduce you to PID controllers, the most common controllers used in industry.
3. To introduce you to first- and second-order dynamic systems and their responses to input changes.
4. To give you experience with a software package very commonly used for control systems analysis, Matlab™/Simulink™.
5. To show you how to make a simulation of a system, how to tune it, and how to use this simulation to design and tune a PID controller for a system.

### 1.1 Control Loop Configuration

Classical control systems are SISO systems, Single-Input-Single-Output, as opposed to MIMO systems, Multiple-Input-Multiple-Output, which are more complicated. For a control system the input is the desired value, and the output is the actual value (See Figure 1.1).

A good example is a cruise control system for an automobile. The user inputs a desired value, say 65 mph. Usually one does not type this in. One drives the car up to this speed manually, then pushes a button. The speedometer senses

Figure 1.1: *SISO control system*

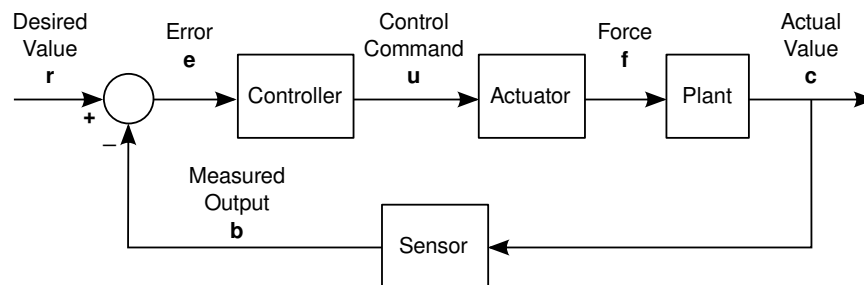
the speed, stores this in an on-board computer, and then it is the job of the cruise control to keep the automobile at this speed.

This type of control loop is called a regulator. It is the job of the control system to hold the automobile at this speed, even though there are disturbances that try to change the speed. For an automobile under cruise control, the primary disturbance is a change in terrain, an ascent or a descent.

Thus, when everything is working as it should, the actual value is equal to the desired value. In German these variables are known as the *Sollwert*, the “should-value”, and the *Istwert*, the “is-value.” In controls it is always good to bring things down to earth, because controls can get so theoretical that one quickly loses sight of what is going on or why one is doing what one is doing. So think of these two values as “what you want” versus “what you’ve got.” When what you’ve got isn’t what you want, then something’s wrong. When  $Sollwert - Istwert \neq 0$ , then the control loop is not doing its job, and something is broken or something needs to be changed to make this difference 0.

Actually this difference has a name, the error. That’s not error in the sense of a mistake. Rather it’s error in the sense of deviation. In a perfectly functioning control system, the error should be 0, and what you’ve got should be what you want.

Let’s look inside the control loop, at the anatomy of a control loop. Almost all control loops are the same. They are all made up of five components arranged always the same. Sometimes it is not easy to recognize these elements in an actual system. But it’s always a good idea to try. This structure is fundamental to control theory and represents the underlying functions that are needed to make feedback control work. As you probably have already concluded, the basic structure of a feedback control system is a loop (see Figure 1.2). The five

Figure 1.2: *Basic control loop anatomy*

### 1-3 *Introduction to Control Systems – Simulation*

elements are:

1. The comparator
2. The controller
3. The actuator
4. The “plant”
5. The sensor

Let’s discuss these components one by one, in the order that’s easiest to use to identify them in a real system. Usually the easiest element to identify is the sensor. For a cruise control system, the sensor is the speedometer. The sensor always measures the actual value and then feeds it back to the comparator to compare with the desired value. That’s the nature of feedback control, and that’s why it’s called feedback control: the actual value is fed back to the desired value and compared. Another common example is the thermostat in your house. A thermometer in your house measures the temperature in your house and then compares that with the desired temperature you have somehow entered on the faceplate of the thermostat.

The error signal is the output of the comparator. It is also the input to the controller. As you can see, all of these blocks in the block diagram of Figure 1.2 are SISO blocks, and each output becomes the input of another block. The controller takes the input from the comparator, the error, and decides how the system should respond. If the error is 0, then what you’ve got = what you want, and the system should do nothing. If the error is not 0, then the controller should take some action.

Nowadays, with digital controls, the controller is usually just a piece of software running in a computer somewhere. For a cruise control, there is a computer algorithm running in an on-board computer that performs this task. So if someone asked you to point to the controller in the cruise control loop, you’d have a hard time doing that without talking with the engineers that designed it. Or you could just point at a black box in the car and say, “There it is”, and most people would have a hard time disputing this.

If the error is not 0, then the controller needs to take action. Eventually it wants to influence the plant. This is a funny term for the thing that we actually want to control. But control theory grew up in industrial plants, so that is why this block has this name. The plant can be hard to identify. One identifies it often by asking, “What are we trying to control?” and then the plant is the thing that that value is a property of. For example in a cruise control loop, the speed is what we are trying to control. And the speed is a property of the car. So the car is the plant in a cruise control loop.

