

EENG307: Proportional-Integral-Derivative (PID) Control Design, Simulation, and Evaluation*

Lecture 17

Elenya Grant, Kathryn Johnson, and Hisham Sager[†]

Fall 2022

Contents

1 Pre-requisite Material	1
2 Overview of PID Control	2
3 Proportional Control	2
4 Proportional/Integral (PI) Control	6
5 Using Simulink to Simulate Control Systems	7
5.1 Simulink Example: PI Controller	8
6 Lecture Highlights	14
7 Quiz Yourself	14
7.1 Questions	14
7.2 Solutions	16
8 Resources	20
8.1 Books	20
8.2 Web resources	20
9 Appendix: Implementing a PID controller on a microcontroller	21
9.1 Approximation to a Derivative	21
9.2 Approximation to an Integral	21
9.3 Anti-windup - accounting for actuator saturation	22

1 Pre-requisite Material

This lecture assumes that the reader is familiar with the following material:

- Lecture 9: Introduction to Control Concepts and More on Stability
- Lecture 12: Time Response of Higher Order Systems
- Lecture 13: Proportional-Derivative (PD) Control to Improve Transient Response

* This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/3.0/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

[†] Developed and edited by Tyrone Vincent and Kathryn Johnson, Colorado School of Mines, with contributions from Salman Mohagheghi, Chris Coulston, Kevin Moore, CSM and Matt Kupiliuk, University of Alaska, Anchorage

- Lecture 14: Rotational and Fluid Systems

2 Overview of PID Control

Proportional, integral, derivative (PID) control is one of the most common types to appear in industry because it is fairly easy to implement and tune. In Lectures 9 and 13, we have seen some examples of P and PD control. In this lecture, we'll do another example of P control for a fluid system, then show how using PI control can improve the steady-state behavior.

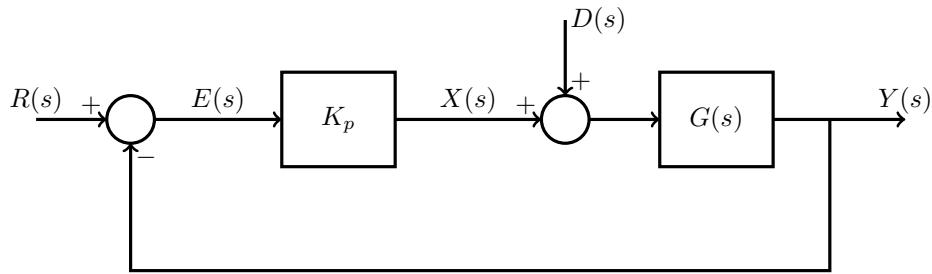
Because PID control can be so commonly used, including in other classes, in the Appendix we provide information about implementing it on a microcontroller.

3 Proportional Control

We have already seen an example of P control for a mass-spring-damper system in Lecture 13: Proportional-Derivative (PD) Control to Improve Transient Response. Let's revisit some of the main points, as well as applying this control strategy to a fluid system and then a more generalized mass-spring-damper system.

To apply feedback control, the output y is measured, compared to a desired reference r to give the error signal $e = r - y$. This error is then applied to the control algorithm. If we use a proportional controller (P), the actuator command x is proportional to the error. The gain of the proportional control will be denoted K_p .

Proportional Control



When $G(s)$ is a first or second order system, the design of the controller can be straightforward, consisting of three main steps.

1. **Collect the design specifications.** Design specifications could be in terms of transient response (rise time, settling time, overshoot) or steady state response (reference tracking or disturbance rejection).
2. **Find the relevant *closed loop* transfer functions for your design specifications.** The closed loop transfer functions give the response of the controlled system, as viewed from a reference or disturbance to the output or error. Specifically, using the block diagram simplification rules,

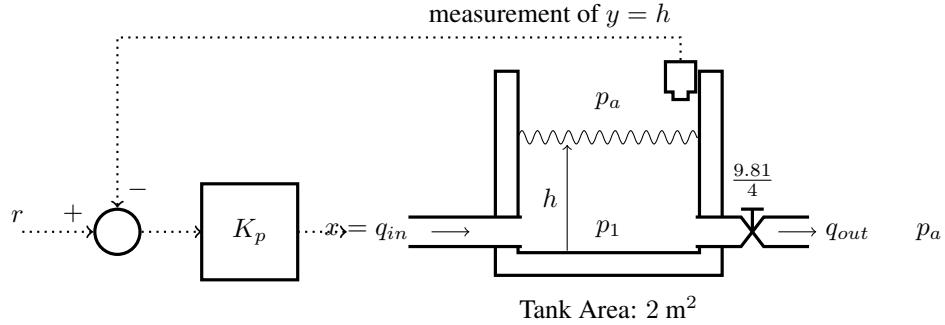
$$\begin{aligned} \frac{Y(s)}{R(s)} &= \frac{K_p G(s)}{1 + K_p G(s)}, \\ \frac{E(s)}{R(s)} &= \frac{1}{1 + K_p G(s)}, \\ \frac{E(s)}{D(s)} &= -\frac{G(s)}{1 + K_p G(s)}. \end{aligned}$$

You will plug the specific open loop transfer function $G(s)$ into these formulas. Note that $\frac{E(s)}{D(s)}$ is only necessary if a disturbance rejection specification is given.

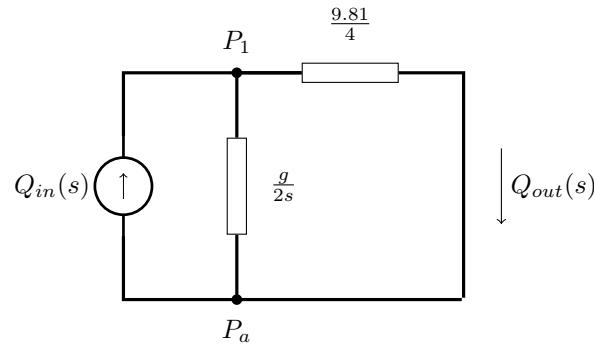
3. **Select K_p so that the design specifications are met, or determine that no K_p exists to meet the specifications.**

Example 1. Design a feedback control system to regulate the height of fluid in a tank, assuming the input flow can be freely assigned. The control system must ensure that the actual tank height follows a reference step command with a settling time $t_s \leq 0.2$ s. In addition, the steady state error between the reference and actual height for a unit step command should be less than or equal to 0.1.

A diagram of the feedback control system is below. Note that the output y is the measurement of the tank height h . This is subtracted from the reference r . The resulting error is multiplied by K_p and the flow is set to be this value. The area of the tank is 2 square meters, the valve constant is $9.81/4$, and the fluid density is $\rho = 1$.



The open loop system transfer function is found using our standard modeling techniques. For example, the following expresses the system as an impedance network with the reference at atmospheric pressure p_a .



