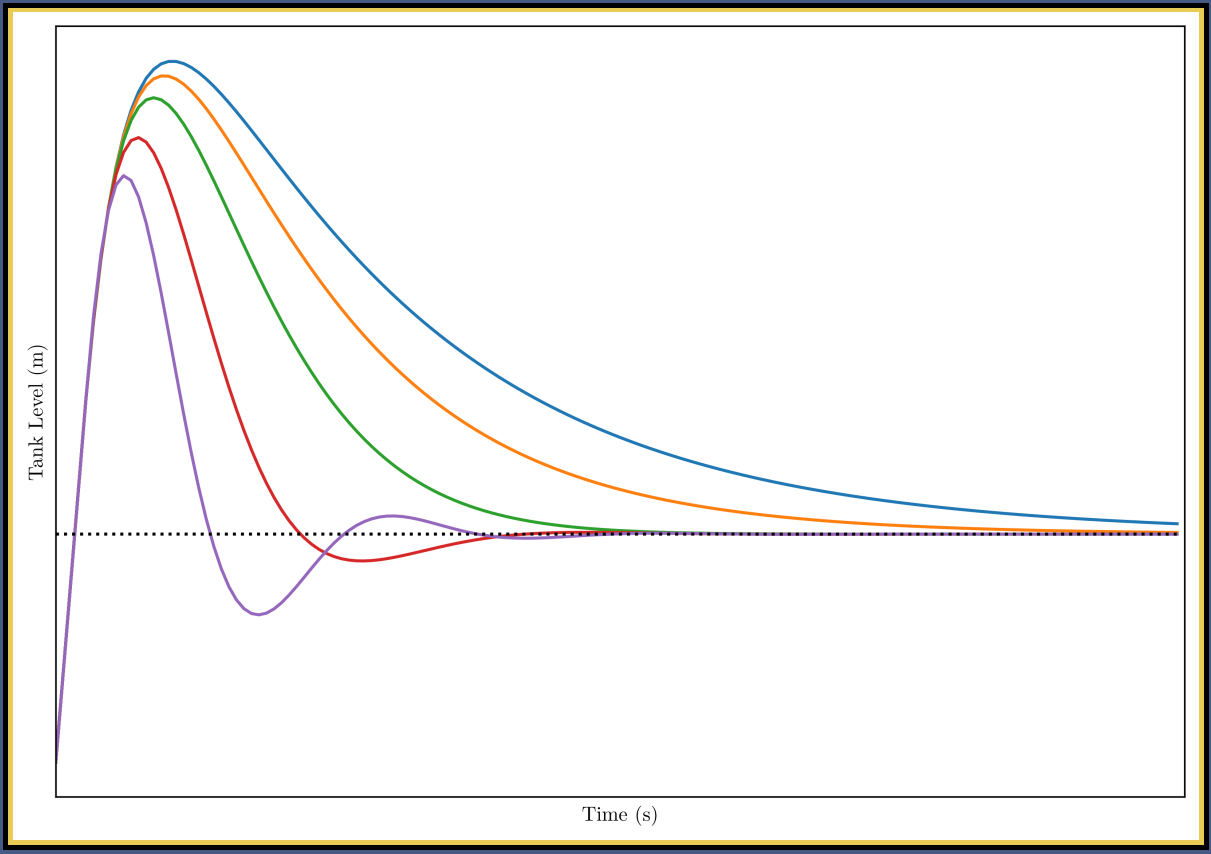
Process Control Simulation

Theory and Implementation

1st Edition



Frank J. Vilimek

Contents

Chapter 1

Introduction and Scope

1.1 Introduction – Process Control is Cool

Process control is an incredibly important field, as it governs how modern plants, fabrication facilities, and factories operate. It makes your car drive, your stove turn on, and the AC run. Process control plays a role in nearly every piece of modern technology. In Chemical Engineering, process control is mostly used in systems involving the transport of fluids – heat exchangers, steam generators, distillation towers, and many more. Some processes require highly specific amounts of fluid to pass through; for example, if too much steam is sent through a heat exchanger, then the cold side might be brought up to a temperature that causes it to flash, increasing the pressure to a dangerous level. Because of the sensitivity of some processes, the tuning for controllers needs to be as optimized as possible. Once a process has the right controllers with optimal tuning, it can run (almost) without a hitch.

1.2 Scope – Aiming for Realism

This document is not so much a textbook as it is a place for me to document my process for developing a process control simulation in Python, but I still feel that I need to introduce the basic theory. I have always thought that process control was an incredibly cool and interesting subject, and I wanted to be able to not only implement the things we learned in class (ECH 157 - Process Dynamics and Control), but also include realistic constraints such as controller saturation and anti-windup techniques. I want this simulation to be impressive. I want it to act like a real system. I want to be able to analyze it with theoretical equations. I want to challenge myself and create something amazing. I will try my best to make the simulation not only accurate, but can scale with whatever equipment I want to use.

1.2.1 Problem Statement and Requirements

A plant contains a set of tanks (reactors, surge tanks, buffer vessels, etc.) and potentially heat exchangers, all of which can be joined together in varying configurations. Each tank can be fitted with any number of controllers, can have as many inlets and outlets as you want, can contain any reaction, and can be fitted with coils or jackets. Each heat exchanger can be sized accordingly, can have any necessary material of construction, and can be in any configuration (i.e. floating head, U-tube, square/triangular pitch, etc.). The design of the plant is up to you; however, there are constraints that must be considered:

1. The process variable must reach the setpoint as quickly as possible
2. The process variable must not overshoot or undershoot the setpoint by more than 5%
3. The process variable must have as short a settling time as possible
4. The controller must take into consideration the physical limitations of the equipment it is controlling (i.e. valves can only open between 0-100%, pumps cannot pump backwards or pump faster than it is rated, etc.)
5. All processes must first be modeled analytically using block diagrams and have a functional solution whenever possible (does not apply if the process is non-linear and cannot be linearized, or if the process is considerably complex)
6. The stability of all processes must be analyzed using any method of your choice
7. The controller must be tested on setpoint tracking and disturbance attenuation
8. Any tuning method used must have accompanying work/calculations

Beyond these constraints, the project is free rein. You do not even need to implement PID control, you can choose to use nonlinear control schemes, MPC, LQR, or any other control methods that you want as long as they are valid within the constraints.

Chapter 2

Process Control Theory and Application

2.1 Setup – Level Control

Let us introduce a problem, and build process control intuition off of that. We will use a problem directly related to this project: level control in a tank, much like the one in figure 2.1

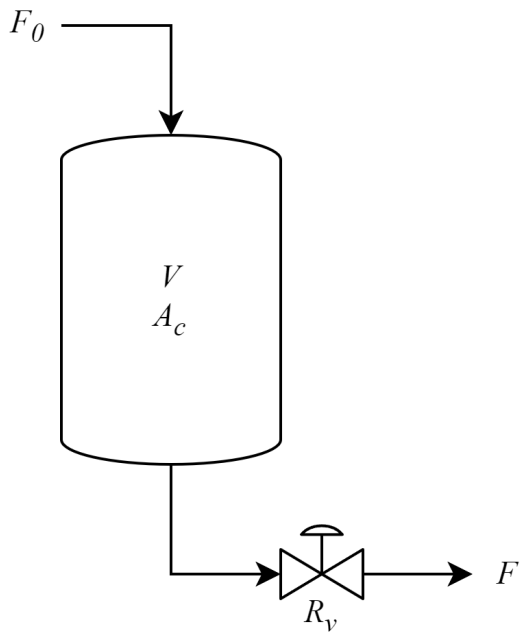


Figure 2.1: A tank with a valve controlling outlet flow rate F.

In this example, a tank with volume V and cross-sectional area A_c is being filled with an inlet liquid stream with volumetric flow rate F_0, and the outlet volumetric flow rate F is

being throttled by a linear valve with resistance R_v. Out of curiosity, you want to measure the liquid level in the tank. To do this, you employ the most famous equation in Chemical Engineering

$$\text{Accumulation} = \text{In} - \text{Out} + \text{Generation} \tag{2.1.1}$$

where "Accumulation" is the change in mass over time, "In" is the mass flow rate into the tank, "Out" is the mass flow rate out of the tank, and "Generation" is zero from the Law of Conservation of Mass

$$\frac{dm}{dt} = \dot{m}_0 - \dot{m} \tag{2.1.2}$$

We can change this mass balance such that it represents volumetric flow and assuming the liquid is incompressible

$$\begin{aligned} \rho\frac{dV}{dt} &= \rho F_0 - \rho F \\ \frac{dV}{dt} &= F_0 - F \\ A_c\frac{dh}{dt} &= F_0 - F \end{aligned} \tag{2.1.3}$$

We can now consider the linear valve, which is described by the expression $F = h/R_v$ (think Ohm's law). Substituting this into eq. (2.1.3) yields

$$\begin{aligned} A_cR_v\frac{dh}{dt} + h &= R_vF_0 \\ \tau\frac{dh}{dt} + h &= KF_0 \end{aligned} \tag{2.1.4}$$

where $\tau = A_cR_v$ and $K = R_v$. τ is called the *time constant* and K is called the *gain*. Equation (2.1.4) is in the form of an ODE that can be solved using an integrating factor

$$\begin{aligned} \tau\frac{dh}{dt}\mu(t) + h\mu(t) &= KF_0\mu(t) \\ \tau\frac{d}{dt}\left[he^{t/\tau}\right] &= KF_0e^{t/\tau} \\ (h - h_0)e^{t/\tau} &= K\Delta F_0\left(e^{t/\tau} - 1\right) \\ h(t) &= K\Delta F_0\left(1 - e^{-t/\tau}\right) + h_0 \end{aligned} \tag{2.1.5}$$

Equation (2.1.5) is the functional solution to our problem. If $\Delta F_0 = 0$, then $h(t) = h_0$; if $\Delta F_0 \neq 0$, then we would see a *step response* as shown in figure 2.2.