Often the actuator is the hardest component to identify, so often we leave it for last. Often it is hard to draw a line between the actuator and the plant. Often it’s hard to answer the question, “Where does the actuator end and the plant

begin?” You’ll see the answer to this dilemma with experience. In the meantime good questions to ask are “What does the controller talk to?” or “Where does the controller send its signal?” or “By what means does the controller influence the plant?”. In a cruise control system the plant is the car. The actuator is the throttle. Or maybe it’s the engine. It’s the thing that causes the car’s speed to increase when the controller notes that the car is going too slow and needs to speed up. What you have is less than what you want, so do something.

Thus these five logical components are always present in a classical control loop. Though it may be hard, it is always of value to try to identify the physical components that correspond to the logical components.

Note also in Figure 1.2 that the signals between the blocks also have names:

**r** desired value (“r” stands for reference value; this is also known as the controller setpoint)

**e** error

**u** command

**f** force

**c** actual value (“c” stands for controlled value)

**b** measured value

These variable names are not standard by any means, but one sees them often. You should be aware that often variations of them are used. But we need names for them so that we can refer to them when talking about what’s going on in the loop.

As stated before, it is the job of a regulator to maintain a desired value in the face of disturbances that tend to cause deviations in the actual value. Thus in a cruise control the car is going 65 and you want it to go 65. Then you encounter a grade, an ascent, and the car slows down. The speedometer senses this, the error becomes positive, and the controller tells the throttle to open to make the car go faster. So where are the disturbances in the loop?

Figure 1.3 is a modified version of the loop shown in Figure 1.2. The disturbances are shown as external influences that come into the loop between the actuator and the plant. Note that sometimes the disturbances come in as positive influences and sometimes as negative influences. In a cruise control system, sometimes you encounter uphill, sometimes downhill.

As also stated, there are two basic kinds of control loops. One is the regulator, presented with the cruise control as an example. The other is the positioner. Actually Figure 1.2 shows a positioner, sometimes called a tracker. While the regulator maintains a desired value in the face of disturbances, the positioner follows the desired value as it changes. This is what one sees in remote control systems or in systems for force amplification.

A good practical example of a positioner is the fly-by-wire system used to pilot a modern airliner. Under manual control, *i.e.* with the pilot pulling and

## 1-5 Introduction to Control Systems – Simulation

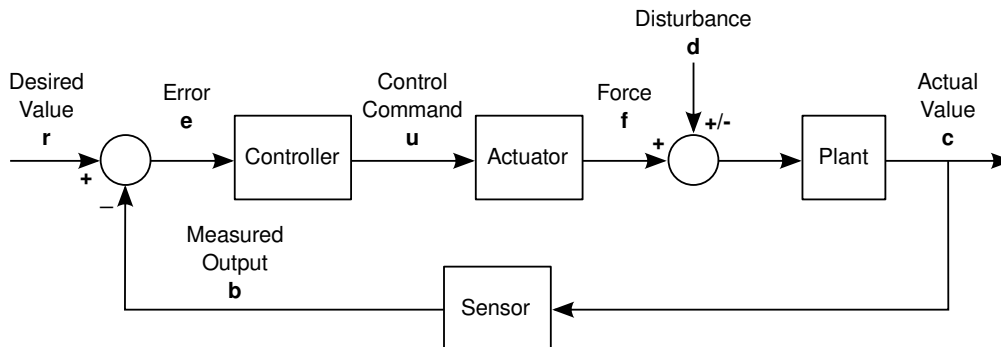


Figure 1.3: Regulator control loop

turning the steering yoke and pushing the rudder pedals, there is no direct connection between his or her actions and the movement of control surfaces that actually steer the airplane. Rather there are sensors on the cockpit controls that sense the pilot's motions with these input devices. Then based on the positions sensed, corresponding motions are commanded on control surfaces that correspond to the pilot's actions in the cockpit. So, for example, a movement backward of half an inch on the control yoke corresponds with an angular deflection of one inch of the elevator surfaces on the horizontal stabilizer at the tail of the plane.

So why go to all this trouble? Why not just hook the yoke directly up to these control surfaces like is done on a small, general aviation airplane? The problem is that with the size of the airliner and the speed it flies, the pilot would have to be Arnold Schwarzenegger to fly the airplane. With a fly-by-wire system, the force applied to move the control surfaces is usually supplied by a hydraulic cylinder, and a hydraulic cylinder can supply a much greater force than even Arnie can.

A positioner is not made to reject disturbances like a regulator is. Thus we assume there are no disturbances trying to knock the actual value off the desired value. The loop is as shown in Figure 1.2.

