

Control Systems Engineering

A Practical Approach

by Frank Owen, PhD, P.E.
Mechanical Engineering Department
California Polytechnic State University
San Luis Obispo, California
May 2012

Table of Contents

Preface

Acknowledgements

Chapter 1 – Introduction to control systems

Chapter 2 – Laplace transformations

Chapter 3 – System modeling

Chapter 4 – First- and second-order system response

Chapter 5 – Stability

Chapter 6 – Steady-state error

Chapter 7 – Root locus

Chapter 8 – Frequency response

Chapter 9 – Designing and tuning PID controllers

Chapter 10 – An introduction to digital control

Preface

Preface

Why this book?

This book has been written for controls students at Cal Poly quite simply to save them money. There are many, many good controls books available, but they have, in my opinion, three flaws.

- 1) They are very expensive.
- 2) They are rather reference books than a basic, first book—what one needs when first approaching the subject. Thus we have found at Cal Poly that we buy a book for a lot of money and then use only a small part of it. It is not that the parts that we don't use don't have any value. They do. But one doesn't need to buy a complete reference book to understand the basics and the essentials of a topic.
- 3) They are highly mathematical. Controls is a very mathematical topic, perhaps the most heavily laden mathematically in mechanical engineering. There are many good engineers in industry that are not particularly adept at mathematics, who practice engineering with as much intuition and common sense as mathematical adeptness. A mathematical approach to controls loses sight of this, leaves many people behind, and does not take advantage of the fact that this topic also makes a lot of sense, a lot of common sense. Thus the approach taken here is to include what math is necessary but to appeal to common sense and intuition whenever possible. With today's modeling tools—read here Matlab/Simulink—a great deal of the math can be skipped and replaced with model building, to pose and answer questions that start with “What would happen if we...?”

In addition it has always been my conjecture that what we have developed at Cal Poly in our controls lab would also be very useful to controls engineers in industry. Our lab, while not unique, is very rare. It brings controls down to earth and teaches controls engineers how to deal with real systems, how to model them and then tune the models, and how to set up and tune PID controllers for real systems. These are the essential skills that a controls engineer must have to operate in industry. In my experience in academia, these *essential* skills are not often taught. Controls students have their heads filled with mathematics, indeed the mathematics of complex numbers, but then they are not given even a starting notion of how such knowledge is applicable in the real world. This book focuses ever on the real world of controls in industry. It tries never to lose sight of that goal and tries to avoid the alluring trap of mathematical elegance and indeed mathematical snobbishness that seems common in the field of academic controls. So the book has also been written for industrial practitioners of control theory who need to understand the topic and then bring into play to their advantage.

The other influence that led me to write this book was the three years I spent teaching controls in Germany, two years at the Munich University of Applied Sciences and one year at the Karlsruhe University of Applied Sciences. In Germany textbooks are rare. Rather students work from a *script*, a collection of the professor's notes organized and printed for student use. This book is really a script, a collection of my notes from teaching controls over the past decade. Though writing a book is a lot of

Preface

work, it's common practice to have short, directed scripts at low costs for poor students. So I thought, why don't we do the same at Cal Poly? We have lots and lots of experience teaching controls, so we should be able to come up with a good script. Besides, with our involvement in our laboratory, we have already demonstrated that we can come up with a high-quality document for teaching the lab portion of the course.

Thus it is my hope that students will benefit from this practical approach to controls just as they are assuredly benefiting from saving almost \$200 (in 2010). And I hope that this script serves as an example of what could be done in other courses at Cal Poly if professors would take their hard-won experience, collect it, and make it available at low cost to those eager to learn but without a lot of money to buy expensive reference books. This does not mean that one shouldn't buy the expensive reference books. Maybe one needs them in his or her work. But at that stage, one has the means to buy them or one's company will buy them when there is a need.

The use of Matlab/Simulink

It is hard nowadays to envision practicing controls engineering without Matlab/Simulink. The employment of this software in analyzing systems and designing controllers—indeed now in running real controllers in physical systems—is de rigeur. This text does not include a tutorial in learning Matlab/Simulink. That's available online or with the software. It is assumed that the reader has some knowledge of this software. Problems are posed in the text that directly direct the student to use this software. Occasionally tips are given in specific applications that illustrate the utility of a particular Matlab command or Simulink procedure. If the user's knowledge of this software is not at a level where these references to it make sense, he or she should explore the software a bit, researching its help facility for background knowledge. Controls requires knowing about only a tiny bit of Matlab and Simulink. So the reader is not required to do any extensive foundation-building in order to be effective with Matlab/Simulink in his or her study of the subject.

Acknowledgements

Wow, how did a hillbilly guy from a lawyer family in Mississippi ever get to the point that he could sit down and write a controls book almost directly out of his head? Well, if I told that whole story, that'd be a book in itself. Lots of hard-won experience but also lots of help along the way. It's always been my contention that when bestowing thanks, we never go far back enough. So I want to at least go back and thank my high school mathematics and physics teacher, Mac Egger. He didn't plant the original seed, but he was there close to the beginning. Then there were lots of twists and turns to get to this point. Along the way: Glen Masada at the University of Texas at Austin taught me classical controls when I went and got a mid-career PhD. Just before that I worked at one of the largest coal-fired plants in the United States, American Electric Power's Gavin plant in Gallipolis, Ohio. A plant engineer there taught me a lot, Randy Scheidler. He taught me a lot about power plants but also about the level of knowledge of good plant engineers in the United States. Randy served as a sort of model to use in keeping what I've written practical, of trying never to write anything without showing how it is used. And I must back-up further and thank the folks at Trax Corporation in Lynchburg, Virginia for their invitation to come

Preface

work for seven months with them in 1992 on power plant simulators. I learned lots about steam power plants and how they're controlled from this experience at Trax.

Since my arrival at Cal Poly in 1998 I have been involved in controls as often as possible. The course there was handed off to me by Mike Ianci, Ed Garner, and Ed Baker. They had built a very practical, hands-on lab. Though we've replaced much of the equipment in it, some is still left from those days, and much of what was added can be viewed as refinements and improvements of what they bequeathed to us. Here I have the pleasure of working with very practical, hand-on people like myself—John Ridgely, Charles Birdsong, Bill Murray, and Xi Wu. All have contributed one way or another to our work in making this laboratory more practical and hands-on. Some have had even to deal with the consequences of what I regarded as a good idea at the time, that required a lot of work on their parts to implement, to work the bugs out. Through their efforts we have a top-notch controls lab. I have seen a better one nowhere, neither in America nor in Europe. It was their sweat that made this lab the great teaching tool that it has become.

My two sojourns in Germany were important contributors to this book. I had the pleasure of working with several accomplished controls engineers there. Two stand out. Manfred Schuster at the Munich University of Applied Sciences welcomed me into his lab and gave me a very concise script to teach out of. His script is, in fact, a model for mine. At the Karlsruhe University of Applied Sciences I had the pleasure of working with Helmut Scherf, a committed controls nerd. Helmut is a rare, rare example of a practical controls engineer. He has published a great book in German of Simulink models of very common, practical systems. He has built and is still building practical, low-cost systems for his controls lab that serve as useful platforms for turning on the controls light in students' heads. Many of his perceptive, cut-to-the-quick methods of thinking about controls topics have been incorporated into this script.

So that's the story in brief of how this book came about. I hope that you enjoy it and find it useful.

Frank Owen

San Luis Obispo, California, U.S.A.

May 2012

Chapter 1 – Introduction to Control Systems

Goals

The purpose of this chapter is to give you an overview of the topic of control systems and to introduce you to the basic concepts that you need to go forward. Presented are

- Basic control loop anatomy, the parts and pieces of control loops and how they are configured
- Positioners vs. regulators, the two basic types of control loops
- A fly-by-wire system vs. a cruise control system, iconic examples of the positioner and the regulator
- A beginning discussion of block diagrams
- PID controllers, the most commonly used controllers in industry

Examples of control systems used in industry

Control theory is a relatively new field in engineering when compared with core topics, such as statics, dynamics, thermodynamics, etc. Early examples of control systems were developed actually before the science was fully understood. For example the fly-ball governor developed by James Watt to control overspeed of his steam engine was developed out of necessity, long before the science of controls came into being. Figure 1.1 shows an example of this controller. The fly-balls are mounted on a shaft that turns and is driven by the engine through the pulley shown. As the engine speeds up, the fly-balls are flung outward by their centrifugal force. This outward movement pulls the lever arm down, which raises its other end. This is tied to the steam inlet valve, which closes as the flyball weights move further outward. So if the engine tries to run away, the inlet steam valve will close, shutting off the fluid driving the engine.

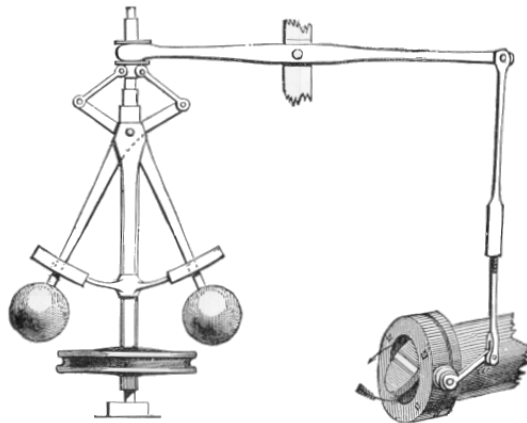


Figure 1.1 – Flyball governor

Many say that the development of the airplane by the Wright brothers was enabled by their understanding of controls--that and the development of a light-weight engine powerful enough to

Chapter 1 – Introduction to Control Systems

propel their machine into the air. Their development of wing warping enabled them to steer their airplane, something that had been impossible up to that point. And it is certainly true that much of control theory grew up with the airplane, as airplanes were developed during the two world wars and also throughout the 20th century for civilian purposes. As jet engines were developed and airplanes became bigger, it became ever more problematic to pilot an aircraft with just mechanical connections between the pilot's controls in the cockpit and the surfaces elsewhere on the airplane that steer it through the air. Thus the *fly-by-wire* system was developed, which cut this direct connection between the cockpit controls and the control surfaces on the airplane. In a fly-by-wire system, the movements of the stick or yoke and the rudder pedals in the cockpit are merely sensed by sensors. Electrical signals are then sent to actuators driving the appropriate surfaces, and then these move the ailerons, the elevator, or the rudder to steer the plane according to the control inputs made by the pilot. Of course the force applied by the actuator on the control surface can be many times what a human could apply directly. And the force can be applied at an actuator far distant from the pilot. So fly-by-wire brings with it the advantage of force amplification and remote control.

In industry one finds control systems of many types. In a refinery, chemical plant, food processing plant, or a power generation facility one finds control loops for controlling tank levels, pressures of fluids at various places in a plant, power output, valve position, pump, fan, or turbine speed. Modern-day fighter jets actually are designed to be unstable. This allows them to maneuver quickly. They can only fly because a control system stabilizes their flight, making corrections at a speed that no pilot could match. If one of these plane's control system failed in flight, the plane would be unflyable and would crash. There has been a tremendous growth of control system use in the modern automobile. There are even now drive-by-wire and brake-by-wire systems, where, like in the airplane, the direct mechanical or hydraulic connection between input devices and what they control has been cut and replaced by wish-sensing devices and then transmission of an electrical signal to an actuator to turn the wheels or to apply the brakes. Like the control of the unstable airplane, skid detection and control take advantage of an automatic control system's speed. A driver who loses control of his/her car may be saved by such a system. It springs automatically into action upon detecting a skid situation and applies the correct braking forces to rescue the car from the skid...and it does this before the driver is even aware that a problem exists.

Besides these applications, control theory is useful even for analyzing manually controlled systems. A human operator is in this case actually playing the role of the controller. A human's sensing of and reacting to inputs while manually controlling an industrial system or a vehicle is actually a study in controls. His or her reaction times, the force feedback or the angular travel of a steering wheel or an operating lever, such human-machine issues are within the realm of control theory. Mathematical models of the human controller have even been developed, so that a dynamic model of a manually operated system can be completed and studied.