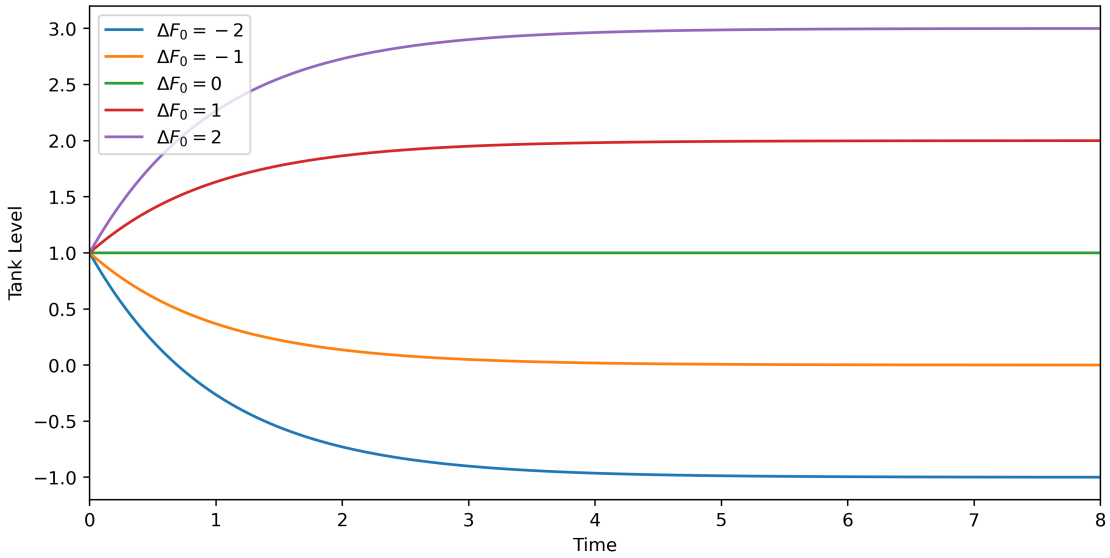


Figure 2.2: Step response for liquid level with varying ΔF_0.

The step response will change in much the same way when the value of K is varied. The value of K can intuitively be thought of as "how far will the response go?" It scales the response, and can change the sign (positive/negative).

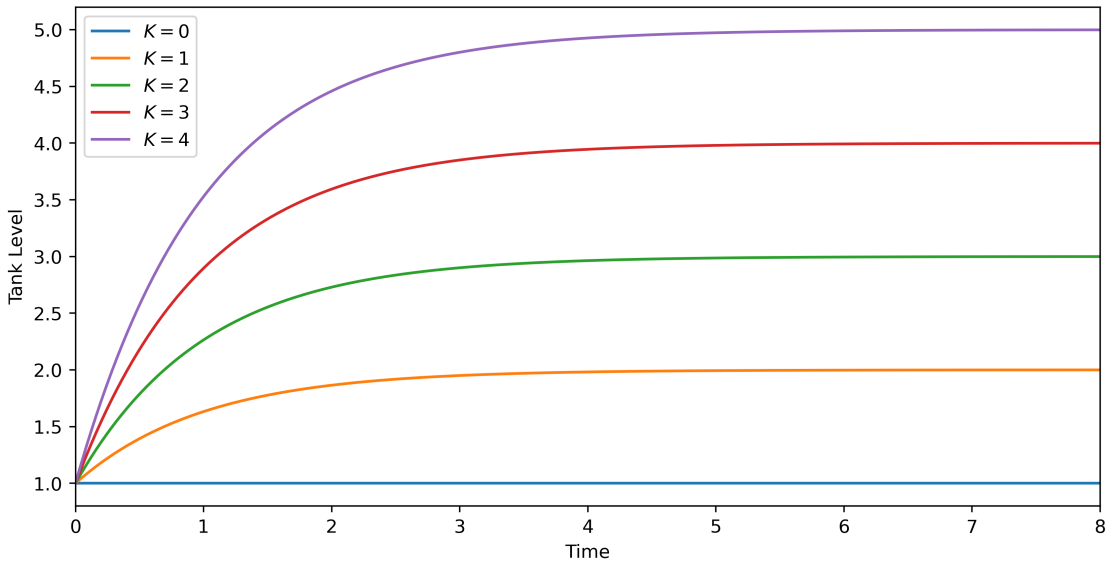


Figure 2.3: Step response for liquid level with varying K.

When the value of τ is changed, the slope at $t = 0$ is changed. The value of τ can intuitively be thought of as "how fast will the response go?" A small τ will make the response faster, and a larger τ will make the response slower.

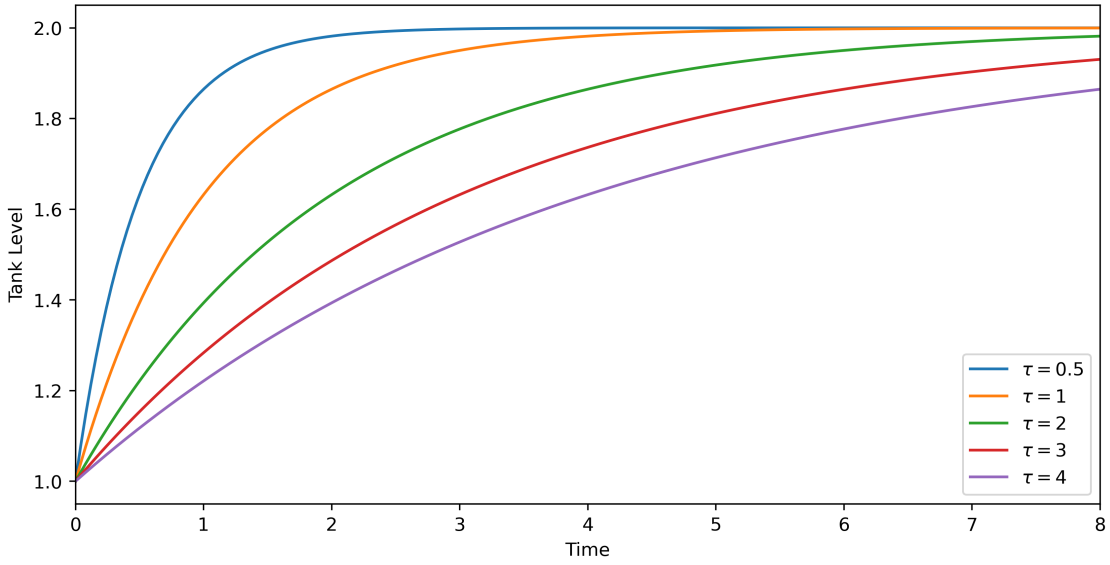


Figure 2.4: Step response for liquid level with varying τ.

The gain, K, does *technically* change how quickly the response increases, however it also changes where it settles after a long time – it stretches and squishes the graph of the response vertically. The time constant, τ, does change how quickly the response increases but does *not* change where the response settles after a long time – it stretches and squishes the graph of the response horizontally.

2.1.1 Block Diagram Representation

The system that we just described seems simple enough: there's an inlet, a tank, and an outlet with a valve. When systems are as simple as this, we can easily envision all the possibilities. But if we added a second tank after the first one, an inlet pump to the first tank, and an outlet valve for the second tank, then the system would become *much* more difficult to describe. To help deal with this, we can instead represent the system using *block diagrams*. The block diagram for our system looks like this

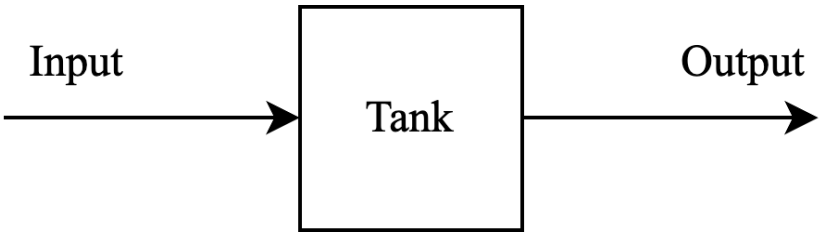


Figure 2.5: Block diagram for the single tank system.

Notice that the tank does not have "inlets" and "outlets." In process control, the system gets abstracted and instead of using system-specific language (because not all control systems will have an inlet and outlet stream), it is better to use system "inputs" and "outputs." An input is any information that is fed to a block in the system, and a block is often a unit operation but can also be controllers, controller logic, or any number of things that can manipulate the input information. An output is any new information that the block had made from the input. From this, it is accurate to say that *a block diagram shows the flow of information* rather than material.

Block diagrams can be quite complicated for certain systems, especially ones that use controllers and advanced logic. Hopefully I can build up to the point of making diagrams as complex as this

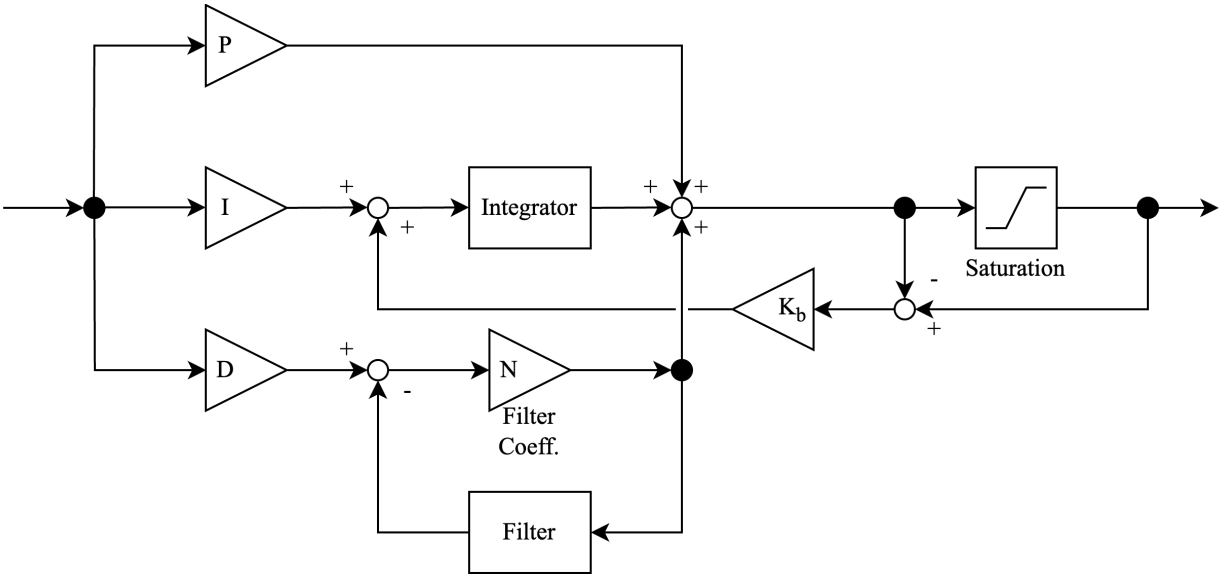


Figure 2.6: Relatively complex block diagram for integral anti-windup techniques.