In sum, the differences between a regulator and a positioner are:

In a regulator loop:

- There are disturbances
- The desired value changes seldom if at all
- The main aim of the control loop is disturbance rejection

In a positioner loop:

- There are no disturbances
- Changes in the desired value are frequent

- The main aim of the control loop is to follow the changing desired value

You might look at the regulator loop and say, “Hey, that’s not a SISO loop. There are two inputs. That’s a MISO system.” And you would be right. But as is often the case in controls, we find a trick way to force it to be a SISO system. What we do is we take the desired value and make it our reference value. Thus it becomes a kind of 0. Thus nothing is coming into the loop at this point, so we can rework the loop into the configuration shown in Figure 1.4. Note that

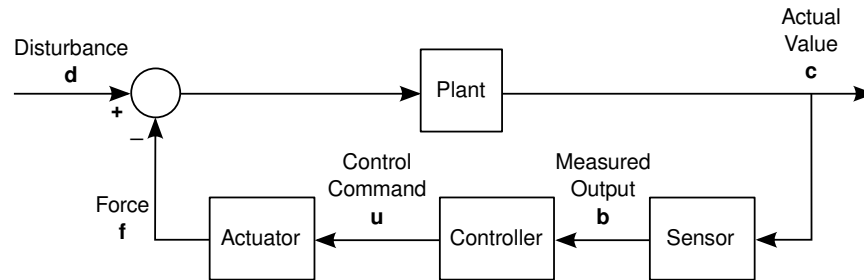


Figure 1.4: *Reworked regulator loop*

the disturbance is now the input. In going around the loop the comparator still subtracts the sensed value from the now-zeroed desired value, so a negative sign is introduced to the sensed value. We can carry this forward to the disturbance summing junction, where it changes the sign. This may seem a little confusing at first. We’ll see how to deal with this once we start modeling systems with Simulink™.

### 1.1.1 PID Controllers

PID (Proportional-Integral-Derivative) controllers are by far the most common controllers used in industry. The name refers to three different actions that the controller makes in responding to a non-zero input, the error, as we have seen above. Thus we speak of proportional action, integral action, and derivative action. The three actions occur simultaneously. The configuration of the controller is a parallel configuration, as is demonstrated in Figure 1.5. What is shown in this figure are the contents of the block labeled “Controller” in the loop diagrams above. Note in the figure that the input signal, the error, is first treated one way or another and then multiplied by a constant. The top path is the proportional path. Here the output is proportional to the error, hence the name. There is no action taken on the input signal. It is just multiplied by  $K_P$  and then passed on downstream to the output. The integral action is the second path. Note that the error (the input) is first integrated (multiplied by  $1/s$ ; this will be explained in the study of Laplace transforms). The output of the  $1/s$  block is the integral of the error. Thus if you plotted the error curve *vs.* time, this signal would represent the net area under this error curve through time. This is then multiplied by  $K_I$  and becomes the integral action. The

## 1-7 Introduction to Control Systems – Simulation

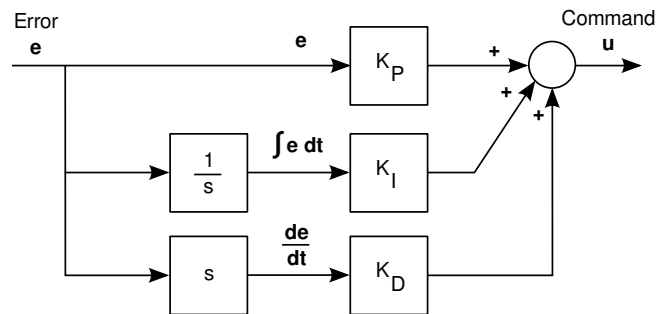


Figure 1.5: *PID controller configuration*

derivative action is the third path. Note that the error is first differentiated, *i.e.* multiplied by  $s$ . The output of the  $s$  block is then not the error but the rate of change of the error at the current time. This change rate is then multiplied by  $K_D$  to become the derivative action. All three actions are added together in the summing block to become the total PID controller action.

Why one would do this is at this point not clear at all. But as we shall see, each of these actions has a specific use or justification and usually improves the control response. The three constants –  $K_P$ ,  $K_I$ , and  $K_D$  – are called the controller gains.  $K_P$  is the proportional gain,  $K_I$  is the integral gain, and  $K_D$  is the derivative gain. It is also often the case that one of the actions is not present. As we shall see, the proportional action is by far the most sensible and useful action. Often controllers have only P action – that is,  $K_I = 0$  and  $K_D = 0$ . We call these “P-only controllers” or just “P-controllers.” A controller with no D action is called a “PI controller.” One with no I action is a “PD controller.” So we encounter P, PI, PD, and PID controllers. Note that all of these have P action. There may be an oddball case without P action, but that is what it is, an oddball case.

### 1.1.2 Exercise 1

Use Simulink to model a PID controller. Make all gains 1 for now. For input, put in a unit step. Plot all three actions and then also the total controller action. Run the simulation for 10 seconds and see what happens with all actions. Deliverables: Plots of the three actions, then some commentary on your observations of them. How do they behave over time?

The rationale behind the three actions will become clear through activities in the lab. But we can start with the sense of proportional action. Remember that the input to the controller is the error, the deviation between the desired value and the actual value. If the error is 0, then we don’t want the controller to do anything. What you’ve got is what you want. Hands off. So the proportional action is 0. If the deviation between desired and actual is small, you want the controller to give the plant a nudge rather than a large action. So the controller

takes the small error, multiplies it by the proportional gain, and then issues a small output command to the actuator (which then moves the plant). If the deviation is large, then the difference between the desired and the actual value is large, and you want the controller to command a large action. The large error is multiplied by the proportional gain, and this large command is then sent to the actuator. Thus proportional action, a commanded action that is proportional to the error, is just common-sensical.