Much of what has been discussed here can be illustrated with the example of a pilot in an airplane. Take the case of an airplane without a fly-by-wire system, with direct connections via cables and pulleys between the cockpit controls and the control surfaces, as one finds in a small, general-aviation airplane. A controls expert might study the effect of a human controller during some flight maneuver or critical

Chapter 1 – Introduction to Control Systems

situation. This system is purely mechanical with a human controller in the loop. But some small planes also have autopilots, so a controls engineer had to design a system that would sense flight conditions and operate the controls without intervention by the pilot. Even more complicated is the case of a larger airplane with a fly-by-wire system. Controls engineers designed the sensing-and-reaction link between the cockpit controls and the corresponding motion of the control surfaces. Even when the plane is being flown in manual mode, a sophisticated control system is engaged simply to sense the pilot's movement of the input control levers and pedals and transmit the commanded motion to actuators that will bring it about. Now consider the case of a fly-by-wire system with an active autopilot. The control system senses flight conditions—altitude, heading, and speed—and automatically operates the proper actuators—elevator, ailerons and rudder, and throttle—to maintain desired values. Thus these four variants of doing the same task—flying an airplane—show increasingly complex examples of modern control systems.

Basic control system anatomy

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 1.2).

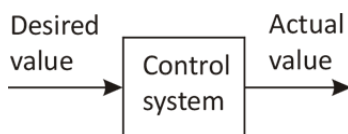


Figure 1.2 – SISO control system

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 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.

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, one quickly loses sight of what is going on or why one is doing what one is doing. I like to refer to 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

Chapter 1 – Introduction to Control Systems

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 1.3).

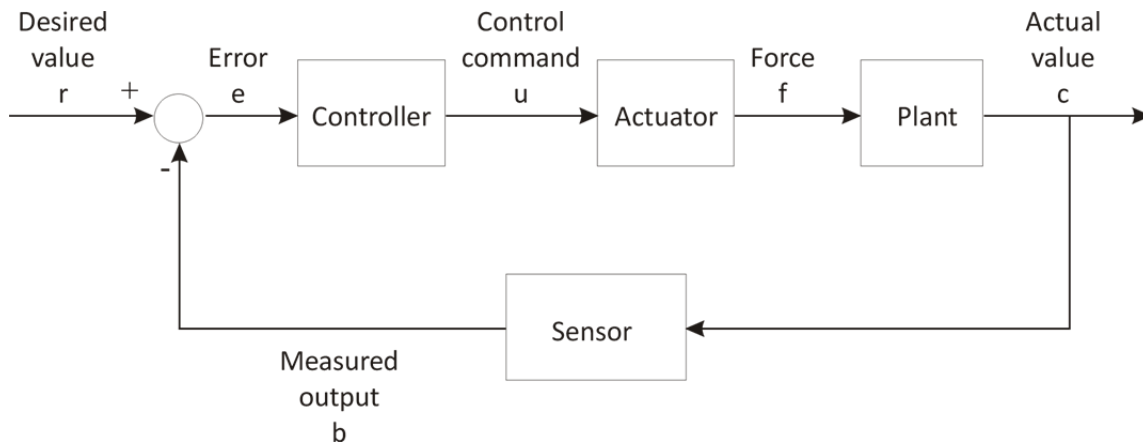


Figure 1.3 – Basic control loop anatomy

The five 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. I'll present them 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. The comparator is just the summing block that takes as input the desired value and the measured 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 interior temperature 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 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.

Chapter 1 – Introduction to Control Systems

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 this dilemma with experience. So 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 2 that the *signals* between the blocks also have names:

1. r – desired value (" r " stands for *reference value*; this is also known as the controller *setpoint*)
2. e – error
3. u – command
4. f – force
5. c – actual value (" c " stands for *controlled value*)
6. 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.

Two types of control loops: positioner and regulator

Control loops come in two flavors—positioners (also known as trackers) and regulators. Both are made up of the same components presented above. What differentiates them is actually how they are used, what their purpose is. The control loop shown in Figure 1.3 is a positioner. This loop can be modified to

Chapter 1 – Introduction to Control Systems

configure a regulator, as shown in Figure 1.4. Note that the difference is the addition of the *disturbance* between the actuator and the plant. This hints at the difference between the two loops. A positioner has a desired value that changes often. A user is operating a plant through the control system. It is the job of the control system to sense the operator's wishes and drive the plant to the point that the user desires. In contrast, in a regulator system, the user wants normally that the actual value stay at some preselected level, even though external influences are working to drive the system off of the preselected level. A good example is the cruise control system of a car. You set the desired speed to a fixed value. But upward and downward grades tend to make the speed deviate from its desired value. It is the job of the control system to keep the system at a preselected speed in the face of disturbances that tend to deflect the actual speed from this wished-for speed.

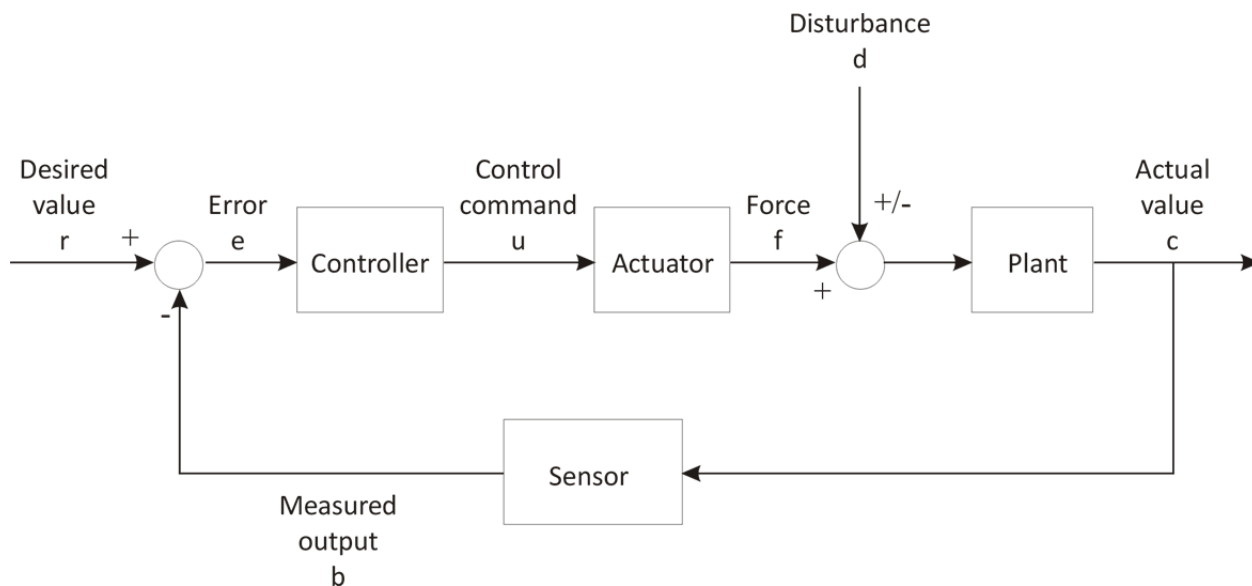


Figure 1.4 – Regulator loop

It is common to arrange control loops so that the input is on the left-hand side and the output is on the right-hand side. With a regulator loop, when the desired operating level is chosen, the loop selects this level as the reference level and considers it to be 0. The loop works in deviations from this operating level. We shall see later how this is done. At present it suffices to note that the desired value is the 0 reference value, so the r input can be eliminated. The loop can be reconfigured as shown in Figure 1.5. Here the input is the disturbance, and the output is still the same variable of interest, c . The reworking of loops as shown in this example is commonly done in controls and is known as *block-diagram algebra*. We shall see many more examples of this in the material to come.

Chapter 1 – Introduction to Control Systems

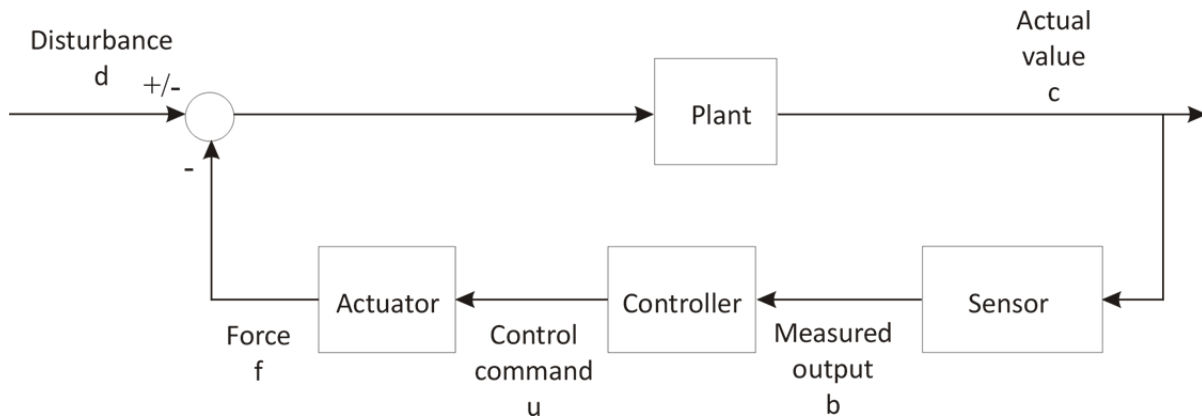


Figure 1.5 – Reworked regulator loop

PID controllers, the workhorse of the industry

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 1.6.

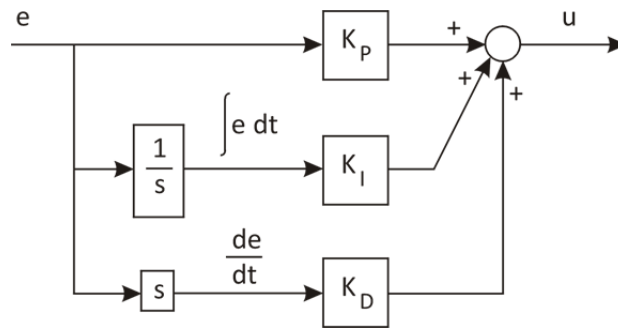


Figure 1.6 – PID controller configuration

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. The output of the integrator 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 derivative action is the third path. Note that the error is first differentiated. The output of the derivative 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.

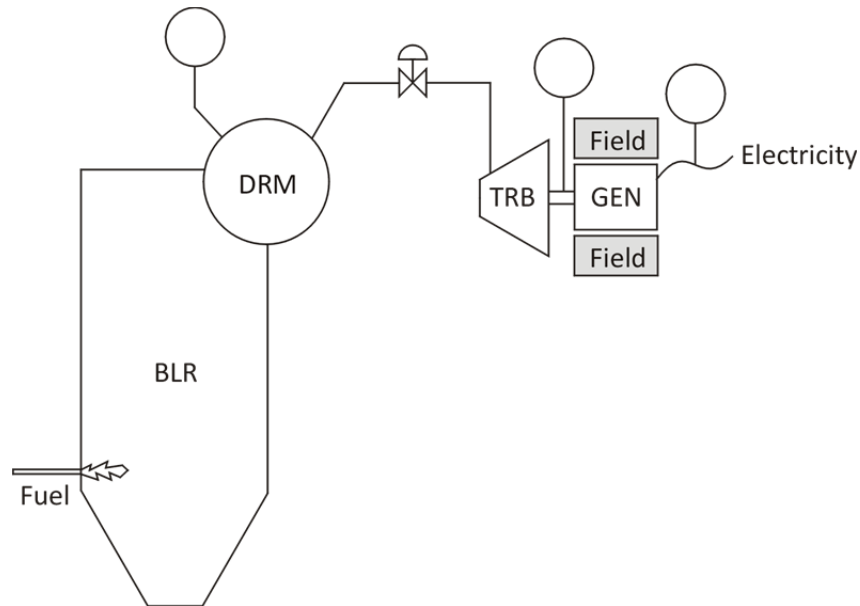
Chapter 1 – Introduction to Control Systems

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 and $K_D = 0$. We call these *P-only* 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.