2.2 Laplace Transforms

For our current system it was not too difficult to find a functional solution, however if we added those tanks and pumps mentioned earlier, finding a solution would be very difficult and time-consuming. To help we can use *Laplace Transforms*, a way of transforming a function from the time-domain to the frequency-domain. The Laplace Transform, denoted as $\mathcal{L}$, of a function $f(t)$ is defined as

$$\mathcal{L}\{f(t)\} = F(s) = \int_0^\infty f(t)e^{-st}dt \tag{2.2.1}$$

The resulting function, $F(s)$, exists in the frequency-domain, which has ties to probability and the frequency that values exist at certain times. Let us do an example calculation for $f(t) = 1$

$$\begin{aligned}\mathcal{L}\{f(t)\} = F(s) &= \int_0^\infty (1)e^{-st}dt \\ &= -\frac{1}{s}\left[e^{-st}\right]_0^\infty \\ &= \frac{1}{s}\left[e^{-s(0)} - e^{-s(\infty)}\right] \\ &= \frac{1}{s}\end{aligned} \tag{2.2.2}$$

This calculation can be done for any number of functions. Table 2.1 shows the most common Laplace Transforms we will use

Table 2.1: Common functions and their Laplace Transforms

$f(t)$	$F(s)$
1	$\frac{1}{s}$
t	$\frac{1}{s^2}$
e^{-at}	$\frac{1}{s+a}$
$\sin(at)$	$\frac{a}{s^2+a^2}$
$\cos(at)$	$\frac{s}{s^2+a^2}$
$\frac{df}{dt}$	$sF(s) - f(0)$
$\frac{d^2f}{dt^2}$	$s^2F(s) - sf(0) - \left.\frac{df}{dt}\right\|_0$

2.2.1 Deviation Variables

Before we can use Laplace Transforms to solve problems, we need to introduce a couple more things, one of them being *deviation variables*. Deviation variables are a remapping of process variables (the variable we are trying to control) defined as

$$f'(t) = f(t) - f_s(t) \tag{2.2.3}$$

where $f'(t)$ is the deviation variable and $f_s(t)$ is the process variable at steady state. This makes it so that

$$f(0) = \begin{cases} f_0,\ t < 0 \\ f_0 + \Delta f_0,\ t \geq 0 \end{cases} \qquad \Rightarrow \qquad f'(0) = \begin{cases} 0,\ t < 0 \\ \Delta f_0,\ t \geq 0 \end{cases}$$

For Laplace Transforms of derivatives, this yields $sf'(0) = 0$, making initial conditions irrelevant to the dynamics of the system. Instead of carrying constants throughout the problem, we can convert all process variables into deviation variables, solve for the functional solution, and then convert back into process variables.

In most cases eq. (2.2.3) does not even need to be used, and as a shortcut the constants in the equation can be erased and all process variables can have a "prime" added to them. To show this, let us use eq. (2.1.4) from before and convert it to deviation variable form. We can begin by taking the original function and subtracting from it the equation when it is at steady state

$$\begin{aligned} \tau\frac{dh}{dt} - \tau\frac{dh_s}{dt} + h - h_s &= KF_0 - KF_{0s} \\ \tau\frac{d(h-h_s)}{dt} + (h - h_s) &= K(F_0 - F_{0s}) \\ \tau\frac{dh'}{dt} + h' &= KF_0' \end{aligned} \tag{2.2.4}$$

which is exactly what we would have gotten if we did the shortcut method.

2.2.2 Transfer Functions

The last thing we need to introduce is *transfer functions*. Transfer functions are the basis of process control calculations, and make it much easier to link together many pieces of equipment together – our two-tank, two-valve, and pump system from earlier can be done much more simply by using transfer functions instead of using typical ODE solving techniques. A transfer function $G(s)$ is defined as

$$G(s) = \frac{\text{Output}(s)}{\text{Input}(s)} \tag{2.2.5}$$

where the Input(s) and Output(s) are the system input and output, respectively, after taking their Laplace transforms. To find the output of the system, multiply $G(s)$ by the input to the system

$$\text{Output}(s) = G(s)\text{Input}(s) \tag{2.2.6}$$

It seems redundant at this point because we only have one transfer function, but its usefulness becomes more apparent when transfer functions are linked together. One transfer function describes one "action" in a system. The change in inlet flow rate changes the liquid level, and so this needs one transfer function. The valve at the outlet of the tank also changes the liquid level, so that needs a transfer function too.

When equipment is linked together, their transfer functions are also linked. Adding equipment to a system results in their transfer functions multiplying or dividing each other, depending on what they do and where they are added. For example, in the system where one tank feeds another, the system's transfer function needs to account for: the effect that the inlet has on the first tank's level; the effect of the first tank's level on the first tank's outlet flow rate; the effect of the first tank's outlet flow rate on the second tank's level; and finally the effect of the second tank's level on the second tank's outlet flow rate – there needs to be four total transfer functions

$$G_{sys}(s) = \frac{h_2(s)}{F_0(s)} = \left[\frac{h_1(s)}{F_0(s)}\right]\left[\frac{F_1(s)}{h_1(s)}\right]\left[\frac{h_2(s)}{F_1(s)}\right]\left[\frac{F_2(s)}{h_2(s)}\right] \tag{2.2.7}$$

To solve for $h_2(t)$, you would need to multiply $G_{sys}(s)$ by $F_0(s)$ and take the inverse Laplace Transform

$$h_2(t) = \mathcal{L}^{-1}\left\{F_0(s)G_{sys}(s)\right\} \tag{2.2.8}$$

Taking the Inverse Laplace Transform requires looking Table 2.1 and seeing which transforms fit. Often times the transfer function needs to be manipulated using partial fraction decomposition to find the right transform.

2.3 Putting it All Together

Let us put Laplace transforms, deviation variables, and transfer functions together and solve the two tank system. An inlet pipe has volumetric flow rate F_0 and feeds into Tank 1 that has a volume V_1 and cross-sectional area A_{c1}. The outlet of Tank 1 has a linear valve with

resistance R_1 that feeds into Tank 2, which has a volume of V_2 and cross-sectional area A_{c2}. The outlet of Tank 2 also has a linear valve, with resistance R_2. We want to know how the liquid level in Tank 2 changes when F_0 experiences a step change.

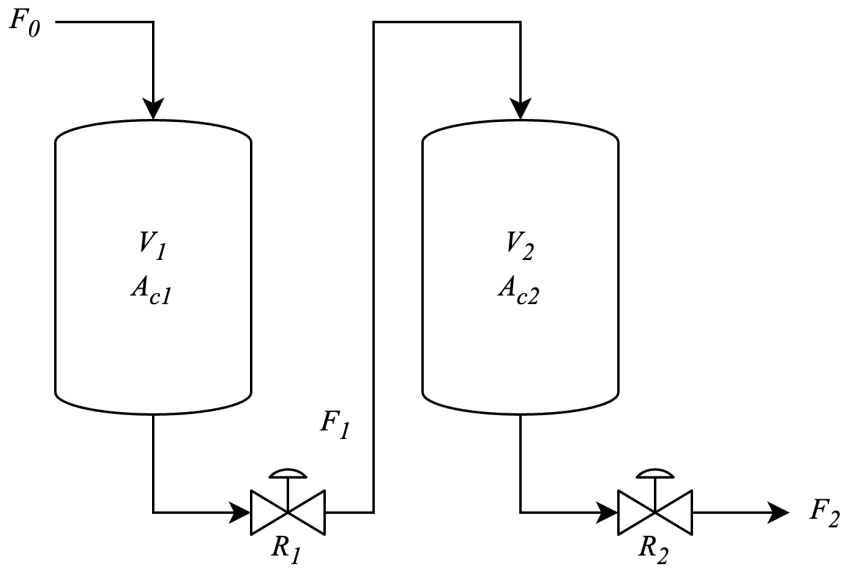


Figure 2.7: Two tank cascading system.

First, let us recognize that each tank will have their own mass balance

$$\frac{dm_1}{dt} = \dot{m}_{01} - \dot{m}_1 \tag{2.3.1}$$

$$\frac{dm_2}{dt} = \dot{m}_{02} - \dot{m}_2 \tag{2.3.2}$$

Like in eq. (2.1.4), let us convert our mass balances so that they describe liquid levels

$$\tau_1 \frac{dh_1}{dt} + h_1 = K_1 F_0 \tag{2.3.3}$$

$$\tau_2 \frac{dh_2}{dt} + h_2 = K_2 F_1 \tag{2.3.4}$$

where $\tau_1 = A_{c1}R_1$, $\tau_2 = A_{c2}R_2$, $K_1 = R_1$, and $K_2 = R_2$. Let us convert the equations into deviation variable form

$$\tau_1 \frac{dh_1'}{dt} + h_1' = K_1 F_0' \tag{2.3.5}$$

$$\tau_2 \frac{dh_2'}{dt} + h_2' = K_2 F_1' \tag{2.3.6}$$

Focusing on eq. (2.3.5), let us take the Laplace Transform

$$\begin{aligned}\tau_1\left[sH_1'(s) - \cancel{h_1'(0)}\right] + H_1'(s) &= K_1F_0'(s)\\ \tau_1 sH_1'(s) + H_1'(s) &= K_1F_0'(s)\\ H_1'(s)\left[\tau_1 s + 1\right] &= K_1F_0'(s)\end{aligned}$$

We can now make the transfer function $G_1(s)$ describing the effect of changing F_0 on $h_1(t)$

$$G_1(s) = \frac{H_1'(s)}{F_0'(s)} = \frac{K_1}{\tau_1 s + 1} \tag{2.3.7}$$

A transfer function of this form is first-order, as the denominator is a first-order polynomial with respect to s. We can also make the transfer function $G_2(s)$ that describes the effect of changing $h_1(t)$ on F_1

$$\begin{aligned}F_1 &= \frac{h_1(t)}{R_1}\\ F_1' &= \frac{h_1'(t)}{R_1}\\ \mathcal{L}\left\{F_1'\right\} &= \mathcal{L}\left\{\frac{h_1'(t)}{R_1}\right\}\\ F_1'(s) &= \frac{H_1'(s)}{R_1}\\ \Rightarrow G_2(s) &= \frac{F_1'(s)}{H_1'(s)} = \frac{1}{R_1} = \frac{1}{K_1}\end{aligned} \tag{2.3.8}$$

We can continue this calculation to get $G_3(s)$, the effect of changing F_1 on $h_2(t)$, and $G_4(s)$, the effect of changing $h_2(t)$ on F_2

$$G_3(s) = \frac{H_2'(s)}{F_1'(s)} = \frac{K_2}{\tau_2 s + 1} \quad \text{and} \quad G_4(s) = \frac{F_2'(s)}{H_2'(s)} = \frac{1}{K_2} \tag{2.3.9}$$

The system transfer function is thus

$$\begin{aligned}G_{sys}(s) &= G_1(s)G_2(s)G_3(s)G_4(s)\\ &= \left[\frac{K_1}{\tau_1 s + 1}\right]\left[\frac{1}{K_1}\right]\left[\frac{K_2}{\tau_2 s + 1}\right]\left[\frac{1}{K_2}\right]\\ &= \frac{1}{(\tau_1 s + 1)(\tau_2 s + 1)}\end{aligned}$$

Remember that the system transfer function is the output over the input – to solve for the

output, we need to multiply the transfer function by the input

$$G_{sys}(s) = \frac{H_2'(s)}{F_0'(s)} = \frac{1}{(\tau_1 s + 1)(\tau_2 s + 1)}$$
$$H_2'(s) = G_{sys}(s)F_0'(s) = F_0'(s)\frac{1}{(\tau_1 s + 1)(\tau_2 s + 1)}$$