We will do a lot with just proportional control, so you need to understand it well. In tuning the controller, one is adjusting the gains. One always starts with the proportional action and tunes the P part first. After this is set up correctly, one adjusts the I and the D parts, if they are to be used. They are like fine tuning adjustments to the P part. We will discuss these two parts later, but for now their overall purpose will be stated: The I part is used in the event there is a steady-state error, that is, there is a steady deviation of the actual value from the desired value. How this arises will also be explained later. The D part is used to eliminate future error, before it has a chance to develop. With D action one can predict that there will be error before the error develops. D action is predictive or preventive control.

## 1.2 First-Order Systems

In classical control theory (and in the real world) there are two types of systems that are encountered first-order systems and second-order systems. These will be discussed more fully in the lecture part of this class, but their basic behavior when excited will be here described. These systems are physical systems abstracted. For example a very good example of a first-order system is a tank with an inlet and an outlet. More specifically the level in the tank is the value of interest. Maintaining a certain level in a tank is a function that one finds very, very often in chemical, process, or power plants. There is actually a two-tank experiment in the lab that you will do later, so what is covered here will be useful for that lab. There are many other physical systems that can be represented as first-order systems.

With both first- and second-order systems, it is common to talk about their behavior when they are subjected to a sudden change in input. Take the tank in Figure 1.6. Notice that the input flow is governed by the inlet valve. Picture this tank at steady state, with the inlet flow equal to the outlet flow and the tank level stationary at a certain height. If this inlet valve is suddenly opened to a wider setting, the inlet flow will suddenly increase. Then the inlet flow will be greater than the outlet flow, and the level in the tank will rise. The rise in tank level will cause the pressure in the bottom of the tank to increase, and the outlet flow will increase. This will continue until the outlet flow increases to the amount of the new inlet flow. At that point, the tank level will become once again stationary. So the system will have reached a new, higher level of steady-state operation.

The tank system is just a very common example of a first-order system.

## 1-9 Introduction to Control Systems – Simulation

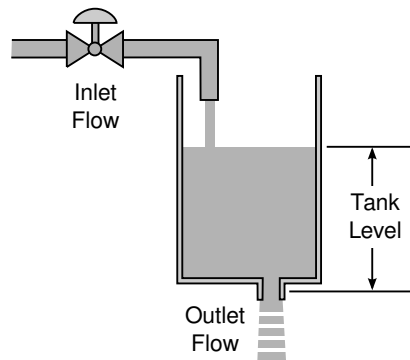


Figure 1.6: *Tank system*

There are many, many such systems in nature. In general, a first-order system, when disturbed, rises first quickly from the original steady state. This rise decreases gradually, and then the system reaches a new steady state. Figure 1.7 shows such a reaction. This reaction to a sudden input change is called a step response. A common test in system dynamics and controls is to subject systems to step inputs, observe their responses, then analyze the response to see what type of system you have and what its dynamic parameters are.

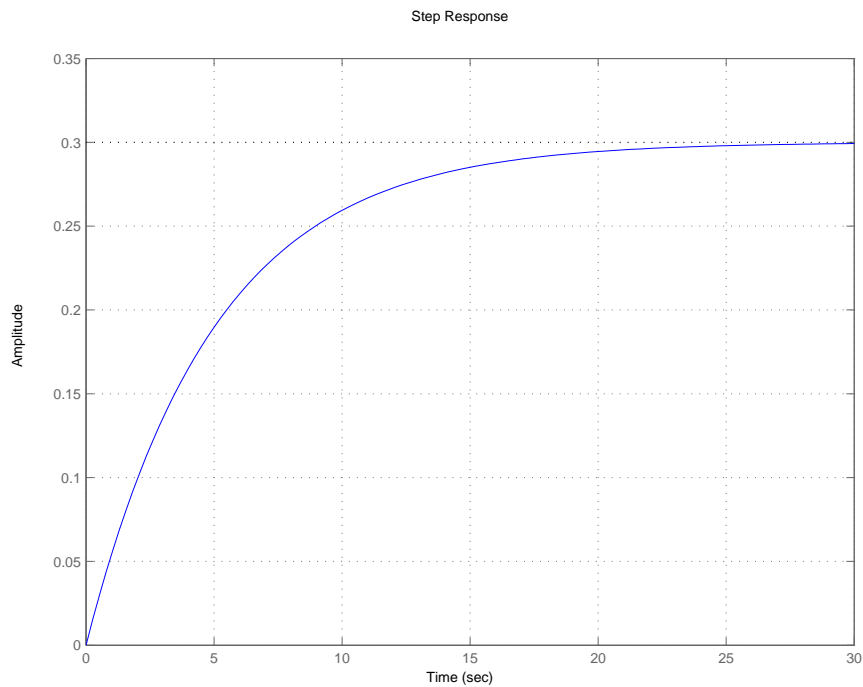


Figure 1.7: *Typical first-order system time response*

A first-order system has two parameters –  $K$ , the steady state gain, and  $T$ , the system time constant. The steady state gain is a measure of the amplification of the input by the system. So, for example with the tank, if a 5% increase in inlet valve position produced an eventual 1.5 inch rise in tank level, we'd say the gain of the system is  $K = 1.5 \text{ in}/5\% = 0.3 \text{ in}/\%$ . Funny units, but when figuring out steady state gains, it is always output/input. Funny units often turn up in controls, so just get used to it. It's always (effect) / (cause).