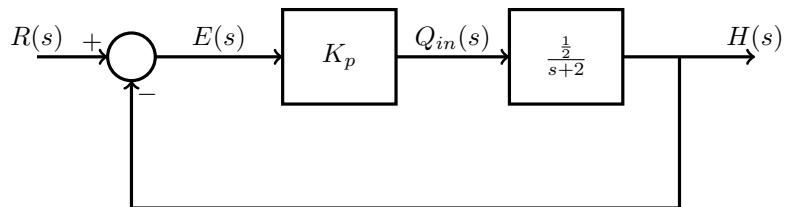
We also have the auxiliary equation $\rho gh = p_1$, or since $\rho = 1$, $h = \frac{1}{g}p_1$. Thus, the transfer function $H(s)/Q_{in}(s)$ can be found by first finding $P_1/Q_{in}(s)$, and then dividing by g . Combining the impedances in parallel,

$$\begin{aligned} \frac{P_1}{Q_{in}(s)} &= \frac{\frac{g}{2s} \frac{9.81}{4}}{\frac{g}{2s} + \frac{9.81}{4}} \\ &= \frac{\frac{g}{2}}{\frac{9.81}{4} + s} \end{aligned}$$

Using $g = 9.81$ and $h = \frac{1}{g}p_1$

$$G(s) = \frac{H(s)}{Q_{in}(s)} = \frac{\frac{1}{2}}{s + 2}$$

The feedback control system can be represented with the following block diagram. (Since no disturbance rejection specification was listed, a disturbance input was not included.) Note that $R(s)$ is the *desired* tank height.



We can now proceed with the design process:

1. Design specifications

- $t_s \leq 0.2$ s
- $e_{ss} \leq 0.1$ for unit step reference

2. Closed loop transfer functions

$$\frac{H(s)}{R(s)} = \frac{K_p/2}{s + \frac{K_p}{2} + 2}$$

$$\frac{E(s)}{R(s)} = \frac{s + 2}{s + \frac{K_p}{2} + 2}$$

Note that the closed loop transfer functions are also first order.

2. Select K_p .

We have two specifications. The first constraint on settling time can be converted to a constraint on the closed loop pole location. We previously showed that for a first order system with pole at $s = -\sigma$,

$$t_s = \frac{4.6}{\sigma}$$

Since our specification is $t_s \leq 0.2$, we require $\frac{4.6}{\sigma} \leq 0.2$ or

$$\sigma \geq \frac{4.6}{0.2} = 23$$

The pole of our closed loop feedback system is at $s = -(\frac{K_p}{2} + 2)$. Thus, we need to choose

$$\frac{K_p}{2} + 2 \geq 23$$

or

$$K_p \geq 42.$$

The second constraint on reference steady state error translates to a constraint on the DC gain of $K_p G(s)$. We can either recall that for a step reference $e_{ss} = \frac{1}{1+K_s}$ where $K_s = K_p G(0) = \frac{K_p}{4}$, or we can solve directly using the final value theorem

$$e_{ss} = \lim_{s \rightarrow 0} sE(s) = \lim_{s \rightarrow 0} s \frac{s+2}{s + \frac{K_p}{2} + 2} \frac{1}{s} = \frac{2}{2 + \frac{K_p}{2}} = \frac{1}{1 + \frac{K_p}{4}}$$

Since we require

$$e_{ss} = \frac{1}{1 + \frac{K_p}{4}} \leq 0.1,$$

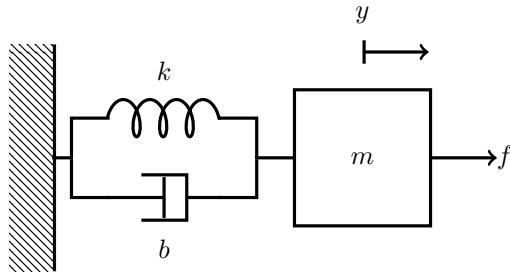
we can solve for the constraint

$$K_p \geq 36$$

Since the most stringent requirement is on the settling time, our final design choice is

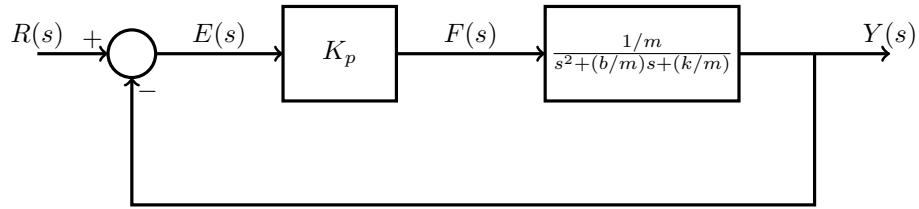
$$K_p \geq 42$$

Example 2. Determine how a proportional feedback control system will change the closed loop behavior when controlling the position of the mass-spring-damper system using force actuation



$$\frac{Y(s)}{F(s)} = \frac{1/m}{s^2 + (b/m)s + (k/m)}$$

The proportional feedback control system would be represented by the block diagram



The closed loop transfer function is

$$\frac{Y(s)}{R(s)} = \frac{K_p/m}{s^2 + (b/m)s + (k + K_p)/m}.$$

We can get a very general sense of how changes in K_p affect the transient response by finding the damping ratio and natural frequency for the closed loop poles. In particular

$$\omega_n = \sqrt{\frac{k + K_p}{m}}, \quad \zeta = \frac{b}{2\sqrt{m}\sqrt{k + K_p}}, \quad \zeta\omega_n = \frac{b}{2m}.$$

From the form of these equations we see that

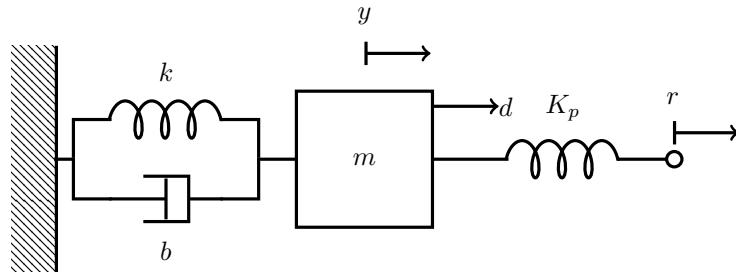
$$\uparrow K_p \implies \begin{cases} \uparrow \omega_n & \implies \text{reduced rise time} \\ \downarrow \zeta & \implies \text{increased overshoot} \\ \leftrightarrow t_s & \implies \text{no change in settling time} \end{cases}$$

Thus, while increasing K_p makes the response faster, it also makes it more oscillatory, and in the second order case, proportional control cannot change the settling time.

A physical intuition for the behavior of proportional control on a second order system can be obtained by implementing proportional control using a *mechanical* controller. Note that in proportional control, the control force f should be given by

$$f(t) = K_p(r(t) - y(t)).$$

Let's implement this force by attaching a spring with spring constant K_p with one end on the mass, and the other end attached to a point that specifies the reference r , as follows:



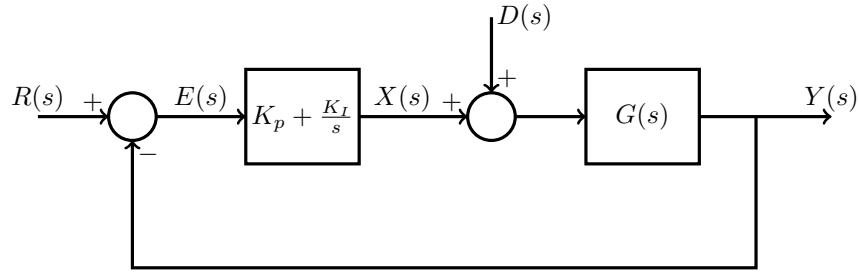
Increasing K_p corresponds to increasing the spring constant. Thus, the system becomes stiffer, but without added damping, this means the response is more oscillatory.

Mass-spring-damper systems tend to be underdamped, often requiring D control to dampen oscillations. See Lecture 13: Proportional-Derivative (PD) Control to Improve Transient Response for more information.