Also remember from section 2.2.1, $F_0' = \Delta F_0$, so $F_0'(s) = \Delta F_0/s$

$$H_2'(s) = \frac{\Delta F_0}{s(\tau_1 s + 1)(\tau_2 s + 1)} \tag{2.3.10}$$

In order to easily take the inverse Laplace Transform of eq. (2.3.10), we need to perform partial fraction decomposition. Doing so yields

$$H_2'(s) = \Delta F_0 \left[\frac{1}{s} + \frac{1}{\tau_2 - \tau_1}\left(\frac{\tau_1}{\tau_1 s + 1} - \frac{\tau_2}{\tau_2 s + 1}\right)\right] \tag{2.3.11}$$

Now that everything is separated, we can use Table 2.1 to perform the inverse transform and yield $h_2'(t)$

$$\mathcal{L}^{-1}\left\{H_2'(s)\right\} = \mathcal{L}^{-1}\left\{\Delta F_0 \left[\frac{1}{s} + \frac{1}{\tau_2 - \tau_1}\left(\frac{\tau_1}{\tau_1 s + 1} - \frac{\tau_2}{\tau_2 s + 1}\right)\right]\right\}$$
$$h_2'(t) = \Delta F_0 \left[1 + \frac{1}{\tau_2 - \tau_1}\left(\tau_1 e^{-t/\tau_1} - \tau_2 e^{-t/\tau_2}\right)\right] \tag{2.3.12}$$

And that is our functional solution in deviation variables. The dynamic response of the level in Tank 2 is shown in figure 2.8 and compared with the dynamic response of Tank 1. Notice that $h_2'(t)$ has no K_1 or K_2 gains. We would expect this because the valves are not changing. Whatever magnitude the step increase Tank 1 experiences, Tank 2 will experience the same magnitude. Tank 2's gain is the same as Tank 1's.

Also notice how the response for Tank 2 is sluggish compared to Tank 1 – this is because Tank 2 had a second-order transfer function, meaning that the information fed to it was already processed by Tank 1. While Tank 1 experienced an immediate sudden jump in flow rate, Tank 2 experienced a more gradual increase in flow rate, causing it to lag behind. This will be true for all systems with an order greater than 1, and the higher the order, the more sluggish the last element's response will be.

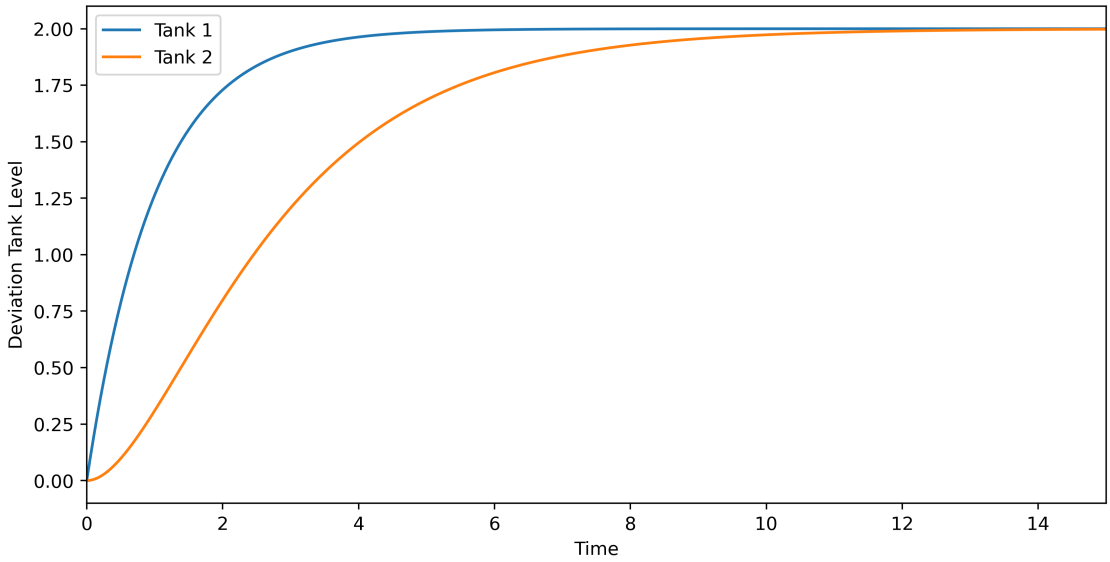


Figure 2.8: The dynamic step response of Tank 1 and Tank 2.

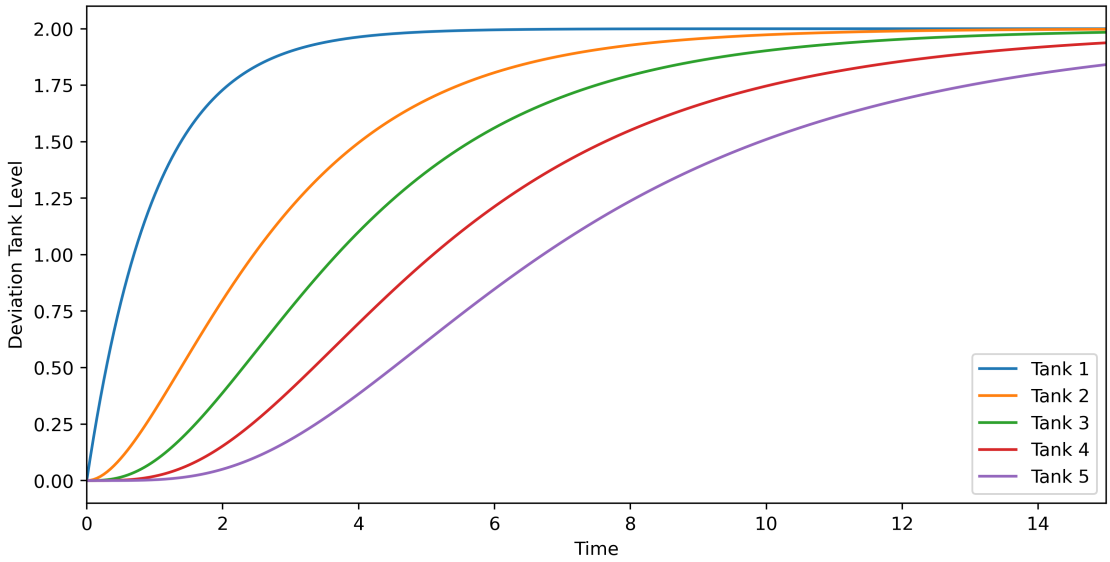


Figure 2.9: The dynamic step response of five cascading tanks.

Chapter 3

Controllers

3.1 Controller Loops

At the heart of process control are controller loops. Controller loops, such as the one in figure 3.1, shows that all input information gets analyzed by some controller logic, the information gets modified by some process, then the process output information is looped back to the beginning for the controller logic to re-analyze and determine if something needs to happen

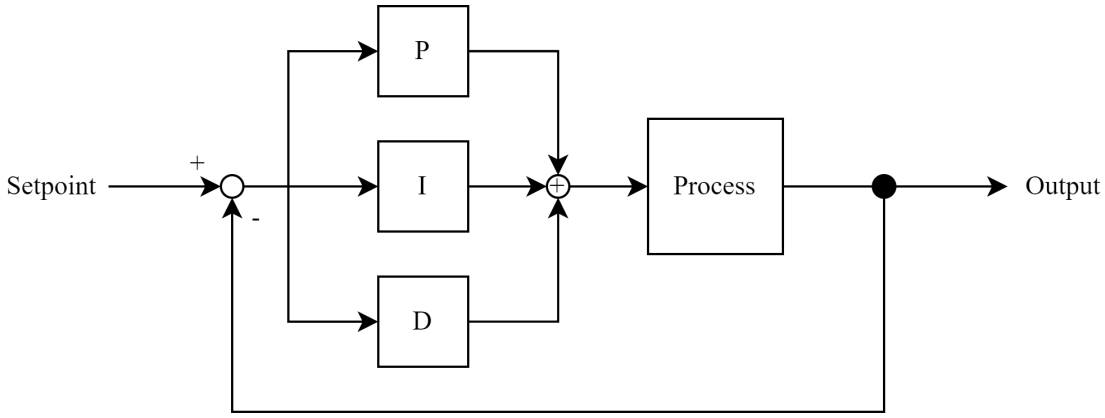


Figure 3.1: Block diagram of a system with a feedback PID controller scheme.

This is what causes process control to become *dynamic*; information gets analyzed and processed, a change is made by a controller, and the process outputs new information. The controller takes this information again and analyzes and processes it again, another change is made, and the process outputs more new information. The controller essentially has to mitigate changes that it made itself, and can totally change the way the system acts. If a controller is grossly improperly tuned, it could lead to catastrophic effects in a plant.

3.1.1 Feedback Control

Feedback control is perhaps the most widely used controller scheme in industry, and figure 3.1 is the block diagram for feedback control. A sensor measures some process variable, applies some control upstream of the sensor in the block diagram, then measures the process variable again to see if more control needs to be applied.

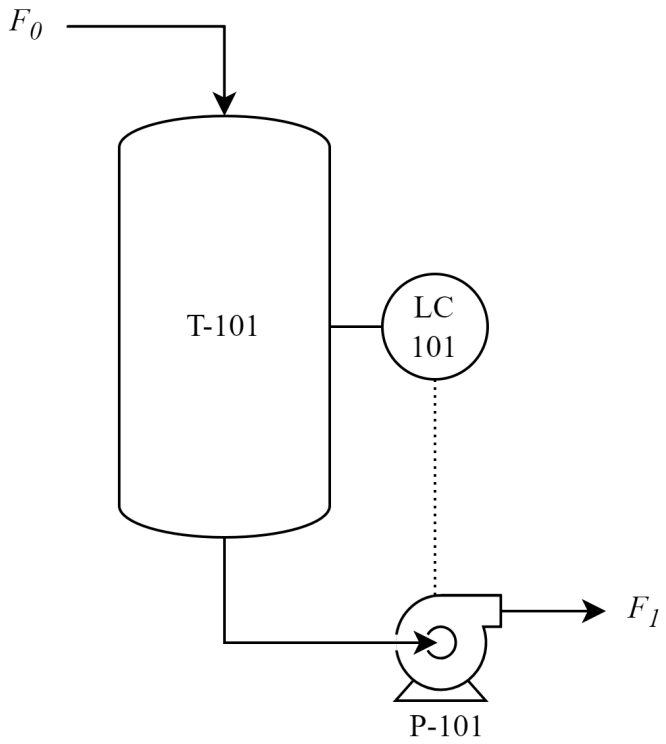


Figure 3.2: A surge tank with a feedback control loop for level control.