Problems

- 1.1 Make a conceptual model of a brake-by-wire system. The force on the brake pedal is sensed as a desired braking force. The greater the pedal force, the greater the force applied by the brake pads to the brake discs. This measured pedal force is sensed by a load cell, which produces a voltage proportional to this force. This voltage is delivered to an interface board, which converts it into a digital number in a microprocessor. The controller running in the microprocessor produces an output signal that is then converted into a voltage that drives an electromechanical actuator. This drives a piston, the master cylinder, and produces a pressure. The pressure works on the brake piston that applies the braking force to the disc pads. The braking force is measured using actually a pressure sensor. Knowing the size of the brake pads, the braking force can be determined. The force applied to the brake pads is not necessarily the same force applied to the pedal, but they are proportionally related. Make a block diagram of this system, showing how all the components fit together to compose the system. Each block should all contain the name of a system component. Each line between the blocks should show the type of signal being transmitted between blocks.
- 1.2 The figure below shows a part of the electrical power generation system in a conventional steam power plant. Fuel (gas, oil, or coal) is supplied on the left-hand side to the boiler. There water is heated into steam and stored in a steam drum. From there the steam flows through a control valve into the turbine, which turns the plant's generator. The generator produces electric power. There are several SISO loops in play here. One loads the generator by increasing or decreasing its electric field. Of course when more electricity is needed, more fuel will be needed to support this. But the connection between fuel in and electricity out is not direct. The three "lollipops" shown represent measured quantities. Think about what these might be. Then write in the blocks the quantities that are measured. Draw dashed lines from these lollipops to the actuators that control them. It is a chain of events that lead from more power required to more fuel supplied. Consider the case of a desired increase in power out. Write out in words the sequence of cause-and-effect events that will lead the steam plant to a new, higher level of operation. Make a copy of the completed diagram to complete the deliverable for this problem.

Chapter 1 – Introduction to Control Systems



- 1.3 Sometimes a plant is a two-part plant, and a disturbance enters the plant midway between these two parts. Draw a regulator loop for such a plant with a negative disturbance entering between Plant 1 and plant 2. Let the disturbance be the input to the loop.
- 1.4 Create a Simulink model of a PID controller. For this you will need to use *gain blocks* for the three controller gains. Use an integrator block and a differentiator block for the integral action and the derivative action. At first just set all gains to a value of 1. As input, use a constant block and set its value to 0. All three control actions are summed with a sum block. Use a scope block at the output of the sum block to capture what output the controller delivers over time. Also place scope blocks on all three control actions to see how they behave. Of course in the real world the controller would be hooked into a system and receive the system error as its input. Its output would be fed downstream to an actuator. With 0 as a steady input, you have modeled the case of a control loop where the actual value is equal to the desired value. What should the controller do and what does it do? Now make the input 1. What does the controller do now?

Chapter 9 – Designing and tuning PID controllers

Goals

- Provide a practical look at the PID, at how it is thought of and understood by practitioners in industry
- Describe several heuristic tuning methods for PID controllers

In Chapters 7 and 8 we have already gotten a good look at the PID controller. In the context of root-locus and frequency-response design procedures we have undertaken the design of PID controllers for various common systems. In a way these approaches give one a false impression. They imply that one must have a system model to tune a PID controller. That is not, however, common practice in many, if not most, industrial applications. For example a tank level controller needs to be implemented. One purchases a commercial PID controller and puts it in service on the tank. One then field tunes the controller, starting with the proportional gain and then adding integral and derivative gain to improve system response. All of this is normally done without a system model.

Indeed, the normal understanding of PID control by a controls technician in a normal plant is much more intuitive and down-to-earth than what you have learned about PID design and tuning via root locus and frequency response. Those two technics are powerful design techniques and ought to be understood by controls technicians...but often they are not. Thus for the controls engineer, it is important also to gain this intuitive grasp of PID control simply to be able to communicate effectively with plant controls technicians. This common-sense understanding of PID control offers yet another perspective on this technology, and this complements the approaches taken already with root locus and frequency response.

There are a number of accepted non-model-based methods for tuning PID controllers—i.e. methods used for field tuning. The most well-known of these is probably the Ziegler-Nichols tuning method. That and others are discussed in this chapter.

The PID controller interface

Figure 9.1 shows the faceplate of a typical PID industrial controller, this one used for temperature control in a kiln or heat-treatment oven.

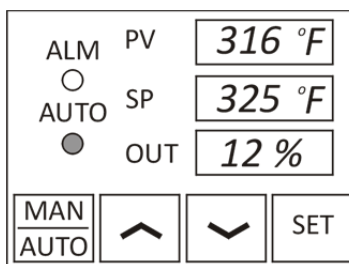


Figure 9.1 – Industrial PID controller faceplate

Chapter 9 – Designing and tuning PID controllers

The interface is somewhat sparse because often there are many of these controllers grouped together, controlling various parts of an industrial process.

The user input panel for this controller consists of the four buttons at the bottom of the faceplate. The first button is the manual/automatic button, used to switch between these two modes of operation. The AUTO light on the left of the faceplate indicates whether the controller is in automatic mode. In manual mode the control function is deactivated. The user can drive the process in manual mode by using the up and down arrows on the center two buttons. This adjusts the output from the controller directly. This output from the controller is some (settable) range. Often the control output is calculated and displayed as percentage of this range, as is shown above in the OUT window. In manual mode, when one clicks the up arrow, for example, the percentage output would increase. To run the controller in automatic mode, one must set the setpoint (and maybe some other parameters, like K_p , T_i , and T_D). To do this, one must read the documentation that comes with the controller. In the case above, the SET button is used to enter a parameter setting procedure to set the controller parameters for automatic operation.

In both manual and automatic operation, the display of the PV (process variable) shows the output coming from the sensor of the variable that is the output from the loop, the variable to be controlled. SP (setpoint) is the desired value of this output variable. This is the input to the control loop. In the above example, the controller is in automatic mode (the AUTO light is lit). The actual value of the controlled process is 316 °F, and the desired value is 325 °F. The controller is putting out 12% of its output range to bring the actual temperature up to the desired value.

The ALM light is an alarm light. This can be set to indicate that the actual value is outside a certain range around the desired value. Often alarms can be set at two levels, a warning level and a severe level. The warning level will have the light burn amber. The severe level will have it burn red. If the AUTO light is green, this green-yellow-red color scheme for the lights will allow an operator to scan a group of these modules quickly and determine

1. which loops are in automatic operation
2. whether there are any warning-level alarms active
3. whether there are any severe-level alarms active

Quarter-cycle damping

Controllers are “designed” to improve system response or to achieve a desired, prescribed response. Various aims are possible:

- Limit a step response to a specified overshoot. Recall that the overshoot is a function solely

$$\text{of } \zeta \left(\%OS = e^{-\left(\frac{\zeta \cdot \pi}{\sqrt{1-\zeta^2}}\right)} \times 100\% \right). \text{ At } \zeta = 1 \text{ there is no overshoot. But by specifying no}$$

Chapter 9 – Designing and tuning PID controllers

overshoot, one must settle for a longer response, a longer time to reach a new setpoint. So often a compromise is made and a little overshoot is accepted for faster reaction speed.

- Specify a specific frequency of oscillation. Note that increasing this frequency decreases the system's speed of response, since the response is often just the first half to three-quarter wave cycle of the response sinusoid.
- Limit or eliminate steady-state error.
- A combination of such specifications.

To meet these needs it is very helpful to remember the geometry especially of complex poles and the meaning of various distances on this plot. Figure 9.2 shows this geometry again.

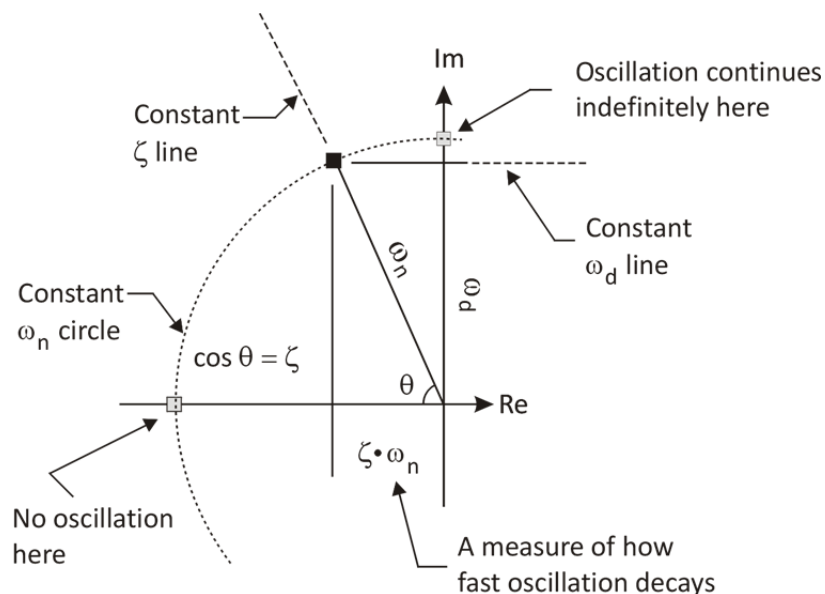


Figure 9.2 – Geometry of complex (oscillating) poles

So by specifying a certain overshoot, one is limiting the dominant closed-loop pole pair to a ray leading from the origin at a certain angle $\theta (= \arccos \zeta)$ from the negative real axis. As explained above, often a controller is set to give a little overshoot (5-10%) in order to have the system respond faster.

Alternatively there is a concept called *quarter-cycle damping*, whereby each oscillation is 1/4 the amplitude of the previous oscillation (see Figure 9.3). For quarter-cycle damping, ζ turns out to be 0.2155. This leads to shorter response times but more overshoot. Which to pick depends on the application and the ability to tolerate overshoot. If the oscillation frequency is specified, then the vertical distance from the real axis, ω_d , is known. If both the desired or tolerable overshoot and the damping ratio are known, then the desired location of the dominant closed-loop pole pair is fixed. Steady-state error is not readily seen on the plot, so no such statements can be made regarding it and its placement of poles on the plot above.

Chapter 9 – Designing and tuning PID controllers

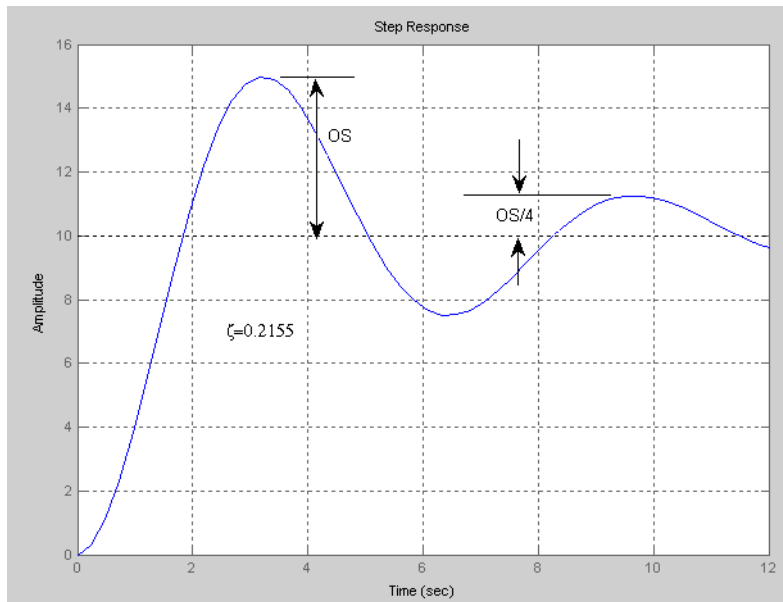


Figure 9.3 – Quarter-cycle damping

The PID controller

As explained in Chapter 1 the PID controller is the workhorse of industry. Most, indeed almost all, control loops in industry are SISO loops with PID controllers. So to be active as a control engineer in industry one must have a good understanding of PID controllers. And by the same token, with a good understanding of PID controllers, one can do almost anything one wants in industry. Even in the rare case that one needs something other than a PID controller, a good understanding of this workhorse controller will stand one in good stead to compare an exotic controller with a conventional PID controller.

Before beginning with the explanation of a PID controller, it is first useful to recall where the controller is placed in the control loop and what its input and output are. The controller is just after the comparator, the summing block that takes the difference between the desired value and the actual value. Thus the input to the controller is the error signal. The controller operates on this error signal and produces a command that is then sent downstream to the actuator. The purpose of the control loop is to drive the error to 0, so that the actual value = the desired value. If everything is working as it should, e will be 0 and the controller will take no action. It will simply put 0 on the output, the input to the actuator. This is a command to the actuator to do nothing. When the actual value is not equal to the desired value, the controller takes action and produces a non-zero command for the actuator.

PID, of course, stands for proportion-integral-derivative. A PID controller has a parallel structure with these three actions (see Figure 9.4). The three controller constants— K_p , K_I , and K_D —can be tuned to adjust the relative strength of each action. The proportional action is the main action, and the other two actions are add-ons to improve the control. Often one sees a P-only controller, i.e. a controller with only $K_p \neq 0$. Often one sees a PID controller with one action—integral or derivative—turned off. Thus the

Chapter 9 – Designing and tuning PID controllers

most common variants are a P controller, a PI controller, a PD controller, or the full-blown PID controller.