The time constant is a measure of how fast or slow the system is. For reasons that will be made clear later, the time constant of a first-order system is the amount of time it takes the system to reach 63.2% of its final value after a step response. Often one wants to know how long one must wait before the system responds and this transitional dynamic is “over.” The rule of thumb in industry is either 3 or 4 time constants. So after a time  $4T$ , the system is 99.9% of its way from the start point to the steady state value.

The model of this system is its *transfer function*. A transfer function has an input and an output. It reacts with the input and produces the output. It transfers the input to the output, hence the name. Sort of a stilted definition, but that's part of controls too. The transfer function for a first-order system is

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

This representation involves the Laplace transform, which will be covered in the lecture. Actually Simulink™ does all of the Laplace transform math that you need in controls. So just accept this definition and let's see how to implement such a model in Simulink™.

### 1.2.1 Exercise 2

Simulink™ has a transfer function block that you can use in a model of a first-order system. Find this block and install it in a new model. As input use a step function. To look at the output, get a scope block and put it on the output of the transfer function. Let's say we want to use the rule of thumb that the step response of this system is over after 4 time constants. Let's say we have a system that has a response that lasts 4 seconds. Let's say that the system has a steady state gain of 0.5. Make a model of this system and look at its step response. Does it behave as desired? How long does it take for the step response to reach 63.2% of its final value? What was the size of the input step? What size output change did it produce? How could you change the system to produce in the same time frame a steady state response that is twice as great as the step input? How could you speed the system up to make it respond twice as fast as the original system?

Set up two more systems with a) a gain of 8 and a time-to-respond of 0.8 seconds and b) a gain of -0.7 and a time-to-respond of 58 seconds.

**Deliverables:** A printout of the model of the first system mentioned above with its step response. Then written-out responses to the questions posed.

## 1-11 Introduction to Control Systems – Simulation

Printouts of the two models (make sure that one can see what's in the blocks, i.e. make them large enough for this); printouts of their response plots. On the response plots, indicate what shows the gains and what shows the time constants. You can do this simply by annotating the plots by hand.

### 1.3 Second-order Systems

Far more interesting than first-order systems are second-order systems. These systems can actually oscillate. A typical example of such a system is the suspension system of an automobile. It oscillates, but the oscillations die out due to the internal friction of the shock absorber. While first-order system response is determined by two system parameters  $K$  and  $T$ , second-order system response is determined by three parameters –  $K$  again,  $\omega_n$ , and  $\zeta$ .  $K$  is just the same as it was with the first-order system, the system gain.  $\omega_n$  is the system's natural frequency, which is the frequency of oscillation the system would experience with no damping ( $\zeta = 0$ ), and  $\zeta$  is the damping ratio of the system, measured between 0% and 100%. If  $\zeta$  is between 0 and 1, the system oscillates, but the oscillations die off. This will be covered in much greater depth in the lecture. A typical second-order step response is shown in Figure 1.8. As is true also with first-order systems, often a second-order system is subjected to a step input, its response is captured on an oscilloscope, and then this is analyzed to determine the three system parameters  $K$ ,  $\omega_n$ , and  $\zeta$ . In fact often before the step response test, the order of the system is not known. One applies the step input, observes the response, and says “Oh, that is a first-order system... and  $K$  and  $T$  are...”

Not all second-order systems are oscillatory as you will see in lecture. Oscillatory systems occur only if  $0 < \zeta < 1$ . And the system does not oscillate at its natural frequency. Rather it oscillates at its damped frequency,  $\omega_d$ . You should remember from Vibrations that  $\omega_d = \omega_n \sqrt{1 - \zeta^2}$ . If one is analyzing a second-order system from its step response, one first looks at the change in the steady state values of the system to get  $K$ , then one measures  $\omega_d$  from the response curve. One then looks at the overshoot. This is the amount that the system oscillates beyond the final value on the first oscillation. From this one can calculate percent overshoot, or “%OS.” In the example shown in Figure 1.8, %OS is a little over 50%. As you will see in the lecture, there is a direct correlation between %OS and  $\zeta$ , namely

$$\%OS = e^{-\zeta\pi/\sqrt{1-\zeta^2}} \times 100\%$$

or

$$\zeta = \frac{-\ln(\%OS/100\%)}{\text{sqr}t{\pi^2 + \ln^2(\%OS/100\%)}}$$

With  $\omega_d$  and  $\zeta$ , one can calculate  $\omega_n$ . Then to check this, one builds a system model with the calculated  $K$ ,  $\omega_n$ , and  $\zeta$  and checks that the simulated step response is the same as the original step response. If it is not, then you've made