Figure 3.2 above shows a simple feedback control scheme for level control in a surge tank. If the inlet flow F_0 experiences a disturbance and affects the liquid level, the level controller LC-101 will measure the disturbance, communicate the necessary response to keep the liquid level at the setpoint to the pump P-101, which changes the flow rate F_1 to change the level.

Feedback control has pros and cons. The pros are that feedback control is corrective: it measures a change in a process variable directly and acts accordingly. Disturbances can come from anywhere, but as long as it changes the process variable, the feedback controller will correct it. Feedback control, if tuned properly, can also be very quick and accurate, and there can be very minimal setpoint overshoot or undershoot.

The cons to feedback control is that it is reactive: the process variable needs to experience a disturbance before the controller can correct it. This might not be the biggest deal for most processes, but can be detrimental for those that require extreme precision. Because it is reactive, another con is that feedback control does not handle time delays well, and can introduce extreme instability in systems with large time delays. Lastly, feedback control (specifically for PID controllers) is somewhat limited to linear systems; for systems with nonlinear dynamics (like some reactive systems or systems with phase changes), the model needs to be linearized to get the proper tuning parameters, and even then the controller may be inaccurate.

3.1.2 Closed-Loop Transfer Functions and FVT

Now that the concept of controllers has been introduced, we can consider how they behave in closed loops. Remember that closed loops are information loops in block diagrams that show input information being analyzed by equipment logic, going through a process, and the output being fed back to the logic to figure out if control needs to be applied. The block diagram for a general closed loop system is shown in figure 3.3.

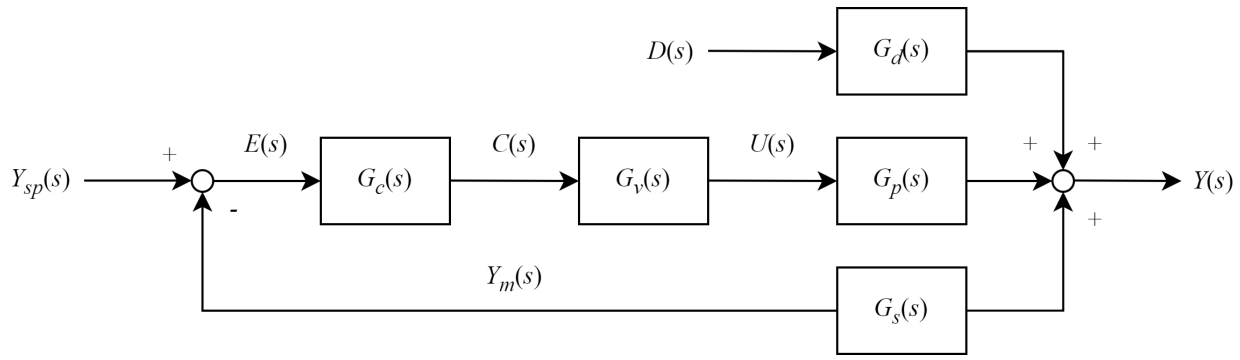


Figure 3.3: Block diagram for a closed loop control system

where $Y_{sp}(s)$ and $Y(s)$ are the Laplace Transforms of the system input and output, respectively. Each block represents a different aspect of either the process or the controller: $G_c(s)$ describes the dynamics of the controller itself; $G_v(s)$ describes the dynamics of the final element (i.e. valves, pumps, etc.); $G_p(s)$ describes the dynamics of the process; $G_d(s)$ describes the dynamics of the disturbance; and $G_m(s)$ describes the dynamics of sensors and transmitters.

3.2 Proportional, Integral, Derivative (PID) Control

Proportional, integral, derivative (PID) control is the most widely used method for feedback control in industry. As the name implies, it consists of three parts, or *modes*: a proportional mode, which applies control that is linearly proportional to the error; an integral mode, which applies control according to the integral of the error; and a derivative mode, which applies control according to the derivative of the error. Each mode has an associated gain and/or time constant that can be tuned accordingly – if you need a lot of proportional control, you can increase the proportional gain; if you need quicker integral control, you can decrease the integral time constant, and so on. PID controllers typically operate on the ideal PID equation

$$C(t) = C_0 + K_p \left[e(t) + \frac{1}{T_i} \int_0^t e(t)dt + T_d \frac{de(t)}{dt} \right] \tag{3.2.1}$$

where $C(t)$ is the applied control, C_0 is the controller bias, K_p is the proportional gain, $e(t)$ is the error at time t, T_i is the integral time constant, and T_d is the derivative time constant. Some sources substitute $K_i = K_p/T_i$ and $K_d = K_p T_d$, but they are essentially the same. Figure 3.4 shows an example of a (badly-tuned) PID controller attempting to maintain the level setpoint in a tank after a setpoint change on top, and the actual controller action on the bottom. Comparing the top plot to figure 2.3, we can see that the addition of a controller can significantly change the dynamics of a system. Normally the system would experience a step change and nothing else, but the controller introduces oscillations before eventually settling down to the setpoint.

The shaded regions in the liquid level plot show the portions where the integral is positive or negative, indicating when integral controls acts with the highest magnitude. Notice also that the controller action plot does not settle to zero – though the error settles to zero, the controller needs to apply a non-zero amount of control (in this case, a pump at the tank outlet) to deal with the step disturbance. If the inlet flow rate increases, the controller needs to increase the pump speed to match the inlet flow rate; if the controller action settled to zero, the pump would go back to its original speed and the liquid level would deviate from the setpoint. The properties of each controller mode is described in the next sections.

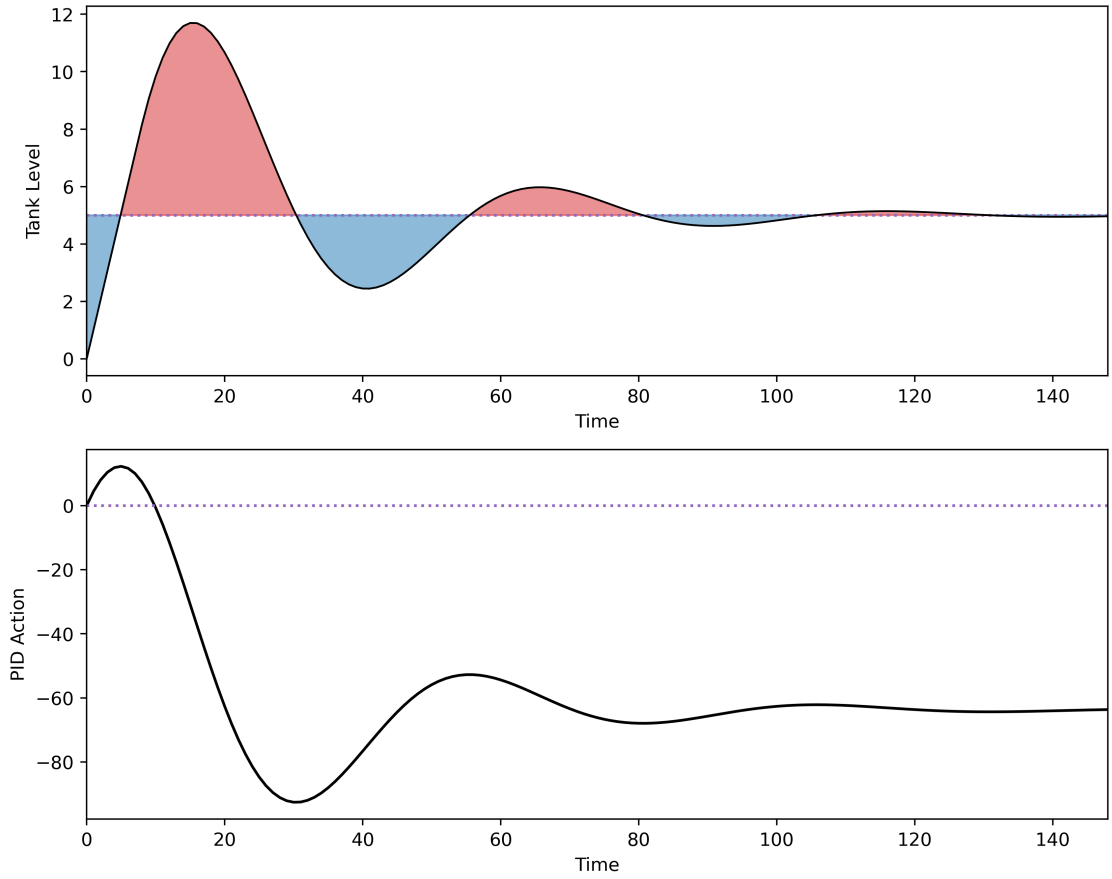


Figure 3.4: (Top) Liquid level with PID control applied. (Bottom) PID controller action.

3.2.1 Proportional Control

Proportional (P) control is a mode that applies control that is proportional to the error. It is described by the equation

$$C(t) = C_0 + K_p e(t) \tag{3.2.2}$$

where $e(t)$ is the error between the setpoint and the process variable $e(t) = y_{sp} - y_{pv}$. A process that is controlled using only P control has pros and cons: the pro is that P-only control acts strongly the instant there is error between the setpoint and process variable; the con is that the response will never reach the setpoint. This can be seen in figure 3.2.2 below. We can show that P-only control will never reach the setpoint

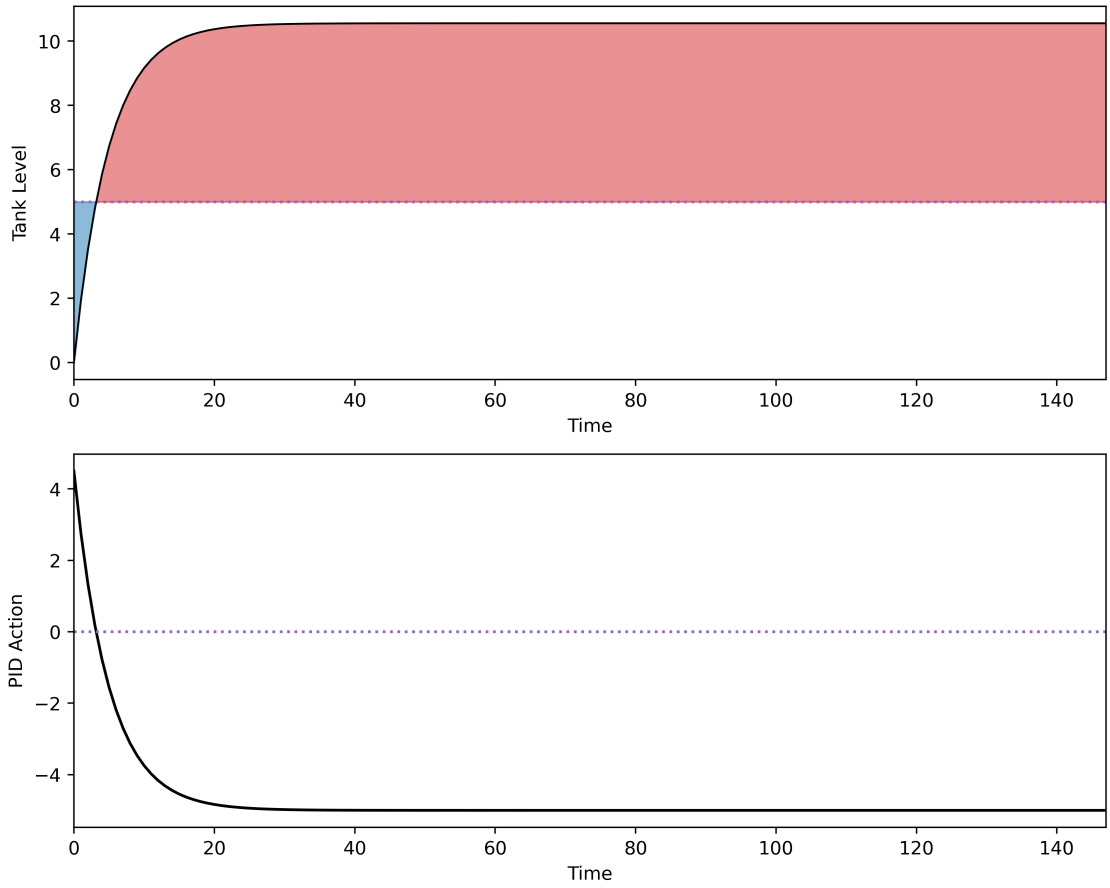


Figure 3.5: Liquid level with P-only control.