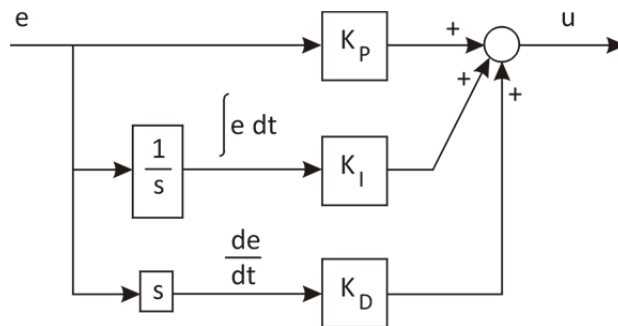


Figure 9.4 – PID controller structure

Of course one might ask, how did anyone ever come up with this structure? The answer lies in the development of the controller. It was developed by a Russian-American engineer Nicholas Minorsky. He was working on an auto-steering system for ships for the U.S. Navy. Minorsky observed helmsmen steering ships and noticed that they acted not only on the error itself but also on the rate at which the error was developing and on the history of the past error. The PID structure was developed from these observations. (See the Wikipedia article on PID controllers.)

Proportional action

As state above, P-action is the main action. One almost never sees a PID controller without P-action. P-action makes a lot of sense. The controller sees the error as input. If the error is small, the controller should suggest a small action, a nudge, to the actuator to get the plant back on track and reduce the error to 0. If the error is large, that means that the actual value has drifted far away from the desired value. The controller needs to suggest a large action to the actuator to bring the plant in line with the desired value. Such a strategy means the controller action should be proportional to the error, i.e. P-action.

The proportional gain, K_p , can also be regarded as the sensitivity of the controller, how great an action it will suggest for a given deviation of actual from desired. If K_p is high, the controller will take a large action for a small deviation of actual from desired. The controller is very sensitive. If K_p is small, the reaction of the controller to a deviation of actual from desired is gentle. A very common-sense, seat-of-the-pants method for tuning K_p is the following. One looks at the actuator and asks, “How much can the actuator give?” For example, take a tank-level loop with the controlling inlet valve operating at a 75% valve opening at design steady state. (See Chapter 3 for the tank-level system description.) The valve has 25% yet to give if the level starts sinking, 75% to close down if the level starts to rise. One would pick the situation that had the most dire consequences to continue—either an overflowing tank or an empty tank—and then pose the question, “How far away from design steady state do I want to go before the valve is giving all it’s got to bring the situation under control?” If the most critical situation is the overflowing tank and one decides to allow the tank level to rise three inches before having the valve give its full 75% to prevent a further rise, then one has the proportional gain. If the tank’s actual level

Chapter 9 – Designing and tuning PID controllers

rises 3 inches above the desired steady state, then $e = -3$ inches. The designer has decided that at this deviation the valve should have closed its 75%, so the controller output should be -75% (recall that such loops operate on deviations from design steady state). So $K_p = 25$ % valve opening/inch. This is a common-sense way to select K_p and often a good way to pick an initial K_p . Regardless of how K_p is selected, it is good practice always to perform a common-sense check on it, to see how much of a deviation will drive the actuator to *saturation* (see below).

Proportional action is based upon the current value of error. Thus it is based upon the present. As we shall see below, integral action is based on the past, and derivative action on the future.

Integral action

Integral action is used to get rid of steady-state error. A system's type is not great enough, so one uses a PI controller to add a free integrator to the open-loop transfer function. Otherwise expressed, we have seen in Chapter 6 that P-only control leads naturally to steady-state error in many cases. The integral term acts not on the error itself but rather on the integral of the error—that is, on the accumulated error produced over time.

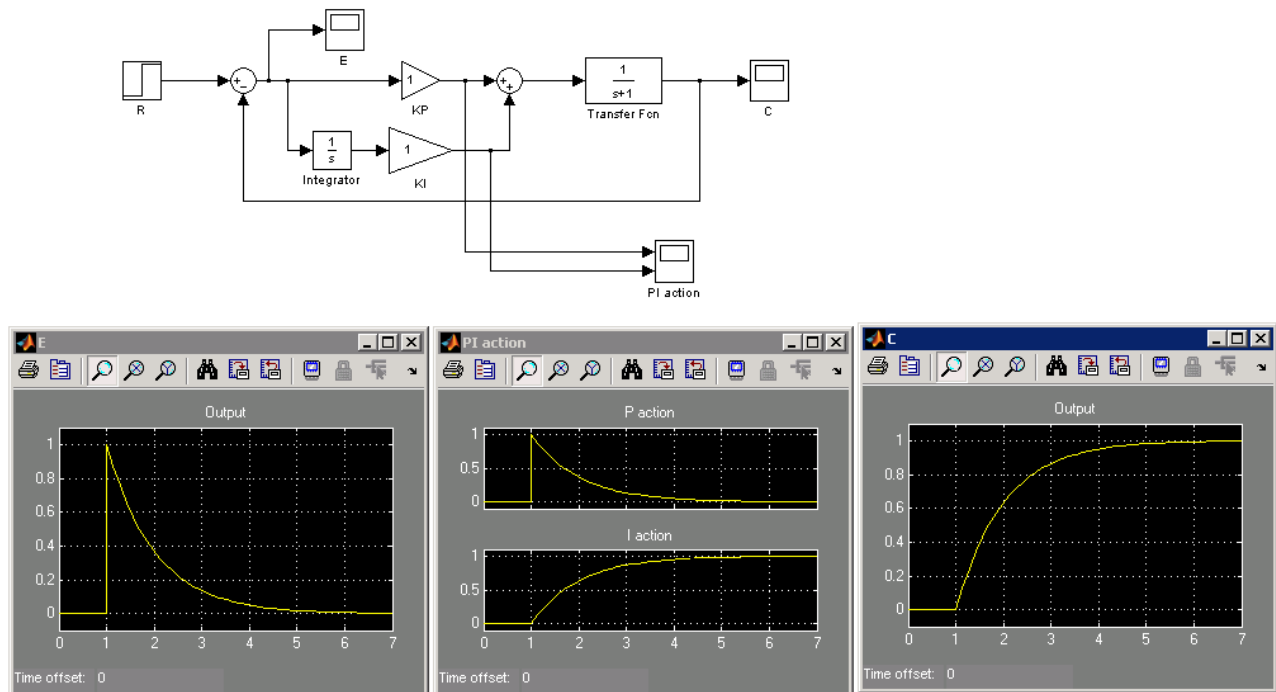


Figure 9.4 – PI control loop

It is useful to look at this action up close. Figure 9.4 shows a PI controller in a loop with a first-order actuator+plant. A unit step is given to the input. Of course, without integral control, the output would not go to 1, the commanded value. There would be steady-state error. (Try this by setting $K_I = 0$ in the model above.) Notice the two actions over time. When the step is input at $t = 1$ sec, the error immediately goes to $K_p \cdot e$. The integral action is 0 because this action is based upon the area under the $e(t)$ curve, and no time has passed to allow the area under the curve to accumulate. As the clock ticks,

Chapter 9 – Designing and tuning PID controllers

this area develops, and the I-action increases. As the plant moves closer to the desired value, the P-action diminishes. In the end, with the plant at the desired value, the P-action is 0 and all non-zero control action is provided by the integral control action.

Noteworthy is that the integral action will continue to accumulate until the error is 0. Once the error is 0, there is no proportional action. Note that with $e = 0$, the integral action is not 0; it just doesn't accumulate any more.

Controls engineers in the field often use the term *reset* to characterize the strength of integral action. Reset is also called *integral time*, T_I . (In the process industry, this variable is often expressed as R . This is not the same R used for the input or reference value in the standard control loop.) This is a comparison of integral action with proportional action. If a PI controller were subjected to a steady input signal, the *reset time* is the time it would take for the integral action to reach the level of the proportional action. So with an input signal of 1, the proportional action would be K_p . The integral action would be

$$K_I \cdot \int 1 \cdot dt = K_I \cdot t$$

At $t = T_I$, $K_p = K_I \cdot T_I$. So $T_I = K_p / K_I$. Since T_I is a measure of how long it takes the integral action to develop, the higher T_I is, the lower K_I is and the more gentle the integral action.

Since integral action accumulates over time, what is important to it is the history of error over time. Integral action “remembers” what has happened, so it is an action based upon past experience.

Derivative action

Derivative control is predictive in nature. It operates on the rate of change of error, not on the error itself. A practical example illustrates this. In Figure 9.5 a supply tank has a level controller on it. As the liquid in the tank is needed, an outlet valve opens to supply more to a downstream process. If the tank starts off at a steady state and then a sudden demand for more liquid downstream occurs, the outlet valve opens quickly, and the level in the tank starts to drop. The trouble with proportional control is that it really does not react until the error has developed. So there is a lag time before the proportional action comes into play, since it can only react to current error, error that has already occurred. This is unfortunate, because, with the valve open and the error developing rapidly, it was entirely predictable that the error would develop before it did develop. The idea behind derivative control lies in this phrase “with the error developing rapidly”. The rate of change of error is high. That means the error is developing rapidly, and there is no reason why the controller shouldn't react to that before the error develops. So a strong derivative action at the start, before the error develops, ensures that less error will actually develop.

Chapter 9 – Designing and tuning PID controllers

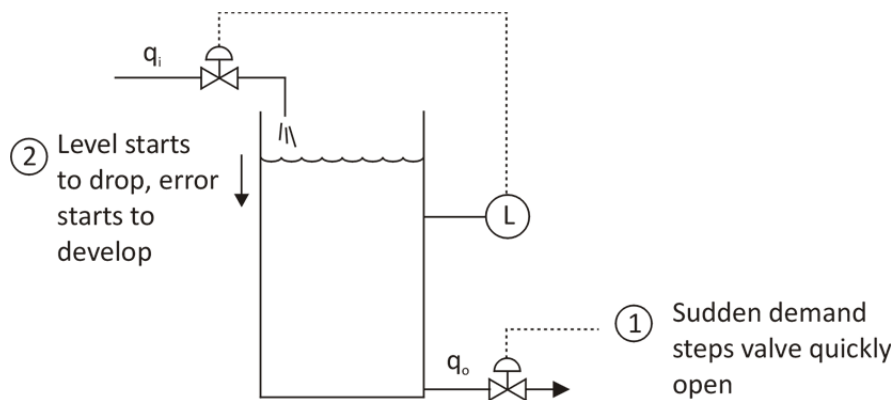


Figure 9.5 – Level regulator experiencing sudden increase in demand

Another scenario would be the following. You are driving a car in manual speed control (the cruise control is not engaged) at a steady speed. You come to a steep incline. You automatically push the accelerator a little harder, because you know that if you don't, the car's speed will decrease. You have just applied derivative control. Through visual clues, the steep incline, you know that the speed will decrease, so you have taken a proactive step to prevent the speed from decreasing before it actually decreased. Not doing this, using only proportional action, would play out like this. You are driving at a steady speed, encounter the incline, and at first do nothing different. You apply the same constant force to the accelerator. After a few moments, you notice that the speed has decreased, so you push the accelerator harder to compensate for the decreased speed. The first scenario seems entirely plausible. The second seems like the driving style of an inexperienced driver in maintaining a constant speed. So derivative control is not something exotic or bizarre. It fits naturally into normal human behavior.