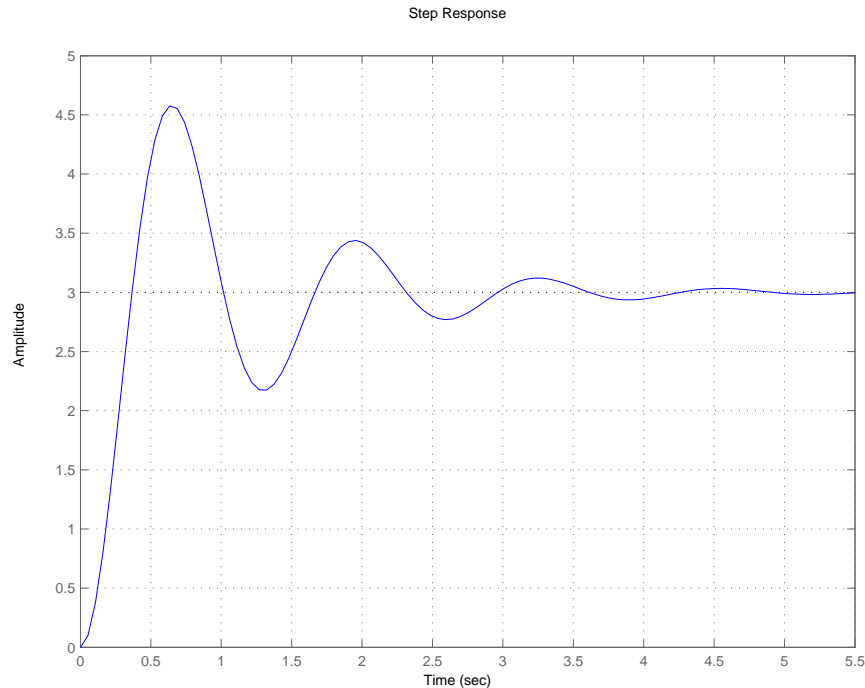


Figure 1.8: *Typical second-order system time response*

a mistake, and you need to find and correct it. The transfer function for a second-order system is

$$G(s) = \frac{K\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

### 1.3.1 Exercise 3

Construct a system with  $K = 3$ ,  $\zeta = 0.15$ , and  $f_n = 1$  Hz. Perform a step response test in Simulink™ and check the results. You can do this by checking the size of the steady state change, the oscillation frequency, and the expected %OS.

Now take the step response shown in Figure 1.8 and analyze it. Figure out  $K$ , %OS, and  $\omega_d$ . From Figure 1.8 figure out  $\zeta$  and  $\omega_n$ . Use the transfer function block in Simulink™, model the analyzed system, and see if you have calculated the parameters correctly. Double the size of the input step. Think about what effect this will have on your system. Apply this step and see if the results are as expected.

Go back to the original system, *i.e.* the one with  $K = 3$ ,  $\zeta = 0.15$ , and  $f_n = 1$  Hz. Double the damping coefficient. Think about what effect this will have on the step response. What will be the new overshoot. See if the effect is as you expect. Go back to the original system. Double the oscillation frequency. See if the effect is as you'd expect.

## 1-13 Introduction to Control Systems – Simulation

**Deliverables:** Printouts of all systems mentioned here along with their step responses. Annotate step responses to make clear that they show what was asked for.

### 1.3.2 Exercise 4

Construct a system with two first-order systems in series, that is, the first system's output is the input to the second system. Let both systems have a gain of 2 and a time constant of 1 second. Perform a step response analysis of this system and see if it does what you expect it to. Probably at this point, you do not know what to expect, but see if the step response has anything in it that is surprising. What is the steady state gain of the entire system? Does the system behave like a first-order system? Technically, two first-order systems together like this constitute a second-order system. Does this system oscillate?

Change this system's constants –  $K_1$ ,  $K_2$ ,  $T_1$ , and  $T_2$  – so that the two first-order systems are very different. Let one system have a small gain, say 0.1, and a small time constant, say 0.1 second. Let the other system have  $K_2 = 10$  and a time constant of 5 seconds. Perform a step response, analyze the response, and note what role each of the two subsystems plays in determining the overall system response.

**Deliverables:** Printouts of each system mentioned along with their respective step responses. Annotate the step responses to answer the questions posed.

## 1.4 The Control Loop

Now that we know a little about how control loops are configured and a little about first- and second-order systems, let's do some playing around with that in Simulink™. First you need to know a little about block diagrams and block algebra.

As you have already seen, the way a block works is that its contents are multiplied by the input, and then this becomes the output. So, for the example in Figure 1.9, we can write  $y = G(s)x$ .