3.2.2 Integral Control

Integral (I) control is a mode that applies control according to the integral of the error. It is described by the equation

$$C(t) = C_0 + \frac{1}{T_i}\int_0^t e(t)dt \tag{3.2.3}$$

A process that is controlled only by I control has pros and cons: the pro is that I-only control will always tend to be around the setpoint; the con is that I-only control will cause its own oscillations. This is because of a phenomena called *integral windup*. Since the I controller acts according to the magnitude of the integral, the controller will over/undershoot the setpoint in an attempt to drive the integral to zero. This first over/undershoot will result in a second over/undershoot, then a third and so on, creating oscillations. Figure 3.6 shows the oscillations, as well as the integral windup being unwound (blue regions) and correcting for the over/undershoot from unwinding (red regions).

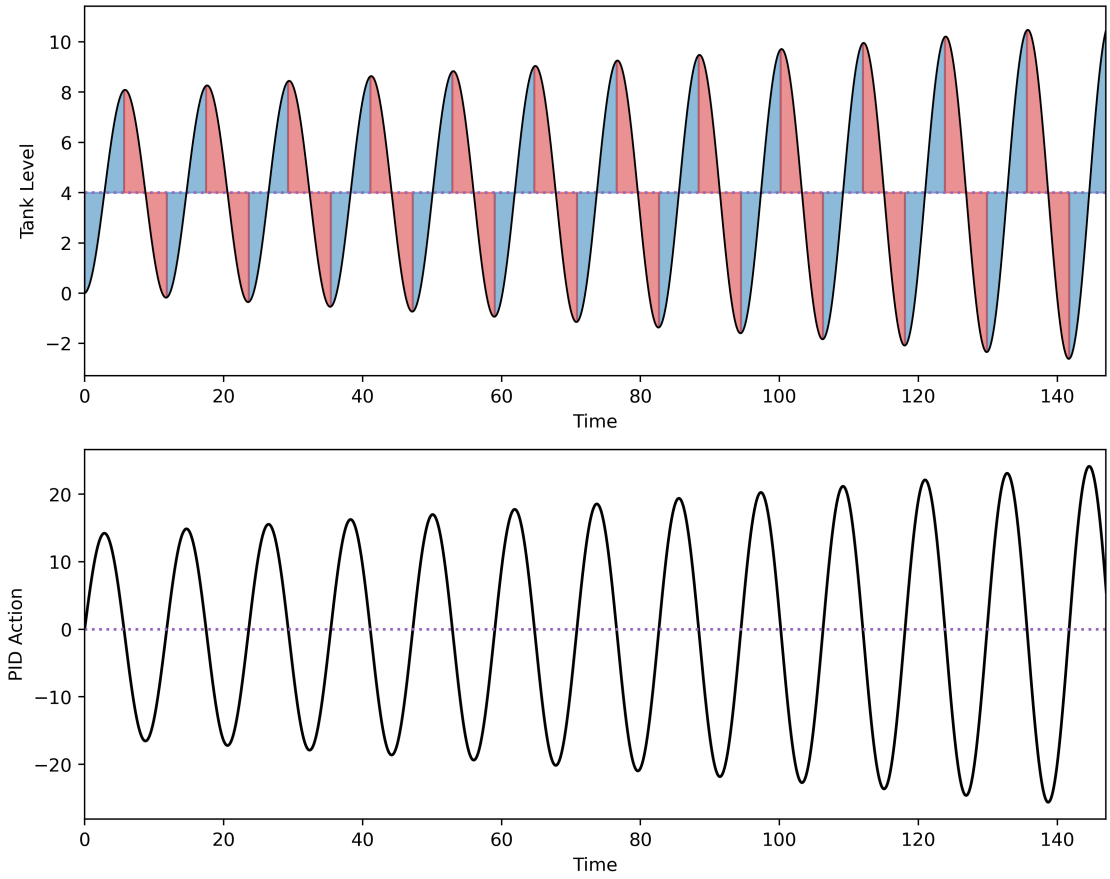


Figure 3.6: Liquid level with I-only control.

3.2.3 Derivative Control

Derivative control is a mode that applies control according to the derivative of the error. It is described by the equation

$$C(t) = C_0 + T_d \frac{de(t)}{dt} \tag{3.2.4}$$

A process that is controlled only by D control has pros and cons: the pro is that D-only control can somewhat predict what the error will be and how much control to apply. This predictive property can often times quicken the controller response. The cons to D-only control are that it does not naturally tend towards the setpoint – since the controller only acts based on the error's rate of change, it will not necessarily drive the error to zero. If the error has a constant slope, for example, then a D-only controller will apply a constant level of control, even if the process variable deviates from the setpoint; this phenomena can be seen in figure 3.7. Another con to D-only control is that it amplifies noisy data – if sensor

data has noise, D-only control can potentially lead to instabilities. Since the controller now has to correct for these instabilities, the overall controller response can even become slower.

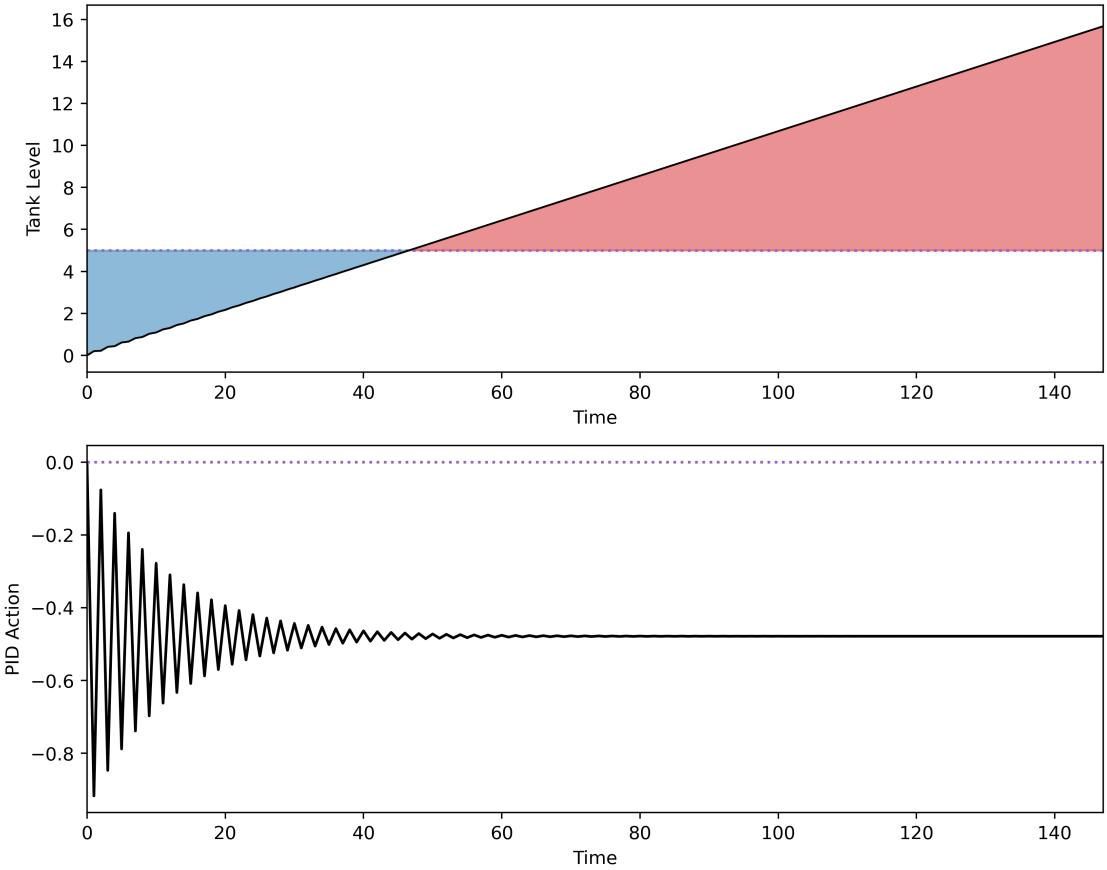


Figure 3.7: Liquid level with D-only control.

3.2.4 Many Combinations to Choose From

P-only, I-only, and D-only control all have their pros and cons, but combined together in the right way, they can provide very quick and accurate control.

Chapter 4

Developing the Simulation

4.1 Base Python Script

Now that we have sufficiently developed the theory behind process control, let us begin implementing it all in a simulation. Of course, we could all buy a license for MATLAB and use Simulink, but where is the fun in that? I want to be able to code my own simulation from scratch. Again, this simulation is for controlling liquid level in a tank.

Since I am most familiar with Python, the simulation will be written in Python as I know it. It might not be the most optimal way of doing things, but it gets the job done. First, we will begin by importing our libraries – you only really need `numpy` and `matplotlib`. Let us also initialize our tank dimensions (you can make them whatever you want).

```
# Import libraries
import numpy                as np
import matplotlib.pyplot as plt

# Tank dimensions and simulation time
Diameter = 3
Height   = 5
Ac       = np.pi * Diameter**2 / 4
Volume   = Ac * Height
MinLevel = 0
Time     = 150
```

Let us now initialize our controller and equipment parameters – their gains, time constants, pump speeds, etc. Not all of it will be used immediately (or even at the same time), but they could be handy later on.

```
# Feedback controller parameters
C0          = 0
Ti          = 5
Td          = 0.1
Kp          = 3
Ki          = Kp / Ti
Kd          = -Kp * Td
CtrlParams = (C0, Kp, Ki, Kd)
```

The `Realism properties` include things such as `WindupMitigation` which chooses a method for dealing with integral windup – for now it is set to 0, but will eventually have 4 windup mitigation methods. The properties also cap the outlet pump speed with `MaxPumpSpeed`; set a maximum integral value with `MaxIntegral` to work with one of the windup mitigation methods; delete old integral data once it reaches a value set by `ResetValue`; define the length of a moving window over which the integral portion integrates with `IntegralWindow`; and finally define the width of a dead band, a small region above and below the setpoint where the controller does not act, with `DeadBand`.