Chapter 9 – Designing and tuning PID controllers

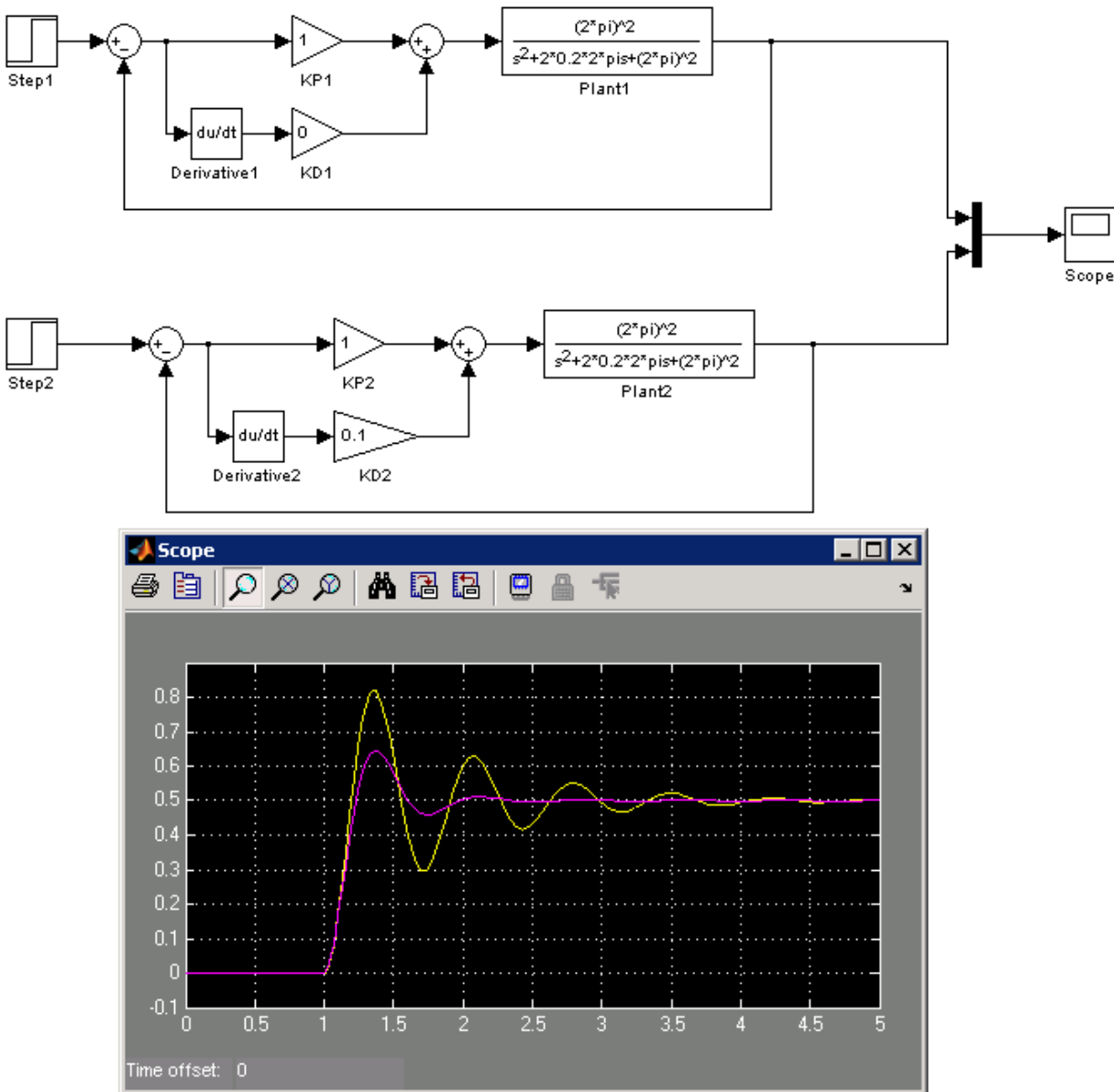


Figure 9.6 – Comparison of P and PD control

Figure 9.6 shows a comparison of P and PD control applied to an underdamped second order plant+actuator subjected to a unit step input. Notice that the system, under P control, oscillates much more vigorously than it does under PD control.

Figure 9.7 shows the separate control actions from this same system. With a unit step input, the error immediately goes to 1. So the proportional action goes to K_p . The sudden increase in error generated by the step input of course produces an infinite rate of error increase. So the initial derivative action is very large. This is known in industry as a *derivative kick*. This is the large spike in Figure 9.7. Also noteworthy about these actions is that the proportional action endures because of the constant steady-

Chapter 9 – Designing and tuning PID controllers

state error. So even though the error is non-zero at the end, it is not changing. And since it is not changing, its rate of change is 0, so the derivative action is 0.

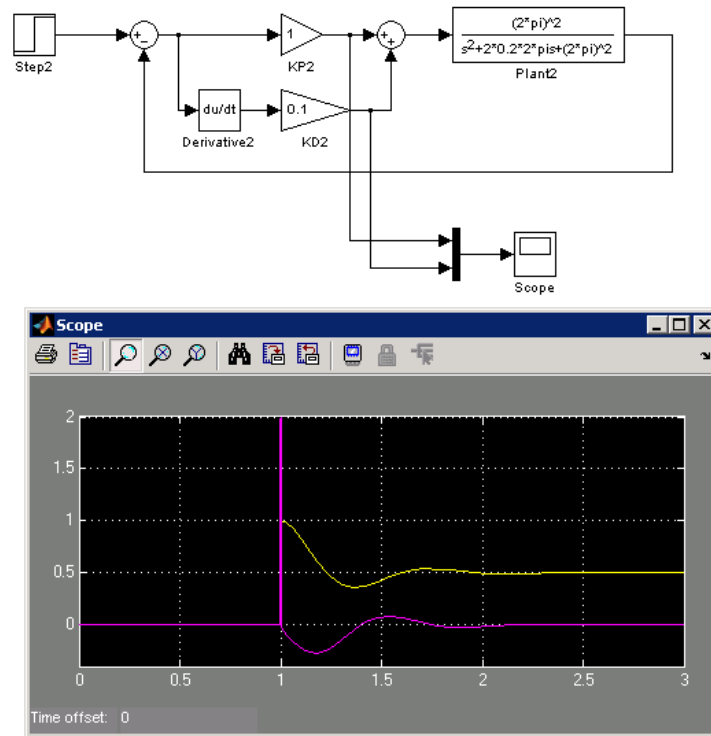


Figure 9.7 – Proportional and derivative actions

In a second common form of the PID controller, the variable K_D is not used. Rather the strength of the derivative action is expressed by the *derivative time*, T_D (see below). In the process industry, this PID parameter is given the symbol T .

The full-blown PID

A full-blown PID contains the three individual elements. The higher the error, the greater the proportional action. But we have seen in Chapter 6 that steady-state error is something that often proportional control action alone cannot get rid of. Integral action is added to do this. It continues to act until the error becomes 0. Derivative action allows the controller to respond in advance. It sees the error coming and acts to stop the growth of error before it occurs.

This is the common-sense, intuitive understanding of PID control that most controls practitioners have who work up-close, hands-on with control loops. Talking to a plant's controls technician in terms of root locus or Bode plots is usually not very productive. That does not mean that these techniques should not be used in a plant. They should be. They are not as well known in the field as they should be. They enhance one's knowledge of system dynamics and what is actually happening inside control loops. But the understanding of controls technology and the language used when talking about it in the field is more intuitive and hands-on than it is theoretical.

Chapter 9 – Designing and tuning PID controllers

Forms of the PID controller

The PID control structure can be expressed in a number of different forms. Three common forms are given in Table 9.1. Notice that each control structure has three parameters that describe it. That means that each set of parameters can be expressed in terms of a different set. Notice that in the first form, K_P is a separate action, with no impact on the other two actions. In the second two forms, the gain K_{PID} is a multiplier for all actions. The K - T_1 - T_2 -form is the form convenient to use for pole cancellation, where the poles to be cancelled are at $1/T_1$ and $1/T_2$.

Form	Block diagram	Transfer function
K_P - K_I - K_D		$G_{PID}(s) = K_P + \frac{K_I}{s} + K_D \cdot s$ $= \frac{K_D \cdot s^2 + K_P \cdot s + K_I}{s}$
K_{PID} - T_I - T_D		$G_{PID}(s) = K_{PID} \left(1 + \frac{1}{T_I \cdot s} + T_D \cdot s \right)$
K - T_1 - T_2		$G_{PID}(s) = \frac{K \cdot (T_1 \cdot s + 1) \cdot (T_2 \cdot s + 1)}{s}$

Table 9.1 – Forms of PID controller

Actuator saturation

A practical problem that often is not evident in the world of Simulink models is that the size of a loop's actuator is limited. A controller may tell a power amplifier to put out 40 volts, but if that amplifier cannot put out more than 15 volts, it will *saturate*. Thus all command signals from the controller that demand more than 15 volts will effectively be chopped off. One solution to such a problem would be to buy and install a more powerful amplifier that could put out 40 volts. But this may not be the correct solution to this problem. For example, take a cruise control for a car. You may have it set at 65 mph and engage it when you are going 40 mph. This will cause the car to react and to try as hard as it can to get to 65 mph as fast as it can. But the engine can only put out a limited amount of power, so it may take some time for the car to react and arrive at the desired speed. During this period, when the car's engine is fully engaged to reach 65 mph, the speed-control loop is saturated. To fix this, one could buy a car with a more powerful engine. But this is expensive, and how important is it anyway to have a car accelerate from 40 mph to 65 mph very quickly? The control loop was designed to hold the car at a speed around 65 mph, not as an acceleration loop to get a car from the entrance ramp of a freeway onto the freeway. So saturation, even though it is a non-linear phenomenon, is not bad in all cases. In the example of the cruise control for the car, saturation when going to driving speed is not bad and can be tolerated. But if the engine of the car were so puny that it could not maintain the car's speed at

Chapter 9 – Designing and tuning PID controllers

relatively gentle includes, then one must ask the question of whether the car's engine was sized properly for the car.

The operating conditions of the loop must be taken into account when designing the actuator for the loop. What kind of load will cause the actuator to saturate must be considered, and the actuator should be sized accordingly for the loop operation conditions. Actuators can also be sized too large. Take the tank-level example that has appeared throughout this text. If the input valve is too large, it has too big a gain. If the tank level drops slightly and this valve opens a little, it lets in a flood of liquid so that the level rises quickly. Shutting just a little cuts off so much liquid that the tank level drops quickly. Thus a mis-sized, oversized valve leads to a fluctuating tank level and unstable or marginally stable operation in the process that it is a part of.

Saturation depends not just on the size of the actuator but also on how far the actual value is away from the desired value. This was seen above with the cruise control example. This same phenomenon applies also to motion control systems. They are normally designed for somewhat fine control, to maintain position around a narrow range of distance. For gross motions from point A to point B, motion actuators often saturate.

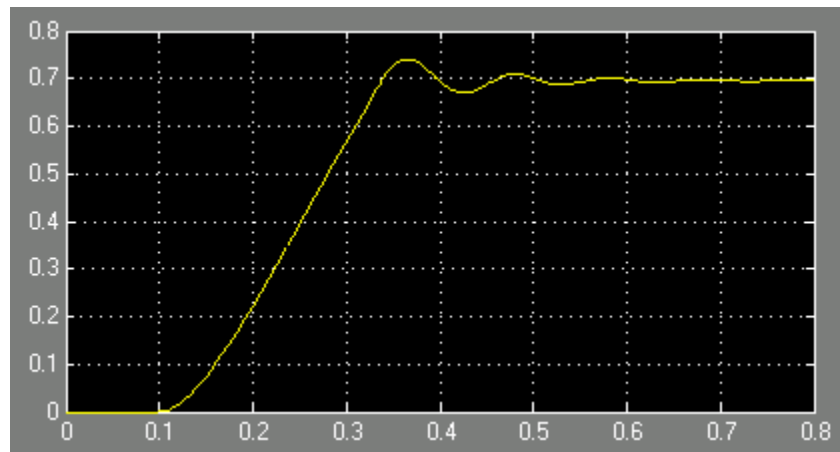


Figure 9.8 – Saturation response

Figure 9.8 shows what saturation looks like on a step response plot. The long, straight climb from 0 up to a new value does not seem to fit with the shape of the curvy sinusoidal oscillation once the system gets within a narrow range of the final value.

A problem with integral control—wind-up

A hidden problem with integral control lurks in the block diagram of a PID controller, in that one is not aware that integral action has associated with it a sort of storehouse of remembered past error that can continue to act, even after the loop reaches its desired value. Consider the example in Figure 9.8. If this system were under integral control, in the saturated state, between $t = 0.1$ sec and a little after 0.3 sec the persistent error would cause an accumulation of area under the $e(t)$ curve. As the system neared its goal and the error became less and less, the proportional term would start to decrease. But the integral

Chapter 9 – Designing and tuning PID controllers

action would continue to act because of the stored error under the error curve, even though the new setpoint had been reached. Thus the integral term would drive the system past the setpoint. In fact the only way for the integral term to shed its accumulated positive error would be for it to accumulate negative error. And the only way for it to do this would be to stay above the setpoint until the negative accumulated error cancelled the positive accumulated error. This tendency of the integral term to accumulate error, even when the actuator is doing all it can to reduce error, is called *integral wind-up*.

The solution to this problem is simple: don't let the area under the error curve accumulate. A switch is put into the integral branch of the PI or PID controller. If the actuator is saturated, turn off the integration of the error. So in the above example, while the actuator is saturated between 0.1 sec and a little over 0.3 sec, the integral term is turned off. When the system nears its desired value and the actuator becomes unsaturated, the integral term is once again turned on. This prevents the overshooting caused by integral wind-up and allows the system to reach its desired value with less fluctuations enroute.