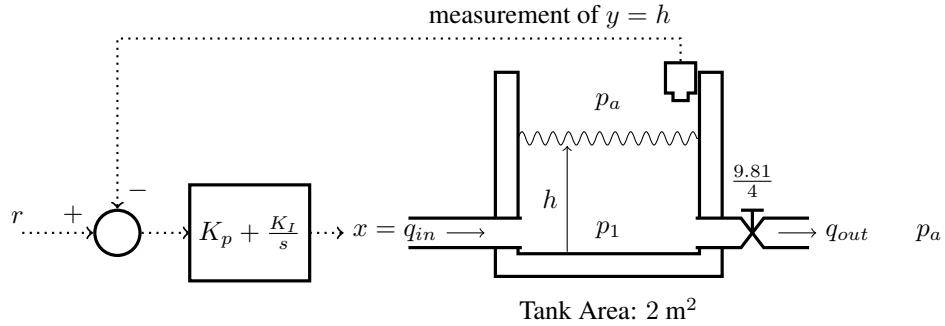
4 Proportional/Integral (PI) Control

As its name suggests, a PI controller consists of both a gain and an integral term. This controller is used when it is desired to improve the steady state response by increasing the system type. This controller is implemented as shown in the following block diagram

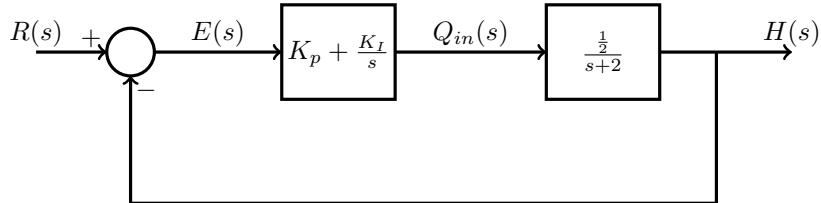
Proportional/Integral (PI) Control



Example 3. Let's return to the tank control problem, with a specified settling time of $t_s \leq 0.2$ s, but now 0 steady state error for a unit step reference input. In order to meet these specifications with a proportional controller, we would need to apply infinite gain, so we instead turn to a PI control



which has block diagram



The closed loop transfer functions are

$$\frac{H(s)}{R(s)} = \frac{\frac{K_p}{2}s + \frac{K_I}{2}}{s^2 + \frac{4+K_p}{2}s + \frac{K_I}{2}}$$

$$\frac{E(s)}{R(s)} = \frac{s^2 + 2s}{s^2 + \frac{4+K_p}{2}s + \frac{K_I}{2}}$$

Note that since $E(s)/R(s)$ has a zero at $s = 0$, we automatically meet the steady state error specification, as

$$e_{ss} = \lim_{s \rightarrow 0} s \frac{s^2 + 2s}{s^2 + \frac{4+K_p}{2}s + \frac{K_I}{2}} \frac{1}{s} = 0$$

We need only choose K_p and K_I to meet our settling time specification.

Since $2\zeta\omega_n = \frac{4+K_p}{2}$, $\zeta\omega_n = 1 + \frac{K_p}{4}$. Thus, we require

$$t_s = \frac{4.6}{\zeta\omega_n} = \frac{4.6}{1 + \frac{K_p}{4}} \leq 0.2$$

so we should choose

$$K_p \geq 88$$

K_I can be chosen as desired. For example, we may wish to have a reasonable damping ratio.

From above, $\zeta = \frac{1+\frac{K_p}{4}}{\omega_n}$, and from $\frac{H(s)}{R(s)}$, $\omega_n = \sqrt{\frac{K_I}{2}}$. With $K_p = 88$,

$$\zeta = \frac{(1 + 88/4)}{\omega_n} = \frac{23}{\sqrt{K_I/2}}$$

So $K_I = 4232$ will give a damping ratio of 0.5.

5 Using Simulink to Simulate Control Systems

In prior semesters, this lecture was the first in which Simulink was formally introduced. We have chosen to leave some of the Simulink content from prior semesters here for easy reference; for students who successfully applied P and PD control previously, the PI control section (Section 5.1) might be of most interest.

MATLAB has a graphical interface to run simulations of systems that are represented by block diagrams. While this is not as useful for creating simulation models of interconnected physical systems (in this case, another product, called Simscape is more useful) it is useful for creating simulations of systems with well defined input/output relationships, such as feedback control systems.

In addition to the example below, you can find a video introduction to Simulink at this link:

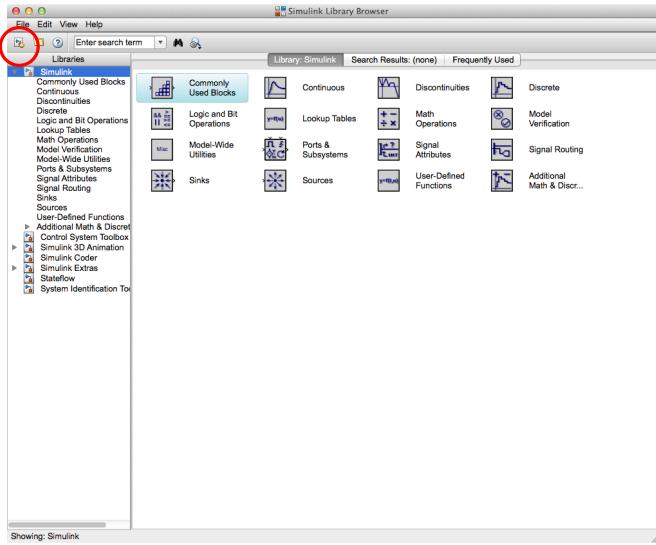
<http://www.mathworks.com/videos/introduction-to-simulink-81623.html>

You can start Simulink from the MATLAB command line:

```
» simulink
```

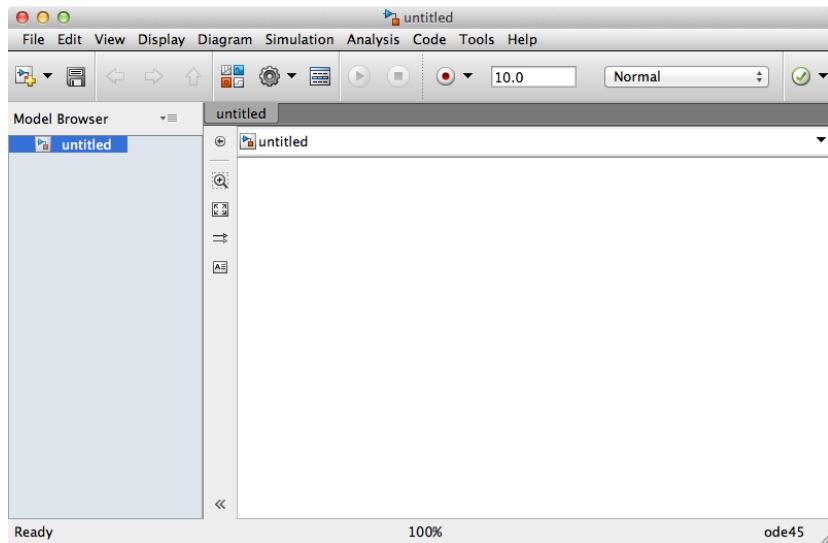
This will open up the following window, which give you access to all of the block elements in the Simulink Library