We must now initialize our setpoints and solution arrays – I like to use arrays and lists, and this method works for me. I wanted to use arrays for the setpoints rather than a single value because I wanted to be able to change it over time, and an array would allow me to easily store all the setpoint data in one object.

```
# Setpoints
FlowSP          = 0.4
F0              = 1
L0              = Height / 2
LspArray        = np.zeros(Time)
FspArray        = np.zeros(Time)
NumOfSetPoints = 3

# Randomly change setpoints and flow rates over time
F_in      = np.zeros(Time)
F_out     = np.zeros(Time)
Level     = np.zeros(Time)
Level[0] = L0

for i in range(NumOfSetPoints):
  RandomScalar = np.random.rand(1)
  for j in range(Time):
    if (Time * i / NumOfSetPoints) <= j <\
    (Time * (i + 1) / NumOfSetPoints):
      LspArray[j] = Height * RandomScalar
      F_in[j]      = F0
```

Inside the nested `for` loop I added the ability for the simulation to randomly choose a scalar to multiply the original setpoint `L0` by for each of the `NumOfSetPoints`. There is also the ability to change the inlet flow rate `F_in` along with the setpoint, but I kept it constant.

Next we can add a callable function for the tank's liquid level given some initial level `L0`, an inlet volumetric flow rate `F_in`, and an outlet volumetric flow rate `F_out`.

```
# Tank level function
def TankLevel(L0, F_in, F_out):
  L = L0 + (F_in + F_out) / Ac
  return L
```

Next comes the `PID` class that yields the controller action.

```
# PID controller class
class PID:
  def __init__(self, C0, Kp, Ki, Kd, Data, Setpoint):

    # Error list
    error = []

    # Derivative control
    for i in range(len(Setpoint)):
      error.append(Setpoint[i] - Data[i])
      if i >= 1:
        derivative = error[-1] - error[-2]
      else:
        derivative = 0

    integral = np.trapz(error)

    # PID controller function
    PIDc = C0 + Kp * error[-1] + Ki * integral +\
           Kd * derivative
    self.action = PIDc
```

The PID class holds many features, so let's go through it. The class accepts values for `C0`, `Kp`, `Ki`, `Kd`, `Data`, `Setpoint`, `WindupMitigation`, `DeadBand`, and `PumpSpeed`. The first four are controller parameters. `Data` is an array or list of liquid level data. `Setpoint` is a list or array of setpoint data. `WindupMitigation` is a parameter that determines which windup mitigation method to use: (0) is no mitigation method; (1) is the "Maximum Integral" method, which caps the integral at a certain point to reduce the size of over/undershoots; (2) is the "Reset Error" method, which deletes all stored error values when the error gets below a certain value to help quicken the integral response; and (3) is the "Moving Window" method, which limits the integral to a moving window of specified width.

With our PID class, we can now begin calculating the liquid level over time. For simplicity,

the outlet flow rate will be equal to the controller output and they will not have their own dynamics (in other words, $G_c(s) = G_v(s) = 1$). And of course you can plot the response.

```
# Loop over data points for control response
for i in range(Time - 1):
  # PID control level
  F_out[i] = PID(*CtrlParams, Level, LspArray[:i + 1]).action
  Level[i + 1] = TankLevel(Level[i], F_in[i], F_out[i])

# Plot results
plt.figure(figsize = (10, 5))
plt.plot(LspArray, color = "tab:purple", linestyle = ":", \
        label = "Setpoint")
plt.plot(Level, color = "k", label = "Tank Level Response")
plt.xlabel("Time")
plt.ylabel("Tank Level")
plt.xlim(xmin = 0, xmax = Time)
plt.legend()
plt.show()
```

Using this exact code, a possible resulting plot is shown below in figure 4.1. The setpoints for this system are $h_{sp1} = 3.74550838$, $h_{sp2} = 0.81473881$, and $h_{sp3} = 4.02921868$.

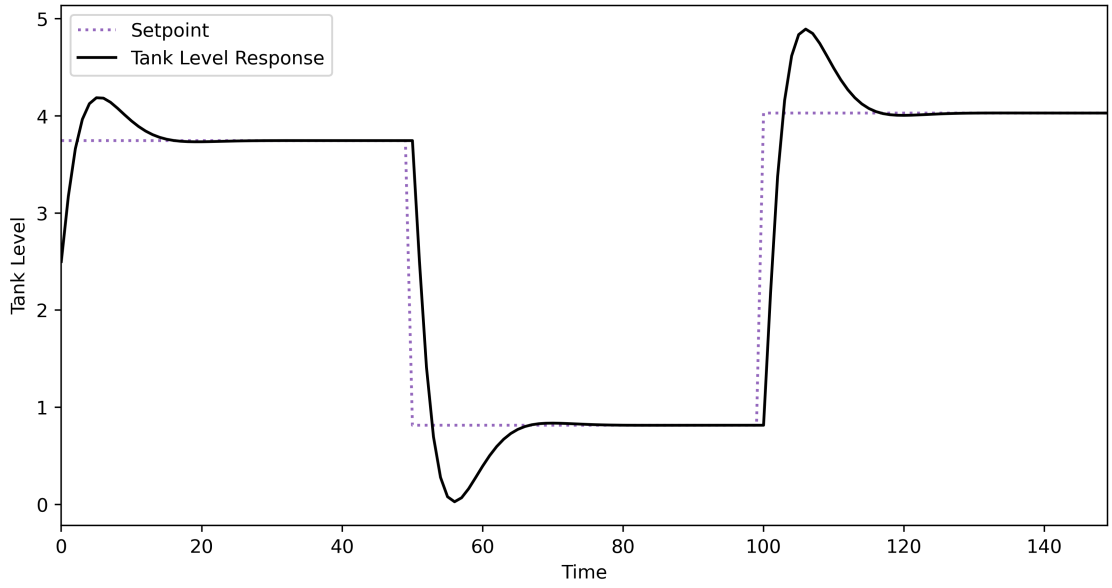


Figure 4.1: Figure produced by base Python code.

This plot shows us a lot of things, one of them being that the controller is not tuned very well. Beyond that, however, the controller does appear to be driving the liquid level towards the setpoints. Something that we do need to fix is that, though not shown, the liquid level can drop to a negative value or rise higher than the tank is tall. Before we fix that, though, let's analyze our current system and see if it matches what we would expect.

4.1.1 Verifying Simulation with Theory

We begin by considering the block diagram to ensure that the system we simulated is the system we will analyze. The block diagram is shown below.

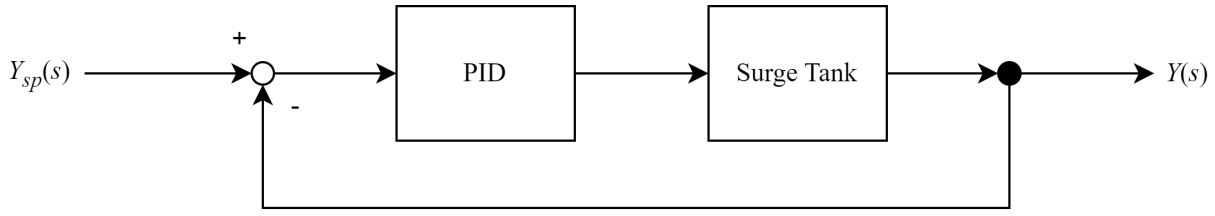


Figure 4.2: Block diagram for this simulation.

This block diagram does seem to match our simulation – there is only one tank, a PID controller, and a feedback loop. We can now consider the setpoint tracking transfer function

$$\frac{H'(s)}{H'_{sp}(s)} = \frac{G_p G_c G_f G_m}{1 + G_p G_c G_f G_m} \tag{4.1.1}$$

where we will set $G_f = G_m = 1$ for simplicity. G_p is the first-order feedback transfer function obtained by taking the Laplace Transform of the tank's mass balance

$$G_p = \frac{1}{\tau_p s} \tag{4.1.2}$$

G_c is found by taking the Laplace Transform of the PID controller equation

$$\begin{aligned}
C(t) &= C_0 + K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de}{dt} \\
C'(t) &= K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de}{dt} \\
C'(s) &= K_p E(s) + \frac{K_i}{s} E(s) + K_d \left[sE(s) - \cancel{e(0)}\right] \\
G_c = \frac{C'(s)}{E(s)} &= K_p + \frac{K_i}{s} + K_d s
\end{aligned} \tag{4.1.3}$$

The setpoint tracking transfer function thus becomes

$$\begin{aligned}
\frac{H'(s)}{H'_{sp}(s)} &= \frac{\left(\frac{1}{\tau_p s}\right)\left(K_p + \frac{K_i}{s} + K_d s\right)}{1 + \left(\frac{1}{\tau_p s}\right)\left(K_p + \frac{K_i}{s} + K_d s\right)} \\
&= \frac{K_d s + K_p + \frac{K_i}{s}}{(K_d + \tau_p)s + K_p + \frac{K_i}{s}} \\
&= \frac{K_d s^2 + K_p s + K_i}{(K_d + \tau_p)s^2 + K_p s + K_i} \\
&= \frac{K_1 s^2 + K_2 s + 1}{K_3 s^2 + K_2 s + 1}
\end{aligned} \tag{4.1.4}$$

where $K_1 = K_d/K_i$; $K_2 = K_p/K_i$; and $K_3 = (K_d + \tau_p)/K_i$. The numerator and denominator have zeros and poles, respectively. The zeros are

$$s = \frac{-\xi_n \pm \sqrt{\xi_n^2 - 1}}{\tau_n}; \quad \text{where} \quad \tau_n = \sqrt{K_1} \text{ and } \xi_n = \frac{K_2}{2\sqrt{K_1}} \tag{4.1.5}$$

where $s \approx -9.796$ and $s \approx -0.204$. The poles are

$$s = \frac{-\xi_d \pm i\sqrt{1 - \xi_d^2}}{\tau_d}; \quad \text{where} \quad \tau_d = \sqrt{K_3} \text{ and } \xi_d = \frac{K_4}{2\sqrt{K_3}} \tag{4.1.6}$$

where $s \approx -0.204 \pm 0.2i$. This produces the Root Locus diagram shown in figure 4.3.