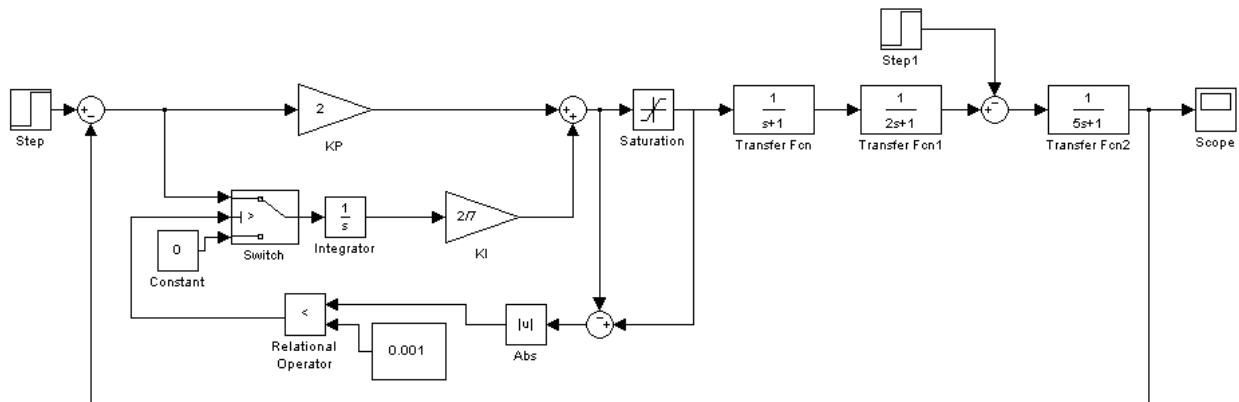


Figure 9.9 - Integrator anti-windup implementation

Figure 9.9 shows a Simulink implementation of an anti-windup scheme. The input and outputs of the saturation block are compared. If they are equal (or close), then the integral is allowed to accumulate error. If they are not, then the actuator is saturated and integral accumulation is turned off.

A problem with derivative control—noise

Figure 9.10 shows the output from a pressure sensor in a water tank. Notice that even though the tank level remains constant at about 17.5 inches, there is a lot of noise in the signal. This presents a problem for derivative action in a PID controller. Recall that the derivative term acts on the rate of change of error, not upon the error itself. With the signal below, the tank level, and thus the error, is not changing in the big picture over time. But the signal itself, when considering the noise, is constantly changing. The derivative of the error is the slope of the error curve. So in this case, even if the error is 0, the noise is constantly making the slope flip-flop up and down, often with a severe magnitude. Derivative action applied to such a signal causes the actuator to act first in one direction vigorously, then at the next

Chapter 9 – Designing and tuning PID controllers

moment in the other direction vigorously. The motor or valve or hydraulic cylinder that is being driven by this controller is being exercised back and forth and for no good reason.

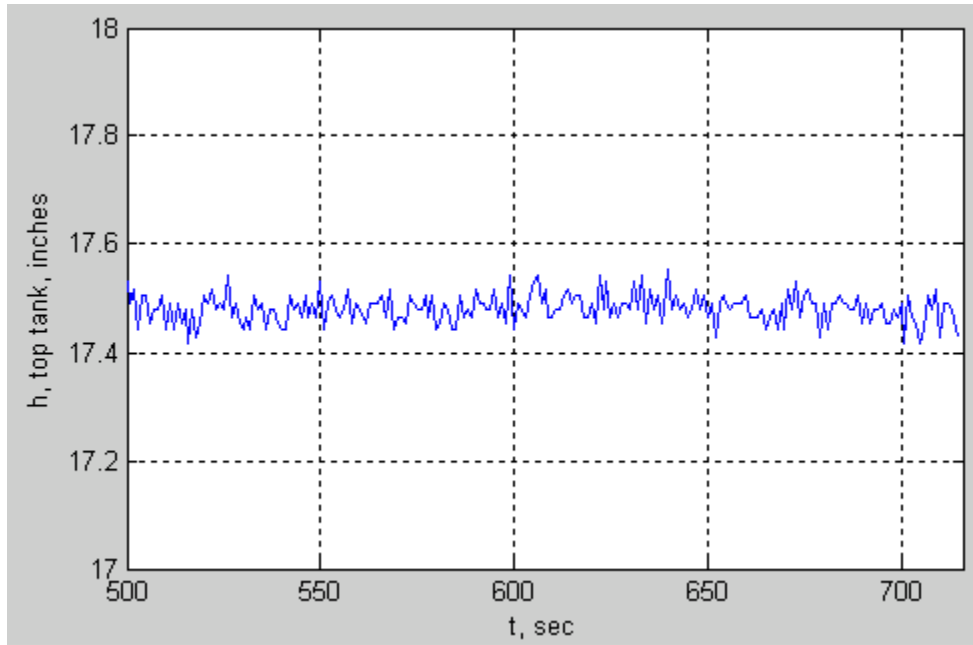


Figure 9.10 – Noise in tank level sensor

The solution to this is to clean the signal up. This can be done with either a hardware filter, a software filter, or with both. It is common to include a first-order filter in the derivative path of a PID controller to clean up the error signal before taking its derivative. The time constant of this filter, T_F , is often based on T_D , the derivative time. A common value for T_F is $T_F = T_D/10$. Figure 9.11 shows such an implementation of a PID controller with filtering.

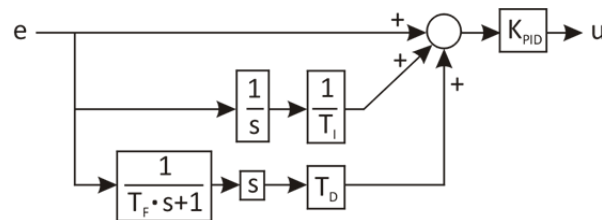


Figure 9.11 – PID controller with derivative filtering

Tuning methods for PID controllers

The next sections deal with different methods for tuning PID controllers. These reflect different strategies for tuning controllers. Most of them are heuristic or field tuning methods. There are literally thousands of schemes like this. The ones illustrated here are some of the better known schemes.

Tuning methods — Replacement of system's natural dynamics with desired dynamics

Chapter 9 – Designing and tuning PID controllers

The $K-T_1-T_2$ form of the PID controller suggests an interesting design strategy for a controller. One can cancel the dynamics of two poles of an open-loop system using the two zeros of the PID and then install one's own dynamics with the controller's remaining pole at the origin and with the controller gain.

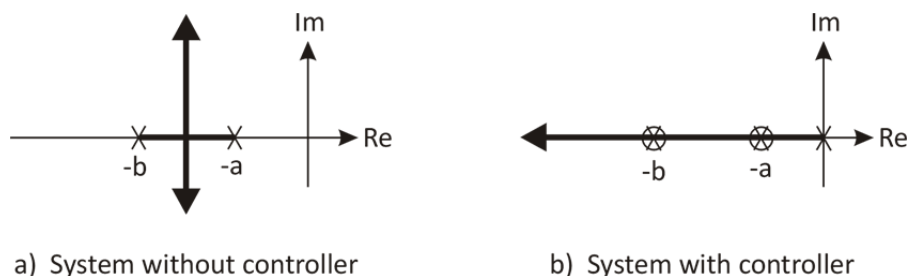


Figure 9.12 – PID pole cancellation

Figure 9.12 shows this for a simple open-loop system with two real, stable poles. With a P-only controller the system would eventually start oscillating as the gain is increased. If one cancels the poles at $-a$ and $-b$ by adding controller zeros there, the closed-loop system becomes a simple first-order that never oscillates as the gain is increased. Thus one has cancelled the original system's behavior and installed a behavior to one's own liking.

A big caveat to this strategy is that this really only works as described for systems whose models perfectly fit the system, and that is virtually never the case. Even if one is lucky or resourceful enough to have a perfect model, physical systems change over time with wear. So a system that has poles at $-a$ and $-b$ will see those poles drift over time. Thus the poles are not quite cancelled, and this changes the behavior of the system. The poles are not quite cancelled. This problem is always encountered with zero/pole cancellation. It is often recommended to place the zeros near the poles in working with this strategy to be explicit about the fact that the zeros will not always (ever?) exactly cancel the poles.

Tuning methods — Ziegler-Nichols tuning

The Ziegler-Nichols tuning algorithm was developed in the 1940s primarily for regulator control loops in the process industry (power generation station, chemical plants, refineries, etc.). As regulators, these loops' purpose is *disturbance rejection*, that is keeping a desired quantity at a certain level despite disturbing influences that try to change it. Ziegler-Nichols is probably the best known and most widely used of the heuristic tuning methods for tuning PID controllers. "Heuristic" simply means "based on experimentation" or "based on trial-and-error". Such methods do not depend on the development of a system model. They are field tuning methods, in that one can apply them to the real system and tune it in place.

In Ziegler-Nichols tuning, tuning parameters K_p , K_i , and K_D are based on K_u and P_u . K_u is the gain that causes a system with a P-only controller to be marginally stable. ("u" stands for "ultimate".) You can find the ultimate gain by a trial and error process. One sets K_p to some low value (K_i and K_D are 0 at this stage). Test the system with this K_p to see if it oscillates continuously (marginally stable). If the oscillations decay, keep increasing K_p . If the oscillations increase in amplitude (unstable system), reduce

Chapter 9 – Designing and tuning PID controllers

K_p . Do this until the system is marginally stable. When you arrive at this point, you have found K_u , the gain that got you there. P_u is the period of the non-decaying oscillations at this point of marginal stability.

Often you will not be able to reach a system's ultimate gain because the system actuator will saturate. This is a situation where the controller demands more of the actuator than it can provide. Remember the actuator gets a command input from the controller and sends a "force" signal to the plant. The actuator is limited in the amount of "force" it can send to the plant. For a valve/tank level-control system, the valve cannot open more than 100%, nor can it close more than 0%.

First step: Find K_u

Find K_u by increasing K_p until the system oscillates without a decay. While you are monitoring the loop output, monitor the actuator at the same time to see whether or not it is saturating. Continue to increase K_p until you find K_u or the actuator saturates.

If the actuator saturates, you will not be able to get K_u . In this case use another method to get K_q ("q" stands for quarter). Adjust the P-only gain until you have quarter-cycle damping. This is a measurement for a second-order, underdamped system. On the response plot look at the first two humps (the first hump is where the %OS is measured). K_q is the value of K_p that makes the height of the second hump 1/4 the height of the first hump. Use the final output value as the reference for measuring the hump heights. Now K_u can be determined from K_q . It is: $K_u = 2 \cdot K_q$.

Second step: Find P_u

P_u is the ultimate period of oscillation. You can find this out from the response plot with $K_p = K_u$, if you were able to find it. If you found K_u from K_q , get P_q . We assume that $P_u = P_q$. This is close enough.

Third step: Find controller gains

Now the suggested Ziegler Nichols settings for P, PI, and PID controllers are:

$$P: K_{PID} = 0.5 \cdot K_u$$

$$PI: K_{PID} = 0.45 \cdot K_u; T_I = 1.2 / P_u$$

$$PID: K_{PID} = 0.6 K_u; T_I = 2 / P_u; T_D = P_u / 8$$

Tuning methods — Astrom-Hagglund relay tuning method for Ziegler-Nichols tuning

Chapter 9 – Designing and tuning PID controllers

One of the problems with the Ziegler-Nichols tuning method is that one turns up the proportional gain to drive the system to the verge of instability. This can be a dangerous thing to do with a real system. To avoid this but still be able to calculate K_i and P_u , an alternative method was developed by Astrom and Hagglund. In this method the controller is replaced with a relay or two-point controller that puts out a positive constant if the error signal is negative and a negative constant if the error signal is positive. This controller is then always pushing the system toward a central operating point (see Figure 9.13).