Simulink Library



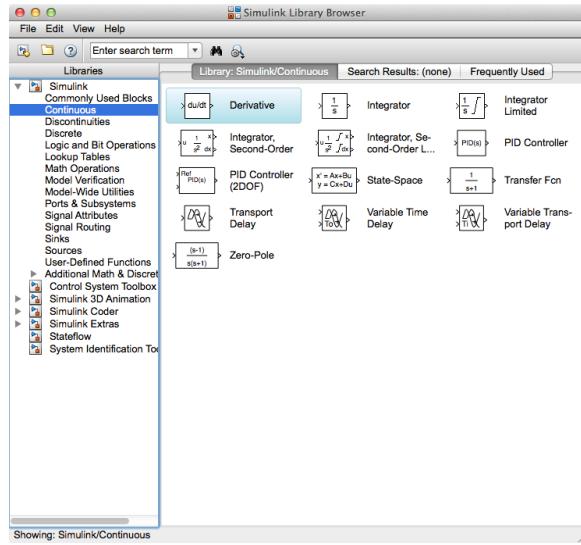
To create a block diagram, we need to open up a new model window. This can be done by clicking on the new model icon (circled in red above). The result is a window that looks like the following:

Blank Simulink Model

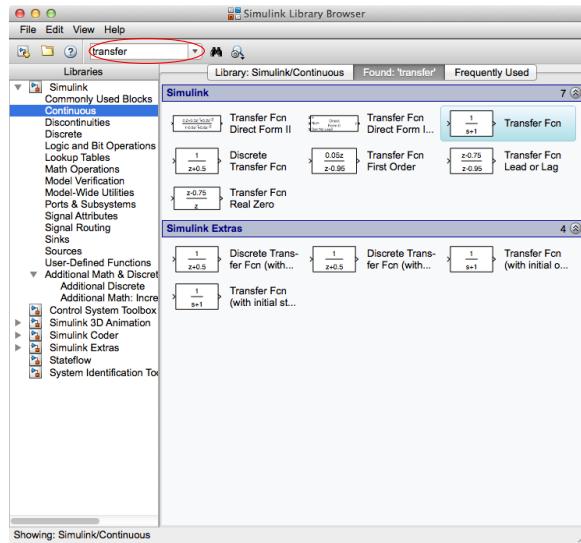


5.1 Simulink Example: PI Controller

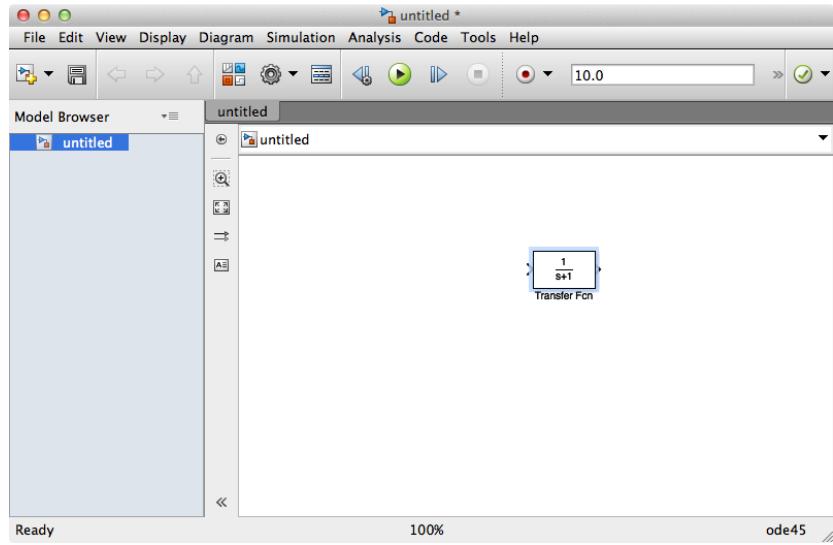
As an example, we will simulate the feedback control system for Example 3. We will grab blocks from the Simulink library and drag them over to the Simulink Model. The first block we need is a transfer function. There are two ways we can find a block. If we know which library it is in, we can click on that library in the “Libraries” window. So, for example, I know that transfer function blocks can be found in the “Continuous” library, and clicking on the word Continuous shows the blocks in that library. Note that the transfer function block is in the third column and third row.



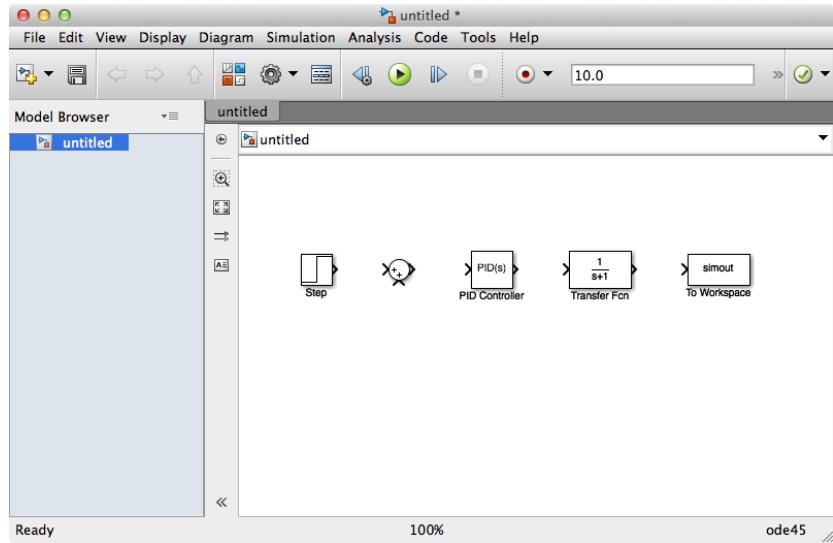
Alternately, we can search for the block using the search function. Type “transfer” into the search area (circled below) and you should see the list of blocks shown below. Note that the transfer function block is in the third column and first row.



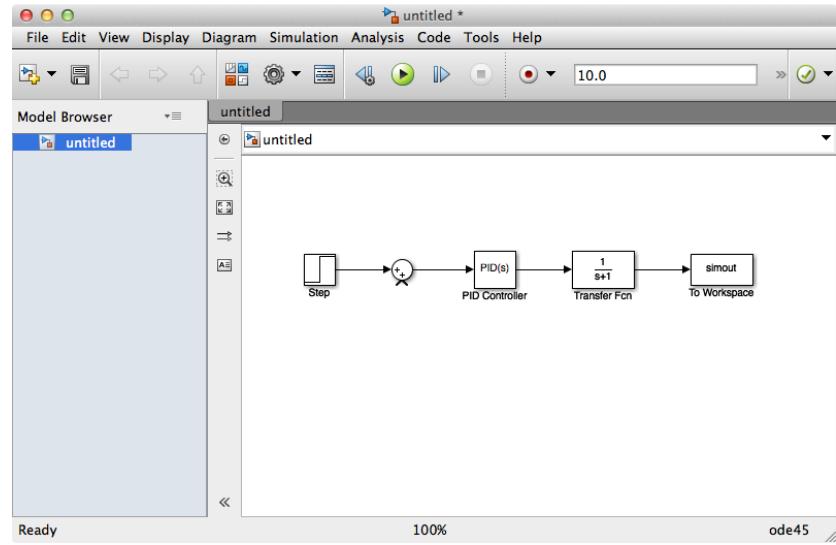
You can click on the transfer function block and drag it over to your model window. The model window should look like the following:



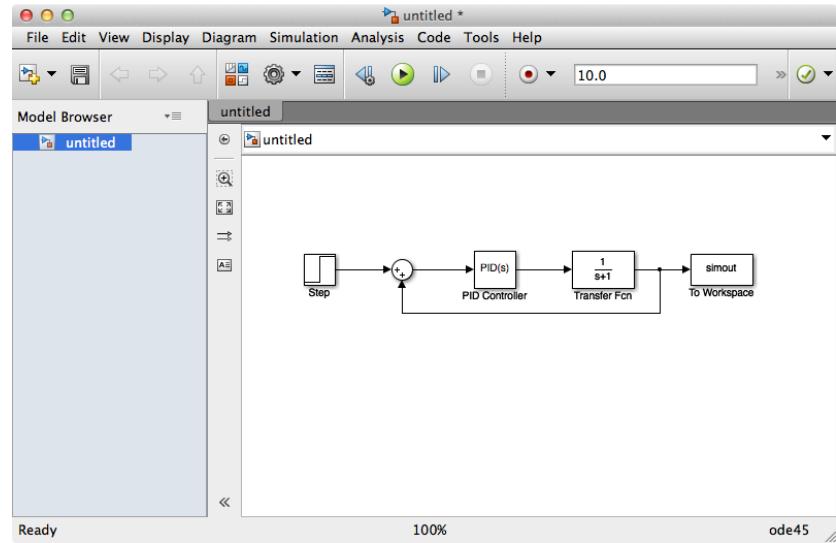
Now do the following: In the library browser, click on the Continuous library, and drag the PID block to the model window, click on the Math Operations library and drag the sum to the model window, click on the Sources library and drag the step to the model window, and click on the Sinks library and drag the workspace block to the model window. Arrange the blocks in a line similar to the following:



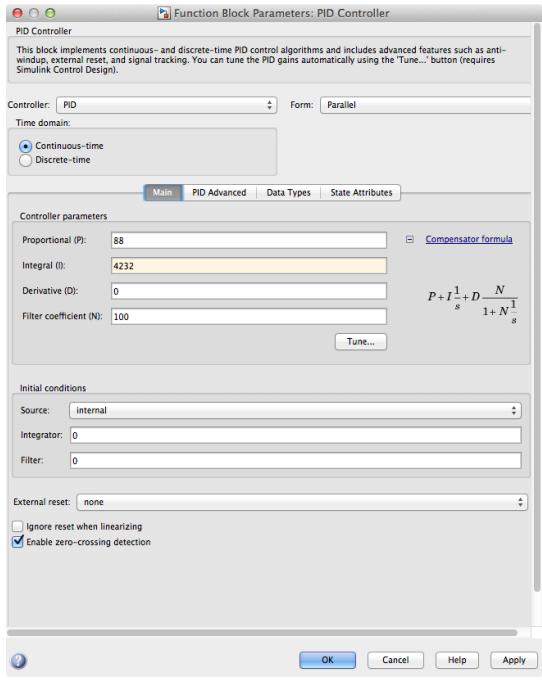
We can now connect up the blocks. Hover over the right side of the step block, in the region of the thick arrow head. The cursor should change to a plus sign. Click and drag the cursor over to the left side of the summer. An arrow should now connect these two blocks. Repeat this process until the model window is as follows:



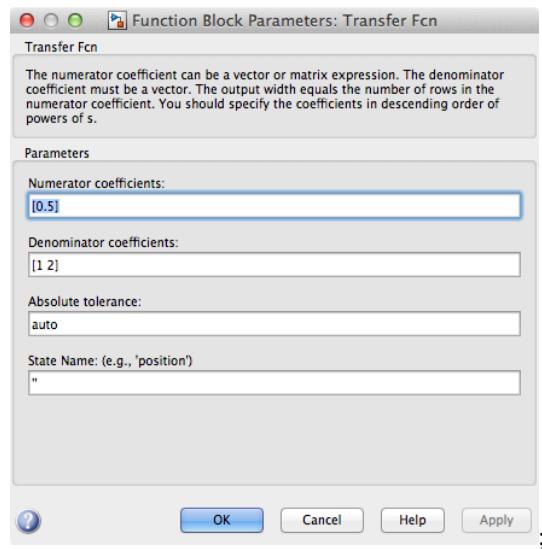
We still need to connect the output of the transfer function block to the lower input of the summer. Do this by holding down the control key while clicking on the arrow that goes between the transfer function and the workspace block. Drag the cursor down and to the left, and then up until you hit the bottom of the summer. The result should look like the following:



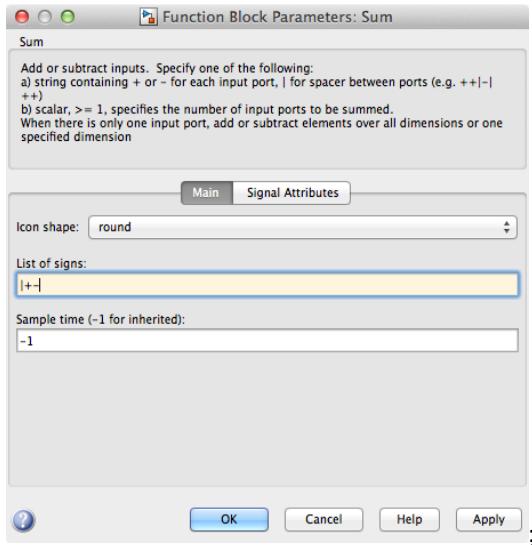
Now we can enter in the data for the blocks. Double click on the PID block, and a parameter window should open. Enter in the data so that the parameter window looks like the following: (Note that we are using K_p and K_I from Example 4). Now hit OK.



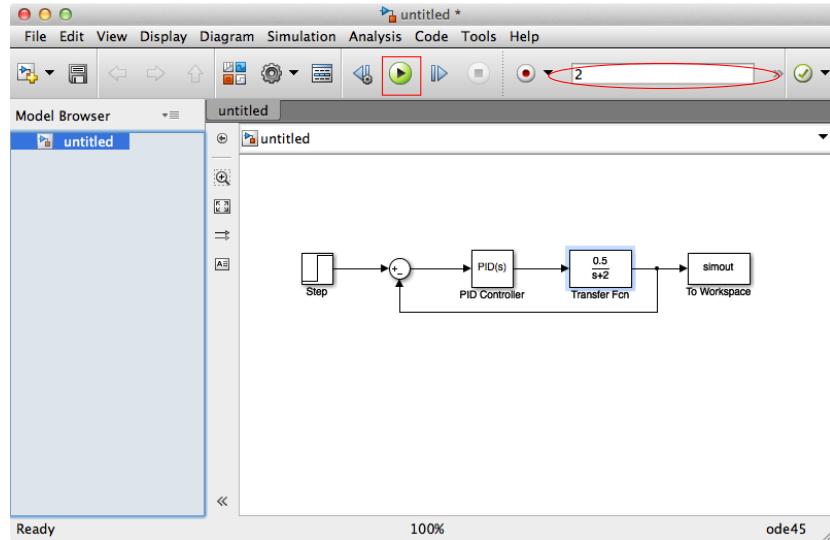
Now double click on the transfer function block, and enter in the system transfer function for Example 4. Hit OK.



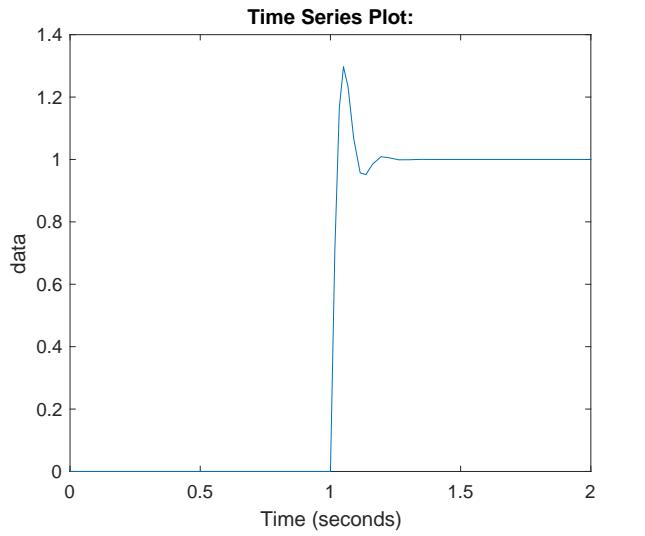
The last thing we need to do is change the summer so that the bottom input is subtracted, not added. Double click on the summer, and change the last plus sign to a minus, as below. Hit OK.