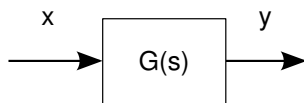


Figure 1.9: Block diagram for the transfer function  $G(s)$

The general control loop of Figure 1.2 is repeated in Figure 1.10 with each signal  $r$ ,  $e$ ,  $u$ , etc. replaced by its Laplace transform  $R(s)$ ,  $E(s)$ ,  $U(s)$ , and so on. You will soon learn how the Laplace transforms of signals are used to analyze the response of a system. Since the controller, actuator, and plant are

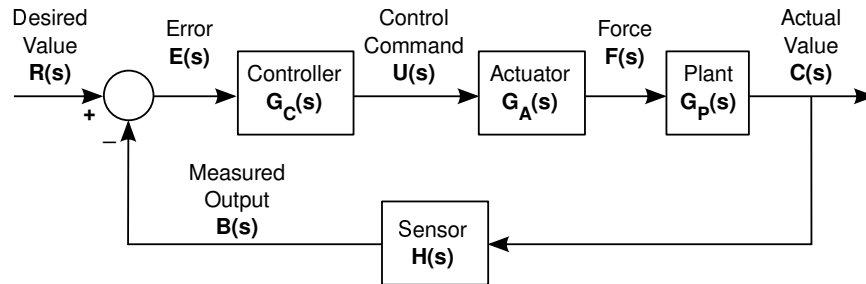


Figure 1.10: *Basic control loop anatomy*

all just linked in series in the feedforward part of the loop, we can combine them all into one single block. Thus

$$G(s) = G_C(s)G_A(s)G_P(s)$$

The *feedback* part of the loop contains just the sensor, with the transfer function  $H(s)$ . So the loop can be simplified to what is shown in Figure 1.11.

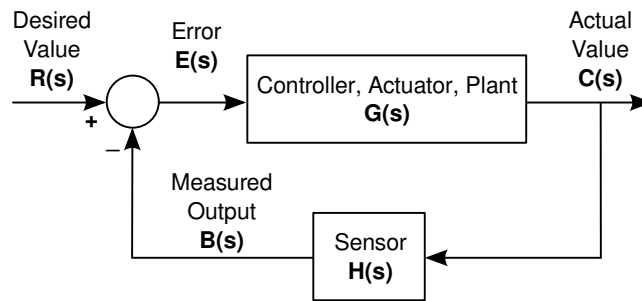


Figure 1.11: *Simplified control loop*

Note that

$$C(s) = G(s)E(s)$$

$$B(s) = H(s)C(s)$$

$$E(s) = R(s) - B(s)$$

It would be nice to replace this entire loop with the overall transfer function of the loop, that is, with one block. This is called the closed-loop transfer function,  $G_{CL}(s)$ . We substitute the second equation into the third, and then the third into the first:

$$E(s) = R(s) - H(s)C(s)$$

$$C(s) = G(s)R(s) - H(s)C(s)$$

Then

$$C(s)[1 + G(s)H(s)] = R(s)G(s)$$

## 1-15 Introduction to Control Systems – Simulation

And finally

$$G_{CL}(s) = C(s)R(s) = \frac{G(s)}{1 + G(s)H(s)}$$

This equation is one of the most important ones you will encounter in controls, so you should commit it to memory. Note that this applies for a feedback loop where the feedback branch of the loop feeds into the feedforward branch with a negative sign.

Now let's use what we've learned to see how some real control loops might behave.

### 1.4.1 Exercise 5

Let's say we have a first-order system we're trying to control. It could be the tank system shown in Figure 1.6. Though this is a regulator system, we'll first analyze it as a positioner system. So we will operate the tank as if were a positioner and drive the tank level up and down. We'll say that the tank is a first-order system with input being flow in gal/min and output being in tank level in inches. Thus  $K$  will have units inches/(gal/min), curious units, but that's how controls works. Let our tank have a steady-state gain of 1 inch/(gal/min). What this means is that if one increases the flow into the tank by 1 gpm, the tank level will rise 1 inch and settle at this new level. Let's say the tank takes 60 seconds to respond to a commanded change in desired level, so using our  $4T$  rule of thumb,  $T = 15$  seconds. Let's model the inlet valve as a simple gain, since the valve reacts very quickly compared with the tank. Let  $K_v l_v = 1\text{gpm}/10\%$  valve opening. Thus  $K_v l_v = 0.10\text{ gpm}/\%$ . Get used to these weird units because they are just part of controls. The gain always has units equal to output units/input units, whatever they may be. You do not want to be unable to deal with units in a control loop. They are always to be checked and often make clear mistakes that you have made.

Now we need a controller. Let's start with just a P-only controller. Note in the loop that the input value is the desired tank level. Let's let this be 0 inches. It really is not 0"; it is some non-zero value. But we make this value a reference value and then let the loop just work on deviations from this reference value. We will do an exercise below (Section 1.4.3) that makes this clear. It requires just a small modification of the model you are making in this exercise. Since the input is in inches and the valve works in % opening, the  $K_P$  of the controller must have the units %/inch. We will simplify the function of the sensor and just say that the sensor reads inches of tank level directly. So the feedback branch delivers inches to the comparator. Thus there is nothing in the feedback loop. We call this a unity-feedback loop. The output is wired directly to the comparator. Later we will see in detail how this happens.