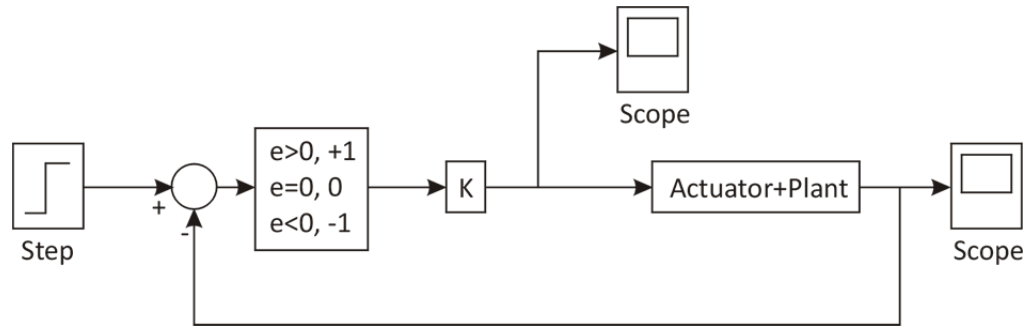


Figure 9.13 - Relay controller of Astrom-Hagglund

A step in the loop setpoint is used to perturb the loop from a stable operating state. K is chosen to give enough of a controller output to get a clear response but not to drive the system too far away from its stable operating point. For example, if the step has a size of 1, the error will suddenly become +1. Something like a 5% commanded change from what the controller can produce before saturating the actuator would be appropriate.

Chapter 9 – Designing and tuning PID controllers

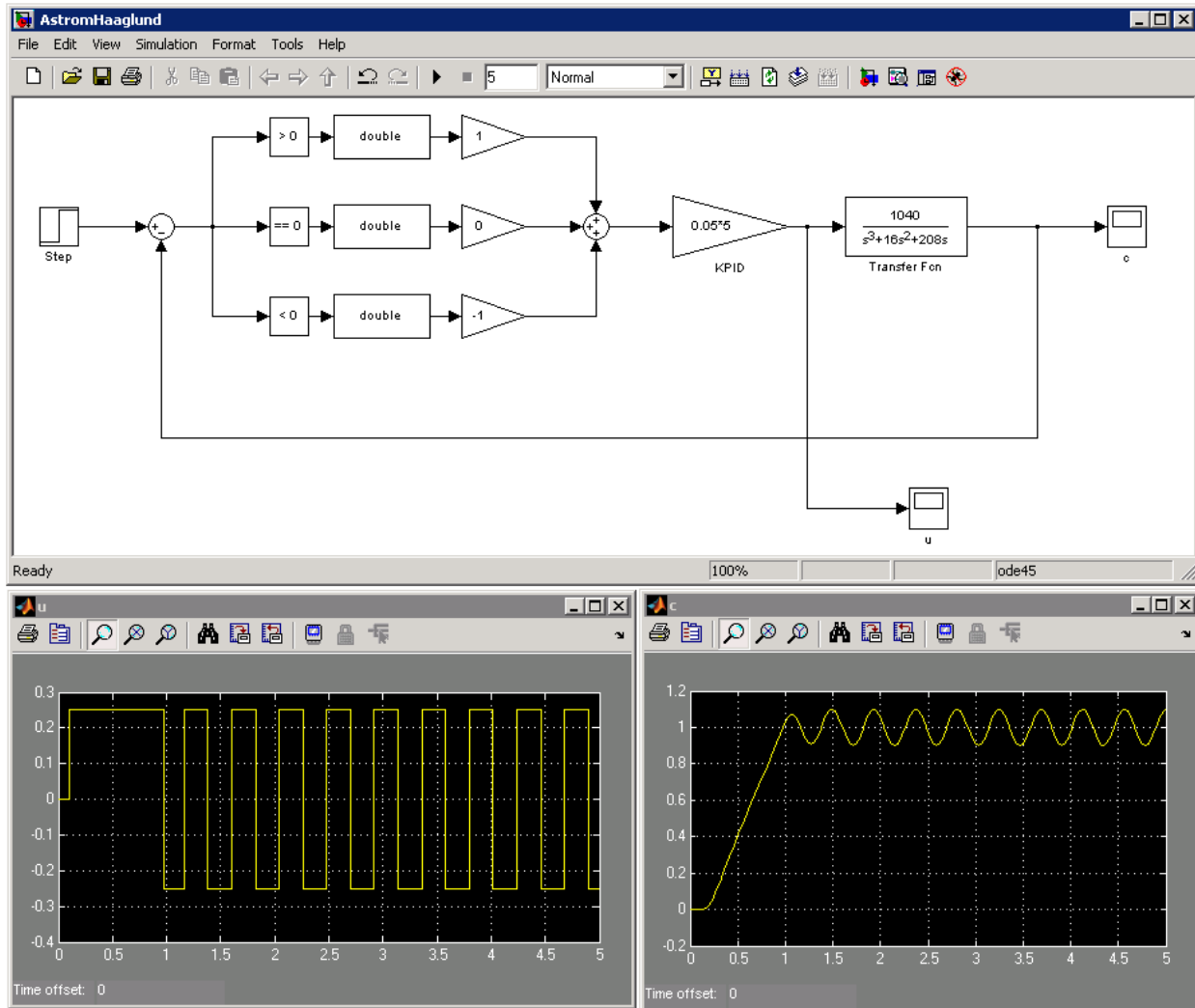


Figure 9.14 - Implementation of Astrom-Hagglund relay controller

Figure 9.14 shows this relay controller implemented in Simulink. With this system it has been determined that a unit step input would saturate the actuator with a controller gain of 5. The relay controller works as follows: If the actual output is below the desired level, the error will be positive, so a constant positive signal will be sent out of the controller nudging the system upward. If the actual output is above the desired level, the error will be negative, and a constant negative signal will be sent out to nudge the system back down. What results is a saw wave coming out of the controller and an oscillation from the system. Three parameters are needed from these plots. Figure 9.15 shows these parameters.

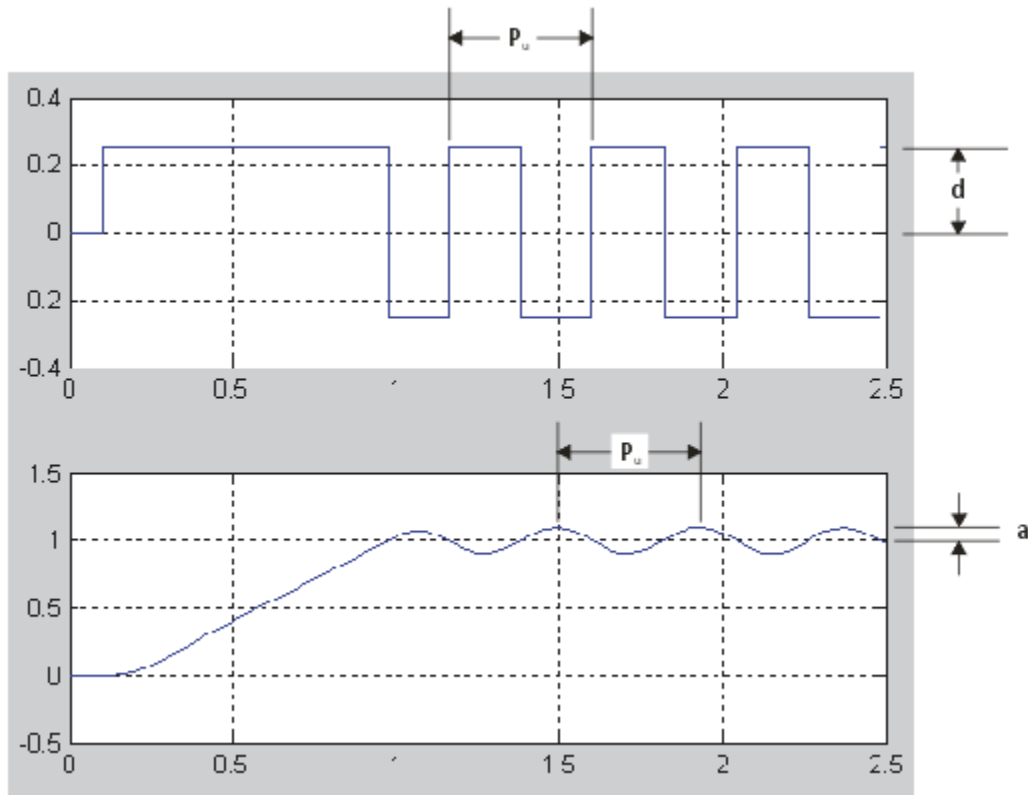


Figure 9.15 - Result of Astrom-Hagglund relay tuning

With d and a one can calculate K_u .

$$K_u = \frac{4 \cdot d}{\pi \cdot a}$$

This is the same K_u that is used in Ziegler-Nichols tuning.

Tuning methods —Chien, Hrones, Reswick tuning

Chien, Hrones, Reswick tuning was developed in 1952 as an alternative to Ziegler-Nichols tuning. Its advantage is that it has separate suggestions to make for controller gains, depending upon whether a loop is a regulator or a positioner loop. The test needed to find the response parameters is also somewhat safer than the Ziegler-Nichols test in that one is not driving the system up to the limits of stability.

An open-loop test is performed on the actuator+plant. This is normally a sequence of first-order or gain elements. Thus the step response appears as shown in Figure 9.16.

Chapter 9 – Designing and tuning PID controllers

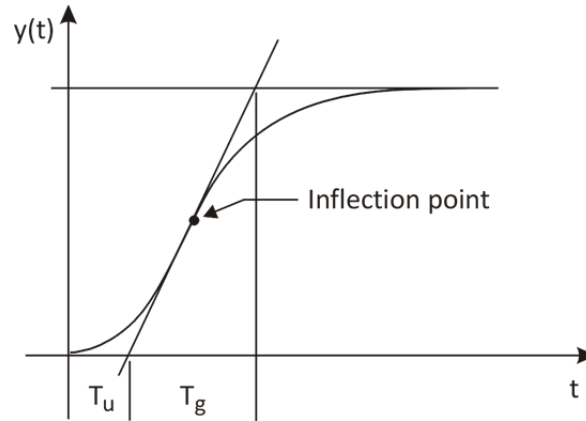


Figure 9.16 – Chien, Hrones, Reswick tuning open-loop step response

With this step response, one identifies the inflection point in the response. A tangent line is drawn through this point and extended until it crosses the horizontal axis and the final output level. This then defines the parameters T_u and T_g . Also used is K_{ss} , the steady-state gain of the actuator+plant. These three parameters are then used in the following table to set the parameters for a P, PI, or PID controller.

	Non-oscillatory response with shortest time to goal		20% overshoot with shortest time to goal	
	Positioner	Regulator	Positioner	Regulator
P	$K_{PID} = \frac{0.3 \cdot T_g}{K_{ss} \cdot T_u}$	$K_{PID} = \frac{0.3 \cdot T_g}{K_{ss} \cdot T_u}$	$K_{PID} = \frac{0.7 \cdot T_g}{K_{ss} \cdot T_u}$	$K_{PID} = \frac{0.7 \cdot T_g}{K_{ss} \cdot T_u}$
PI	$K_{PID} = \frac{0.35 \cdot T_g}{K_{ss} \cdot T_u}$ $T_I = 1.2 \cdot T_u$	$K_{PID} = \frac{0.6 \cdot T_g}{K_{ss} \cdot T_u}$ $T_I = 4 \cdot T_u$	$K_{PID} = \frac{0.6 \cdot T_g}{K_{ss} \cdot T_u}$ $T_I = T_u$	$K_{PID} = \frac{0.7 \cdot T_g}{K_{ss} \cdot T_u}$ $T_I = 2.3 \cdot T_u$
PID	$K_{PID} = \frac{0.6 \cdot T_g}{K_{ss} \cdot T_u}$ $T_I = T_g$ $T_D = 0.5 \cdot T_u$	$K_{PID} = \frac{0.6 \cdot T_g}{K_{ss} \cdot T_u}$ $T_I = 2.4 \cdot T_u$ $T_D = 0.42 \cdot T_u$	$K_{PID} = \frac{0.6 \cdot T_g}{K_{ss} \cdot T_u}$ $T_I = 1.35 \cdot T_g$ $T_D = 0.47 \cdot T_u$	$K_{PID} = \frac{0.6 \cdot T_g}{K_{ss} \cdot T_u}$ $T_I = 2 \cdot T_u$ $T_D = 0.42 \cdot T_u$

Tuning methods — Tietze-Schenk tuning method

This is a tuning method from the world of electrical engineering. The procedure is as follows.

1. Drive the system with a P-only controller.
2. Raise K_{PID} until a weakly damped oscillation is evident. The oscillation should last but does die out. This should be at about a 15° phase margin, so $\zeta \approx 0.15$, so %OS $\approx 60\%$.
3. Find T_s , the period of oscillation.
4. Set $T_D = T_s / (2 \cdot \pi)$.
5. Set $T_I = 10 \cdot T_D$.