We are now ready to run the simulation. To choose the length of the simulation, enter the number of seconds in the box circled in red below. I have chosen 2 seconds. Then click on the green run button that is boxed in red below.



Because we added a To Workspace block, the results of the simulation are in the variable `simout` in the Matlab workspace. The default is for this variable to be a `timeseries`, which is a structured variable that has several elements. Type `» simout` at the command line to see the structure. We can plot the results either by typing `» plot(simout)` or by typing `» plot(simout.Time, simout.Data)`. This should result in the following plot:



6 Lecture Highlights

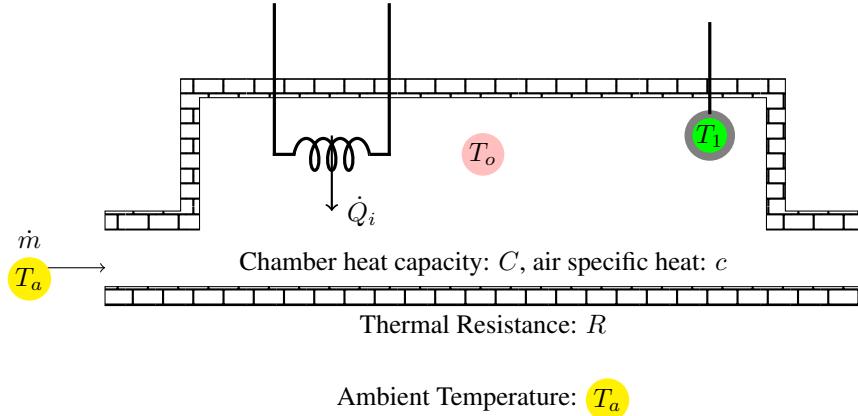
The primary takeaways from this article include

1. make sure to update list for Fa'22
2. Proportional-Integral-Derivative (PID) control is commonly used in industry due to its straightforwardness and long history, which makes tuning the controller gains easier.
3. The proportional gain K_p multiplies the error signal $e(t)$, the integral gain K_I multiplies the integral of the error signal $\int e(t)$, and the derivative gain K_D multiplies the derivative of the error signal $\dot{e}(t)$.
4. The PID gains can be computed to cause the closed-loop poles to lie within the allowable region to meet transient response specifications. However, since rules of thumb are not always precise, it is common for the system response not to meet specifications based on these initial calculations. Iterative tuning and checking response (for example, in Matlab) is common practice.
5. Simulink is a Matlab tool that enables additional flexibility for modeling more complex systems than may be possible in the Matlab command window. *Note: a common point of confusion is the fact that Simulink models signals, such as the reference input, as blocks. This is opposed to the arrow designator for signals in standard block diagram conventions.*

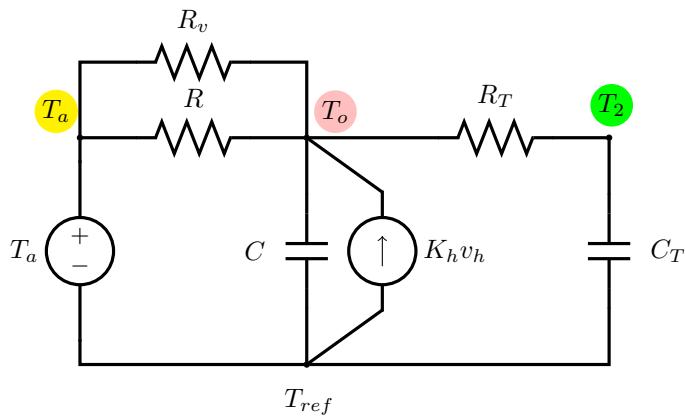
7 Quiz Yourself

7.1 Questions

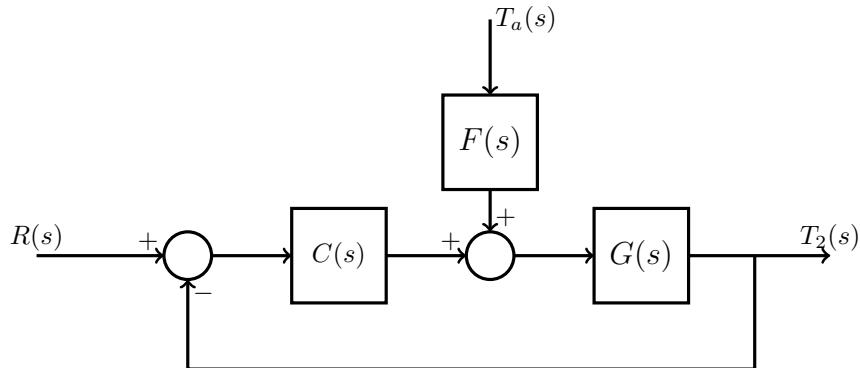
1. Consider a thermal system



that can be modeled with an equivalent thermal circuit (that can be treated just like an electrical circuit containing resistors, capacitors, a “voltage” source $v_{in} = T_a$, and a “current” source $i_{in} = K_h v_h$)



You are going to design a PD controller for this system. You will measure the temperature reading of the thermocouple, compare it to a reference, and then use the PD controller to calculate the required voltage to the heater, v_h . The block diagram of your control system is as follows:



where $C(s) = K_p + sK_d$, $G(s) = \frac{T_2(s)}{V_h(s)}$, and $F(s) = \frac{T_2(s)}{T_a(s) G(s)}$. The system parameters are $R_v = R = 2$, $C = 1$, $R_T = 0.1$, $C_T = 0.1$, $K_h = 1$.

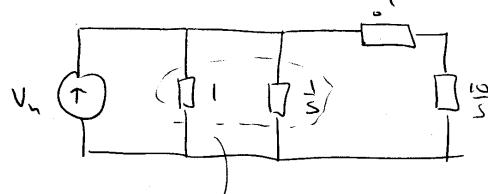
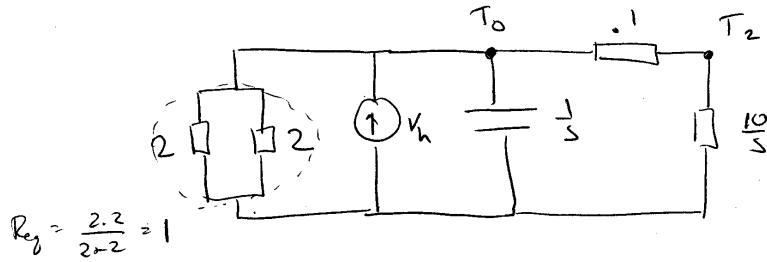
- Find $G(s)$ and $F(s)$. In each case, to find the transfer function from an input to an output signal, set all other sources equal to zero. In other words, to find $G(s) = \frac{T_2(s)}{V_h(s)}$, let $T_a(s) = 0$ (short circuit). To find $F(s) = \frac{T_2(s)}{T_a(s) G(s)}$, let $V_h(s) = 0$ (open circuit).
- Design a PD control so that a unit step command from $R(s)$ has rise time $t_r = 0.1\text{s}$ and overshoot $\%OS = 10\%$. For this design you can assume $T_a(s) = 0$.