Create a model of this system. Let the controller gain be 1%/inch. Run a step response of this system. See how the system responds. If you let the input be a unit step, what you are doing is commanding the tank level to go from its reference value to a new value 1 inch above the reference value. What does the

step response look like? What is the steady-state gain of the closed-loop system compared with the steady-state gain of just the tank alone? What is the time constant of the closed-loop system compared with the time constant of just the tank alone? Use the block diagram algebra above to verify that the responses seen actually make sense. That is, calculate the steady-state gain and the time constant of the closed-loop transfer function and verify that the step response displays these characteristics. Let's say you want to double the speed of the closed-loop system without changing its steady state gain. How would you do this? Do it with your simulation and verify that your technique works.

Another thing to check is to see whether, when you command a 1" increase in tank level, you actually reach a 1" increase. Look at the steady-state output value. Is it 1"?

Now put in a 1" decrease from the reference value. How does this work?

Put in a 1" increase in tank level at  $t = 1$  second, then a 1" decrease from the reference value at  $t = 75$  seconds.

Remember in this exercise that the control loop is working on deviations from a design operating point, not in whole or absolute or entire values. The best way to envision this is to think of the loop before any step is applied to it. Its input is 0 and its output is 0 too. In fact if you follow the multiplications of the blocks around the loop, you will see that every signal in the loop is 0 with 0 as input. But the total values around the loop at this operating point are not 0. In Section 1.4.3 below, we tell you that for a particular case, this operating point is actually 18" in the case of the design tank level. There is also a design valve opening, a design flow rate, etc. What the loop calculations involve are deviations from these design operating conditions. For example, we could size an inlet valve for the system that is, say, 75% open at this 18" operating point. Then the design operating point for the valve is 75%. So to see the real valve opening, one has to take what appears as the valve opening in the loop (the signal coming out of the controller) and add 75% to it. One would have to do something similar to this for all signals in the loop – *i.e.* take each one and add to it its design operating level. This may be confusing at this stage. Simply continue, and with practice and experience it will begin to make sense.

One thing that you almost always have to look out for with all control systems is something called saturation. A control loop's actuator is limited in power. In the case of a tank, the valve can only open 100% and close down to 0%. In the above example with a design operating point of 75%, the valve signal coming out of the controller can only go up to 25% ( $75\% + 25\% = 100\%$ , valve fully open) and down to  $-75\%$  ( $75\% - 75\% = 0\%$ , valve completely closed). In your model, you have nothing that prevents the controller from commanding the valve to open up to 150% (75% signal), for example, or close down to  $-50\%$  ( $-125\%$ ). So your model has a flaw in it. It has this magical valve that can do things that a real valve cannot. Investigate whether or not your system's valve is saturating. (If it's not, you can make it saturate by turning up the controller gain.) Investigate the saturation block in Simulink™ and install it where it makes most sense in your model. With the controller gain commanding

## 1-17 *Introduction to Control Systems – Simulation*

impossible valve openings, make sure that the saturation block does not allow the valve to open greater than 100% or close below 0%. For this the saturation block limits will be set to 25 and -75.

**Deliverables:** Printouts of all models mentioned along with their respective step responses, annotated to answer the questions posed.

### 1.4.2 Exercise 6

For this exercise, let's take a little more generic approach. We will ignore units and not even mention what we are modeling. You should get used to this way of speaking and thinking about things in controls. It is a new way of considering things, but you need to develop the understanding necessary to think and talk this way to get really good with controls. It represents the abstraction needed from the real world to be able to look into the essentials of control theory.

Let's take a second-order system with  $K = 2.3$ ,  $\zeta = 0.15$ , and  $\omega_n = 3$  rad/sec, and control it with a P-only controller in a unity-feedback control loop. Give  $K_P$  a convenient value, just something to start with, say  $K_P = 1$ . Subject this system to a step input and view the step response. Print this out and figure out with this drawing what type of response it is and what its parameters are. Do the block diagram algebra and verify that what you have measured is correct.

What would be the effect of doubling the controller gain? Predict what will happen with response parameters before staging this simulation. Make the new simulation and verify that what you predicted actually transpires.

**Deliverables:** Printout of model of closed-loop system with its step response with  $K_P = 1$ . Annotate the response plot to show the closed-loop steady-state gain, the closed-loop percent overshoot, and the damped period of oscillation. Show your calculations of the closed-loop transfer function in symbolic form (that is with variables instead of with numeric values.) Show your predictions of what will happen to the response parameters of the closed-loop system when you set  $K_P = 2$ . Printout of the step response, annotated to show that the predicted differences actually do occur.

### 1.4.3 Exercise 7

Modify the exercise in Section 1.4.1 to work with real tank levels, not just deviations from a reference tank level. Let's let the reference tank level be 18". Set the system up to allow a user to put in the absolute tank level desired and have the output read out in absolute tank level. If you put in a desired change from 18" to 19", does the tank level actually reach 19"? Put in a change from 18" to 17" and investigate the response of the tank.

**Deliverables:** Printout of modified system showing how total or absolute values are converted to deviation values and then back again. Printouts of the step responses mentioned, annotated to make clear what they show.