Chapter 9 – Designing and tuning PID controllers

This works very well, even with unstable systems. Why does it work? The Bode diagram lies at the root of the rationale. We use the K - T_1 - T_2 -form of the PID controller:

$$G_{PID}(s) = \frac{K \cdot (T_1 \cdot s + 1)(T_2 \cdot s + 1)}{s}$$

So we are adding a gain, two first-order leads, and an integrator. By setting K_{PID} to obtain roughly 60% overshoot at ω_d for the closed-loop system, we have $\Phi_M \approx 15^\circ$ at $\omega = \omega_d$ for the open-loop system. If we place the second lead there, this adds an additional 45° to the phase margin to give $\Phi_M \approx 60^\circ$, so %OS \approx 10%. Thus the gain and one lead have been utilized. Remaining are another lead and the integrator. The integrator is added, as always, to limit or eliminate steady-state error. But the negative effect of adding the integrator will be to lower the phase angle 90° everywhere, i.e. for all ω . To preserve the phase margin at 60° , the first lead is added one decade before $\omega = \omega_d$, so at $\omega = \omega_d/10$. At this break frequency the phase contribution of the first lead is 45° , but by $\omega = \omega_d$, the break frequency of the second lead, the phase contribution of the first lead has reached 90° , so the phase loss of the integrator has been compensated.

Summing up,

$$T_2 = \frac{1}{\omega_{B2}} = \frac{1}{\omega_d} = \frac{T_s}{2\pi}$$

$$T_1 = \frac{1}{\omega_{B1}} = \frac{1}{\left(\frac{1}{10}\right) \cdot \omega_{B2}} = \frac{10}{\omega_{B2}} = 10 \cdot T_2$$

Converting to the K_{PID} - T_I - T_D -form needed by most controllers

$$G_{PID}(s) = K_{PID} \left(1 + \frac{1}{T_I \cdot s} + T_D \cdot s \right) = \frac{K \cdot (T_1 \cdot s + 1) \cdot (T_2 \cdot s + 1)}{s}$$

$$K_{PID} \cdot \left(\frac{T_D \cdot s^2 + s + 1/T_I}{s} \right) = \frac{K \cdot [T_1 \cdot T_2 \cdot s^2 + (T_1 + T_2) \cdot s + 1]}{s}$$

$$\frac{K_{PID} \cdot T_D \cdot s^2 + K_{PID} \cdot s + K_{PID}/T_I}{s} = \frac{K \cdot T_1 \cdot T_2 \cdot s^2 + K \cdot (T_1 + T_2) \cdot s + K}{s}$$

Thus

$$K_{PID} \cdot T_D = K \cdot T_1 \cdot T_2$$

$$K_{PID} = K \cdot (T_1 + T_2)$$

$$K_{PID}/T_I = K$$

From the second result,

Chapter 9 – Designing and tuning PID controllers

$$K_{PID} = K \cdot (T_1 + T_2) = K \cdot \left(\frac{10 \cdot T_s}{2\pi} + \frac{T_s}{2\pi} \right) = \frac{11 \cdot K \cdot T_s}{2\pi}$$

So

$$K = \frac{2\pi \cdot K_{PID}}{11 \cdot T_s}$$

Then

$$T_D = \frac{K \cdot T_1 \cdot T_2}{K_{PID}} = \frac{2\pi \cdot K_{PID}}{11 \cdot T_s} \cdot \frac{1}{K_{PID}} \cdot \frac{10 \cdot T_s^2}{(2\pi)^2} = \frac{10}{11} \cdot \frac{T_s}{2\pi} \approx \frac{T_s}{2\pi}$$

$$T_I = \frac{K_{PID}}{K} = \frac{11 \cdot T_s}{2\pi} \approx 10 \cdot T_D$$

P-only control in a regulator loop

In a regulator loop it is sometimes the case that just P control suffices because of a unique feature of the regulator. The regulator is set up to run off *deviations* from a design operating condition, not from *total amounts* of the loop variables. Take the common example of a tank level controller. The loop is set up as shown in Figure 9.17. For this particular example the tank's design steady state is 18" with the inlet valve open 70%. If the user puts in a setpoint of 18", the setpoint sent to the inner loop is 0". This indicates not that the user wants the tank to empty but rather that he or she wants the deviation from 18" to be 0".

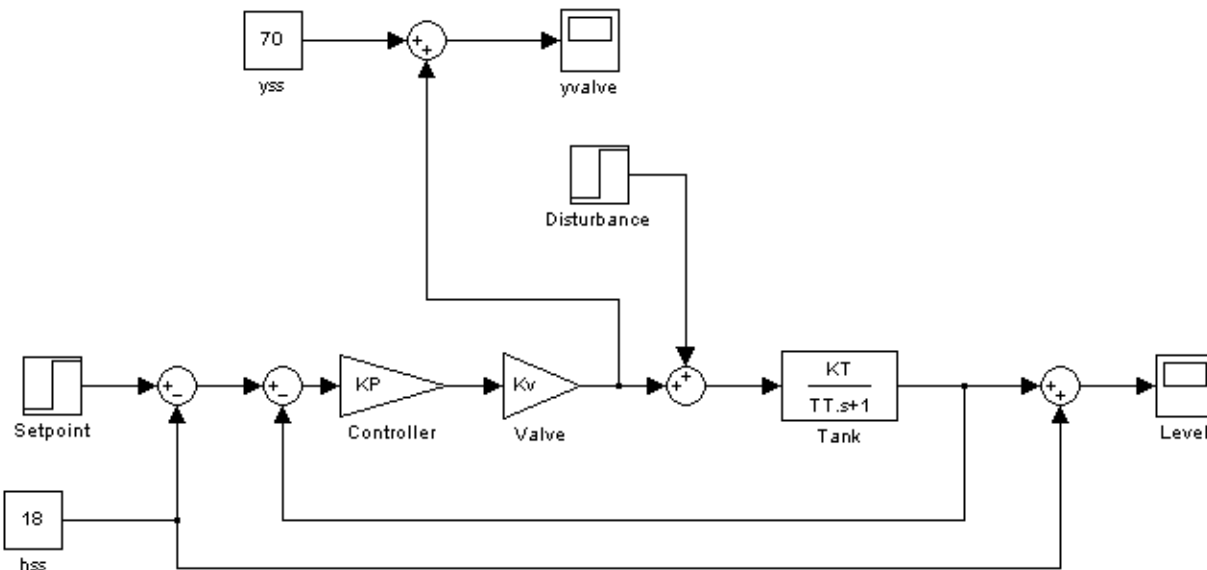


Figure 9.17 - Regulator loop with P-only controller

Chapter 9 – Designing and tuning PID controllers

Notice what happens at this operating level. The inner loop has 0" coming in, and every signal in the inner loop is 0, assuming there is no disturbance. So at this operating level, the design operating level, there is no steady state error. If the setpoint is changed or if a disturbance is input, there will be a steady state error, since the system is a type 0 system with only a proportional controller. If a positive disturbance flow is added, the tank will run a little high. If a negative disturbance flow is added, the tank will run a little low. This amounts to draining more from the tank than the normal design steady state flow rate. The "load" on the tank has been increased, so the tank level will *droop*. But in normal operating conditions, the tank level will remain at the design level without any steady state error.

This is a feature of P control of a type 0 system. It will have no steady state error at one operating point, namely the point where all the signals in the deviation loop are 0. Another common example is a cruise control on a car. When one sets the desired speed by engaging the controller, one is also setting the design operating level and setting up the deviation from the loop. If the controller in this loop is a P-only controller, when one encounters a hill, the increased load will cause the speed to droop a little. That is the speed will remain a few miles per hour below the set desired speed. This is tolerable. Indeed it may even be desirable. By not aggressively insisting with I-control that the speed exactly match the setpoint speed, one operates the car less aggressively and saves fuel.

Problems

- 9.1 Take the system of Figure 9.7 and work out its closed-loop transfer function. Determine ω_{n-CL} and ζ_{CL} . What rôles do K_p and K_D play respectively in ω_{n-CL} and ζ_{CL} ? That is, if K_p increases, what effect does that have respectively on ω_{n-CL} and ζ_{CL} ? If K_D increases, what effect does that have respectively on ω_{n-CL} and ζ_{CL} ?
- 9.2 It is often said of derivative control that some derivative control allows one to turn up K_p without running into the same problems one would without some derivative control. The primary problem encountered without derivative control is increased overshoot and swinging with an input step. To see if this adage is so, take the system of Figure 9.6 and increase K_p . Double it, for example, and compare the result with that of the same system with half the gain and no K_D . What effect on system response does turning K_p up have? That is, with this ability to turn up K_p without its normal, negative consequences, what is gained?
- 9.3 Given K_p , K_I , and K_D , what are K_{PID} , T_I , and T_D —the parameters of the second form of the PID—in terms of the parameters of the first form?
- 9.4 By pole cancellation, you have found out the values that you need of K , T_1 , and T_2 of the third form of the PID controller. The controller you are using, however, needs the three parameters of the second form. Determine expressions for K_p , T_I , and T_D of the second form in terms of K , T_1 , and T_2 of the third form.
- 9.5 Set up a PI-control loop for a first order plant with unity steady-state gain and a time constant of 80 seconds. Use $K_p = 0.6$ and a repeat time of 3 minutes.

Chapter 9 – Designing and tuning PID controllers

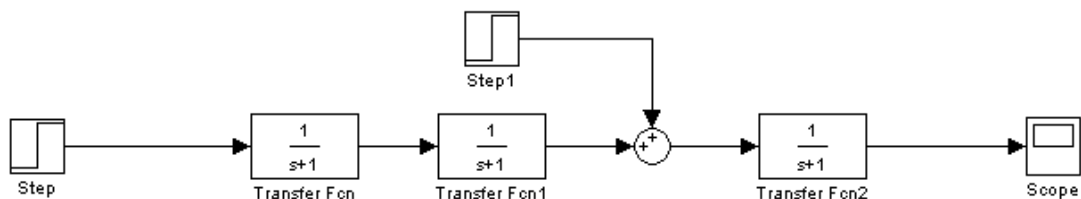
- 9.6 An unity-feedback, open-loop system has poles at $s = -6$ and -10 . It has a steady-state gain of 16.
- Design a controller to cancel this system's dynamics and replace them with a first-order with a time constant of 1 second.
 - Over time, the original system undergoes wear and tear and the open-loop poles drift to -5.5 and -9 , and the gain drifts to 15. How will this affect the closed-loop system's behavior? Give the closed-loop transfer function of both the original and the worn systems. Using Matlab, plot on the same plot the original system's behavior (unit step response) and then the behavior of the worn system.
 - Plot the root loci of each system, showing on each diagram the location of the closed-loop pole.
- 9.7 The actuator+plant with the transfer function

$$G_{OL}(s) = \frac{1040}{s^3 + 16 \cdot s^2 + 208 \cdot s}$$

is to be outfitted with a PID controller in a unity-feedback loop. Use the Ziegler-Nichols tuning method to tune controllers for this system. Install a $K_{PID}-T_I-T_D$ -format controller in a Simulink model. Once you have figured out K_U and P_U for this system, copy the system so that there are three copies of it in a single Simulink model. Set up in the loop respectively a P, a PI, and a PID controller. Run the output from each loop into a MUX block and then to a scope, so that the unit response to each loop can be compared against the other two.

To hand in: Show all calculations, report K_U and P_U clearly. Print out a picture of the Simulink model. Print out the scope output that compares the three different controllers.

- 9.8 The system of the previous example was used to illustrate the Astrom-Hagglund relay controller, used to calculate K_U and P_U . Find the values of these two parameters from the plots given in the text and compare them with what you found in the previous problem.
- 9.9 Take the system shown below



Use the Chien, Hrones, Reswick tuning method to design a PID controller for this system to reject a unit step disturbance. Show your work. Design a controller for no overshoot and then one for 20% overshoot. Give controller gains for each case. Demonstrate the reaction of both loops to a unit-step disturbance.

Chapter 9 – Designing and tuning PID controllers

9.10 Take the system

$$G_{OL}(s) = \frac{30}{0.005 \cdot s + 1} \cdot \frac{0.23}{0.53 \cdot s + 1} \cdot \frac{1}{0.055 \cdot s + 1}$$

Use the Tietze-Schenk tuning method with a Simulink model to design a PID controller for this system. Give the values of K_{PID} , T_I , and T_D that result from Tietze-Schenk. Print out your Simulink model and a unit-step response of the tuned system.