- (c) Confirm your design by finding the closed loop transfer function from $R(s)$ to $T_2(s)$ and using MATLAB to plot the step response (See Lecture 12, section 4). You can assume $T_a(s) = 0$.
- (d) Find the steady state error for the unit step reference command.
- (e) Find the steady state error to a unit step change in the ambient temperature.

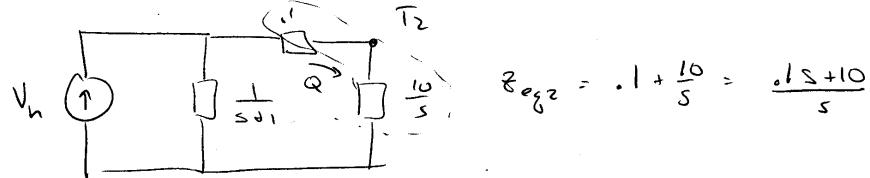
7.2 Solutions

1.

① to find $a(s)$, set $T_a = 0$



$$Z_{eq} = \frac{\frac{1}{s}}{1 + \frac{1}{s}} = \frac{1}{s+1}$$

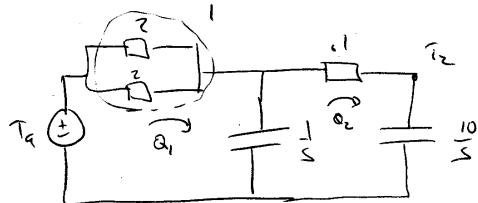


current divider: $Q = \frac{s}{s+10} \cdot V_h = \frac{s}{s^2 + 11s + 10} V_h$

$$T_2 = \frac{10}{s} \cdot Q \quad T_2(s) = \frac{10}{s^2 + 11s + 10} V_h(s)$$

$$G(s) = \frac{T_2(s)}{V_h(s)} = \frac{100}{s^2 + 11s + 100}$$

To find $F(s)$, need $\frac{T_2(s)}{T_1(s)}$, so set $V_u = 0$



$$\begin{bmatrix} T_1(s) \\ 0 \end{bmatrix} = \begin{bmatrix} 1 + \frac{1}{s} & -\frac{1}{s} \\ -\frac{1}{s} & \frac{1}{s} + \frac{10}{s} + 1 \end{bmatrix} \begin{bmatrix} Q_1(s) \\ Q_2(s) \end{bmatrix}$$

$$Q_2(s) = \frac{\begin{vmatrix} \frac{s+1}{s} & T_1(s) \\ -\frac{1}{s} & 0 \end{vmatrix}}{\begin{vmatrix} \frac{s+1}{s} & -\frac{1}{s} \\ -\frac{1}{s} & \frac{1}{s} + \frac{11}{s} \end{vmatrix}} = \frac{\frac{1}{s}}{\frac{(s+1)(s+1)}{s^2} - \frac{1}{s^2}} T_1(s)$$

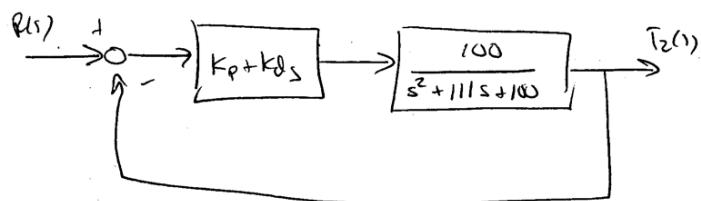
$$= \frac{\frac{1}{s}}{\frac{1}{s^2} + 11s + 10} T_1(s)$$

$$= \frac{s}{s^2 + 11s + 10} T_1(s) = \frac{10s}{s^2 + 11s + 100} T_1(s)$$

$$T_2(s) = \frac{10}{s} Q_2(s) = \frac{100}{s^2 + 11s + 100} T_1(s)$$

$$\text{Thus } F(s) = \frac{100}{s^2 + 11s + 100} \cdot \frac{s^2 + 11s + 100}{100} = 1$$

(b)



$$\text{Want } \zeta_r = 0.1 \quad \%os = 10\%$$

$$\Rightarrow 0.1 = \frac{2\zeta}{\omega_n} \quad \text{or} \quad \omega_n = 22$$

$$\zeta = \frac{\ln(1)}{\sqrt{(\ln(1))^2 + \pi^2}} = 0.59$$

$$\text{desired closed loop denominator: } s^2 + 2\zeta\omega_n s + \omega_n^2 = s^2 + 26s + 484$$

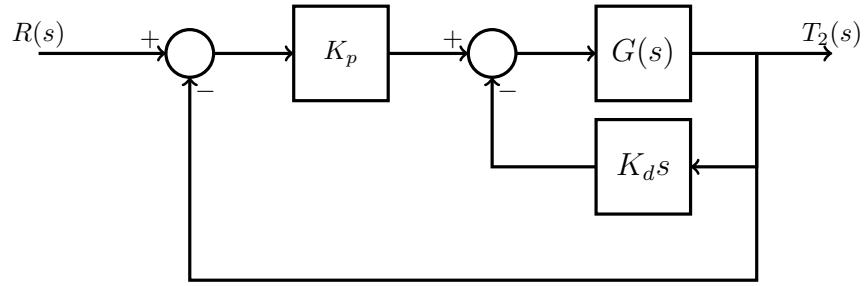
Closed loop denominator as function of K_p, K_d :

$$\frac{T_2(s)}{R(s)} = \frac{\frac{100(K_p + K_d s)}{s^2 + 111s + 100}}{1 + \frac{100(K_p + K_d s)}{s^2 + 111s + 100}} = \frac{100(K_p + K_d s)}{s^2 + (111 + 100K_d)s + 100(1+K_p)}$$

$$100(1+K_p) = 484 \Rightarrow \underline{K_p = 3.84}$$

$$111 + 100K_d = 26 \Rightarrow \underline{K_d = -0.85}$$

(c) We want to simulate the configuration



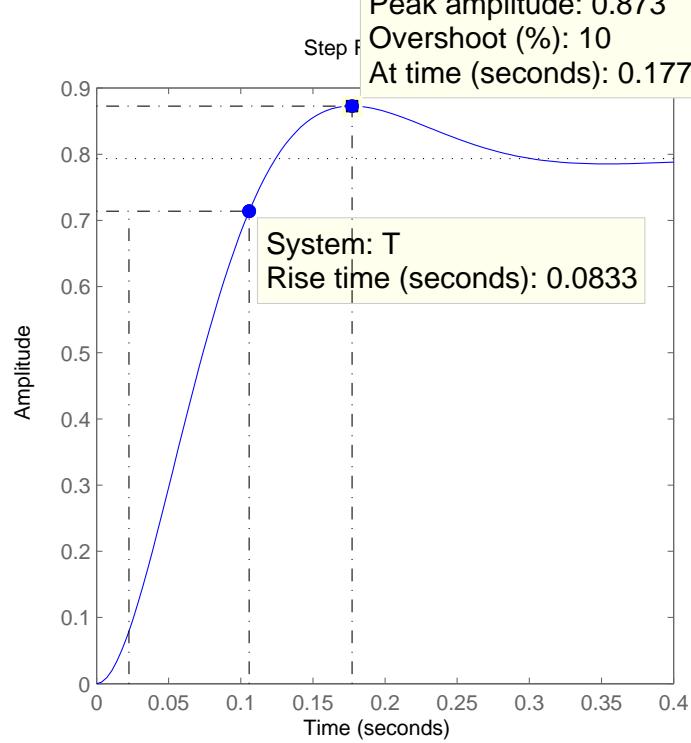
This has closed loop system

$$\frac{T_2(s)}{R(s)} = T(s) = \frac{100}{s^2 + (111 + 100K_d)s + 100(1 + K_p)},$$

and using the code

```
Kp=3.84;
Kd=-0.85;
% Configuration #2 figure(1)
T = tf(100*Kp, [1 111+100*Kd 100*(1+Kp)])
step(T)
```

we get the following plot:



(d) For unit step references

$$e_{ss} = \frac{1}{1 + K_p} \quad \text{where} \quad K_p = \lim_{s \rightarrow 0} \frac{(3.84 - 0.85s)/100}{s^2 + 111s + 100} = 3.84$$

$$e_{ss} = \frac{1}{1 + 3.84} = \frac{1}{4.84} = 0.21 \quad (\text{agrees with plot from part (c)})$$

(e) For 1 degree change in T_b ,

$$\begin{aligned} \frac{T_2(s)}{T_0(s)} &= \frac{\frac{100}{s^2 + 111s + 100}}{1 + \frac{(3.84 - 0.85s)/100}{s^2 + 111s + 100}} = \frac{100}{s^2 + (111 - 85)s + 100 + 3.84} \\ &= \frac{100}{s^2 + 26s + 484} \end{aligned}$$

$$e_{ss} = \lim_{s \rightarrow 0} s \left(\frac{100}{s^2 + 26s + 484} \right) \cdot \frac{1}{s} = \frac{100}{484} = \underline{0.21}$$

8 Resources

8.1 Books

- Gene F. Franklin, J. David Powell and Abbas Emami-Naeini, *Feedback Control of Dynamic Systems*, Pearson
 - 6th and 7th edition: Section 4.3

8.2 Web resources

The following are web resources concerning PID controllers. If you find something else on the web that is useful, or if you find a link that no longer works, please inform your instructor!

- <http://ctms.engin.umich.edu/CTMS/index.php?example=Introduction§ion=ControlPID>
 - An interactive tutorial of PID controllers that is designed to be covered while you have access to MATLAB.
- <https://www.youtube.com/watch?v=XfAt6hNV8XM> A 13 minute video with some examples of PID control.

9 Appendix: Implementing a PID controller on a microcontroller

A PID controller is easily implemented on a micro-controller, and only requires a few lines of code. The main issue is that a micro-controller is a discrete time system, in that it can only read or write in discrete times. Thus, we need to find discrete time approximations to the integral and derivative. The basic functionality is as follows:

Set K_p, K_e, K_i to PID control gains

Set I := 0

Set e_past := 0

Set T_s := 0

Set T_c := current time

Loop:

 Read r, current value of reference variable (supplied by the user, probably input using a user interface)

 Read y, current value of system output

 Calculate e := r-y, the error

 If T_s>0,

 Calculate D := (e - e_past)/T_s, Set e_past := e,

 else

 Set D := 0

 Calculate I := I + T_s*e

 Calculate controller output u = K_p*e + K_i*I + K_d*D

 Set output to u (for example, setting a voltage to u using a digital to analog element)

 Set T_s := current time minus T_c

 Set T_c := current time

End Loop:

The blue line is where the PID control output is calculated. The first time through the loop, only the proportional term will be utilized, as two samples are necessary to calculate the derivative, and the integral of a single sample is zero. In the next subsections, we explain the red and purple lines, which are the discrete time approximations to derivative and integral.

9.1 Approximation to a Derivative

Given a function $e(t)$, the definition of derivative is

$$\frac{de}{dt} = \lim_{h \rightarrow 0} \frac{e(t) - e(t-h)}{h}$$

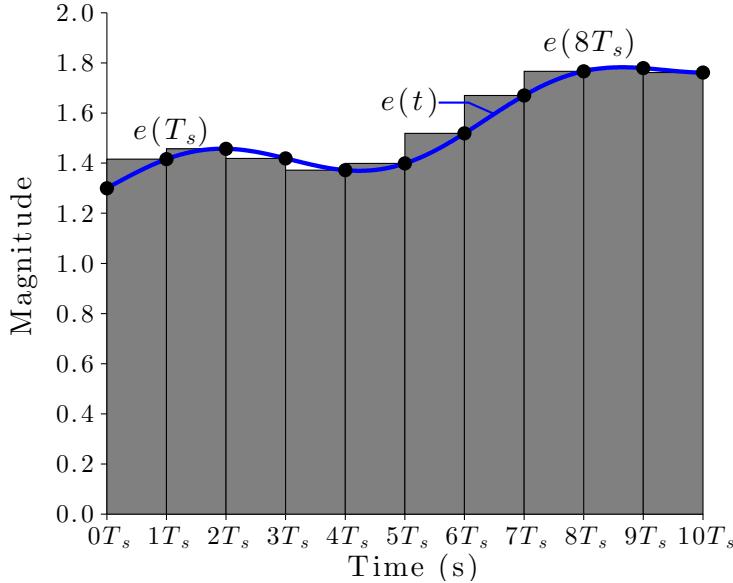
In the micro-controller, we can only calculate $e(t)$ at discrete times as we go through the loop. Our approximation will be to take h to be this cycle time, which is the variable T_s. Thus

$$\frac{de}{dt} \approx \frac{e(t) - e(t - T_s)}{T_s}$$

Note that $e(t - T_s)$ is saved in the variable e_past.

9.2 Approximation to an Integral

The integral of a function is the area under the graph. When we only get discrete samples of the function we can approximate the area using rectangles whose width is the sample time, and height are the samples. This is illustrated below, where the sample time is fixed to T_s seconds.



As an equation, we can define the approximation as follows: Let $i(t)$ be the function that is the integral of $e(t)$ from 0 to t . The exact calculation of $i(t)$ is

$$i(t) = \int_0^t e(\tau) d\tau.$$

Suppose we want to calculate $i(t)$ at sample N , so that $t = NT_s$. The approximation is

$$i(t) = i(NT_s) \approx \sum_{k=1}^N e(kT_s)T_s.$$

Since we keep adding new terms as t increases, we can also calculate this recursively as

$$i(NT_s) = i((N - 1)T_s) + e(NT_s)T_s.$$

9.3 Anti-windup - accounting for actuator saturation

It is very common for the control output $u(t)$ to be only valid over a finite range. For example, if you have a D/A board that can only supply a maximum of ± 10 volts, then u can only lie between -10 and 10. Even if we calculate a larger value of u , only 10 volts will be delivered. In this case, the output is *saturated*. Output saturation can have detrimental effects on the integral control element, as the error will be larger than expected, causing the integral term to *wind-up*, or count up to a very large value. This can be counteracted by limiting the integral term when saturation occurs. The modified code that includes anti-windup is as follows:

Set K_p , K_e , K_i to PID control gains

Set u_{max} to maximum control magnitude

Set $I := 0$

Set $e_{past} := 0$

Set $T_s := 0$

Set $T_c :=$ current time

Loop:

Read r , current value of reference variable (supplied by the user, probably input using a user interface)

Read y , current value of system output

Calculate $e := r - y$, the error

If $T_s > 0$,

```

Calculate D := (e - e_past) / Ts, Set e_past := e,
else
  Set D := 0
Calculate I := I + Ts*e
Calculate controller output u = Kp*e + Ki*I + Kd*D
If abs(u) > umax,
  u := sgn(u) * umax (saturate output)
  I := I - Ts*e (undo integration)
Set output to u (for example, setting a voltage to u using a digital to analog element)
Set Ts := current time minus Tc
Set Tc := current time
End Loop:

```

Note that the new lines are in cyan.