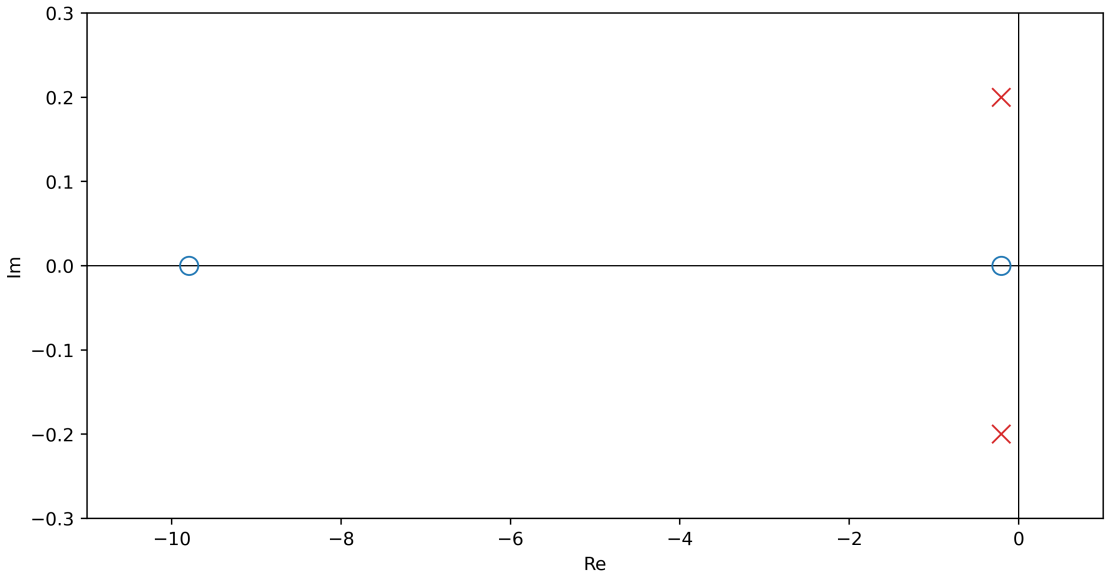


Figure 4.3: Root Locus diagram for the base simulation.

We can see that there is a small imaginary portion, which we would expect given the underdamped response in figure 4.1. There are no zeros or poles in the right-hand plane, so we would expect the system to be stable. Additionally, since all poles lie on the left-hand plane, the system is slightly overdamped. Now that we have shown that our transfer function potentially describes the simulation, we can begin solving for the functional solution. Since it is just a tank with no other dynamics, we just need to solve the first-order transfer function

$$\frac{H'(s)}{H'_{sp}(s)} = \frac{1}{s+P} \tag{4.1.7}$$

where P is a pole of equation (4.1.4). Performing the inverse Laplace Transform, we get the functional solution

$$h(t) = \Delta h_{sp}\left(1 - e^{-Pt}\right) + h_0 \tag{4.1.8}$$

Remembering from equation (4.1.6) that $P = -s = 0.204 \pm 0.2i$, our functional solution must include a sine and cosine term for the imaginary part

$$h(t) = \Delta h_{sp}\left[1 - e^{-0.204t}\left(\cos(0.2t) + \sin(0.2t)\right)\right] + h_0 \tag{4.1.9}$$

Plotting this equation along with the simulation yields
We can see from this plot that our model looks very similar to the simulation, but they

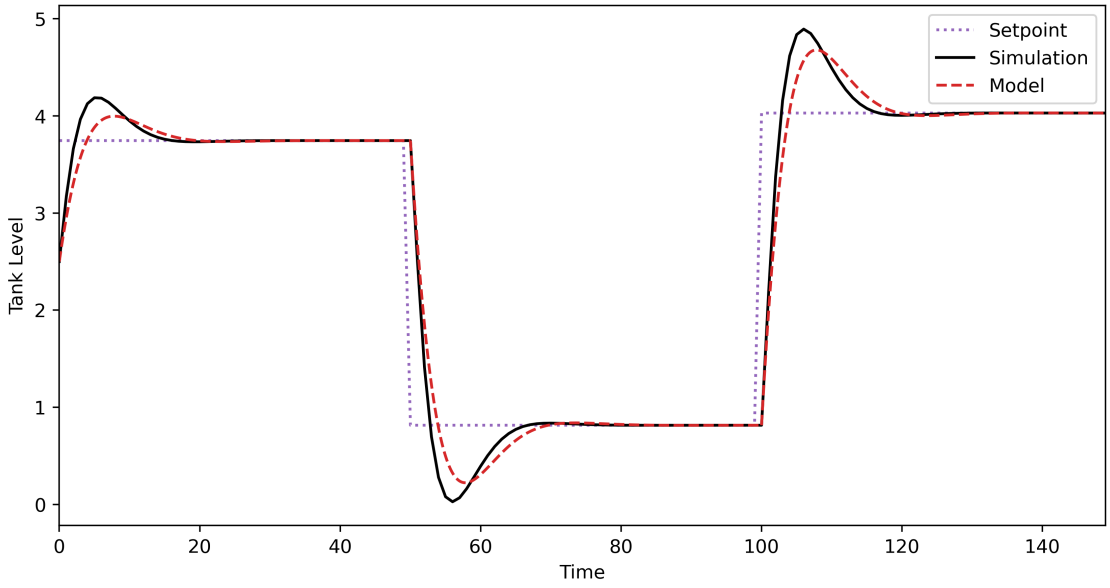


Figure 4.4: Simulation and functional solution responses to setpoint changes.

do not quite match up perfectly. This is due to the sampling time of the simulation – the functional solution assumes that the input data is a continuous stream, but the simulation has discretized data. In order to make the simulation match the model more closely, we need to modify the code such that we can modify how often the controller reads data and applies control. Smaller sampling times should cause the simulation to look more like the functional solution.

4.2 Implementing Data Sampling Rates

As we saw in the first simulation, the simulation response and the functional solution did not match up entirely. One possible reason is because of the data sampling rate; one way to think about this is numerically solving ODEs. If the step size is too big, then the solution will be inaccurate and "sluggish," trailing behind the actual solution; if the step size is too small, then the solution will require accuracy that floating point values cannot achieve, propagating rounding errors and introducing instability. If something like this is happening, then being able to change the sampling rate may help. We begin the code just as before, but this time importing the `math` library

```
# Import libraries
import numpy                as np
import matplotlib.pyplot as plt
import math

# Tank dimensions and simulation time
Diameter = 3
Height   = 5
Ac       = np.pi * Diameter**2 / 4
Volume   = Ac * Height
MinLevel = 0
Time     = 150
```

We can also initialize the `Feedback controller parameters` and `Realism properties`, however this time adding a `SamplingTime` variable and `DataPoints` and `PlottingTime` array each with length `Time/SamplingTime`. This will allow us to set a specific `SamplingTime` that appropriately scales the lengths of all lists and arrays.

```
# Feedback controller parameters
C0         = 0
Ti         = 5
Td         = 0.1
Kp         = 3
Ki         = Kp / Ti
Kd         = Kp * Td
CtrlParams = (C0, Kp, Ki, Kd)

# Realism properties
SamplingTime     = 0.1
DataPoints       = math.ceil(Time / SamplingTime)
PlottingTime     = np.arange(0, Time, SamplingTime)
```

The setpoint `for` loop remains the same. We do need to modify the `TankLevel` function to account for the new sampling rates.

```
def TankLevel(L0, F_in, F_out):
    L = SamplingTime * (F_in + F_out) / Ac + L0
    return L
```

Lastly, we need to change some things in the `PID` class, particularly the derivative and integral terms. Currently, we have a $\Delta t = 1$ second, allowing us to effectively ignore it in calculations. With changing sampling rates, we need the derivative to be divided by the `SamplingTime`, and the integral needs a $dt =$ `SamplingTime`.

```
# PID controller class
class PID:
    def __init__(self, C0, Kp, Ki, Kd, Data, Setpoint, \
                 WindupMitigation, DeadBand, PumpSpeed):

        # Error list
        error = []

        # Derivative control
        for i in range(len(Setpoint)):
            error.append(Setpoint[i] - Data[i])
        if i >= 1:
            derivative = (error[-1] - error[-2]) / SamplingTime
        else:
            derivative = 0

        integral = np.trapz(error, dx = SamplingTime)

        # PID controller function
        PIDc = C0 + Kp * error[-1] + Ki * integral + Kd * \
               derivative

        # Controller saturation
        self.action = PIDc
```

With a new `SamplingTime` of 0.1 seconds, the simulation is now much closer to the functional solution, seen in figure 4.5. We can see in figure 4.6 that a larger `SamplingTime` results in inaccurate data, and a smaller `SamplingTime` results in more accurate data.

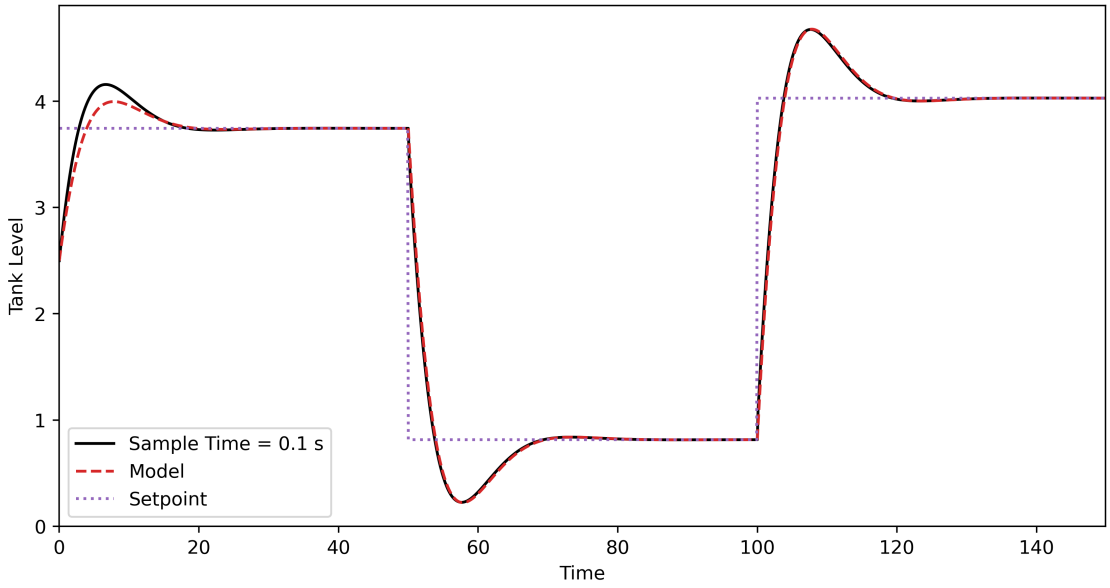


Figure 4.5: Simulation with a smaller sampling time.

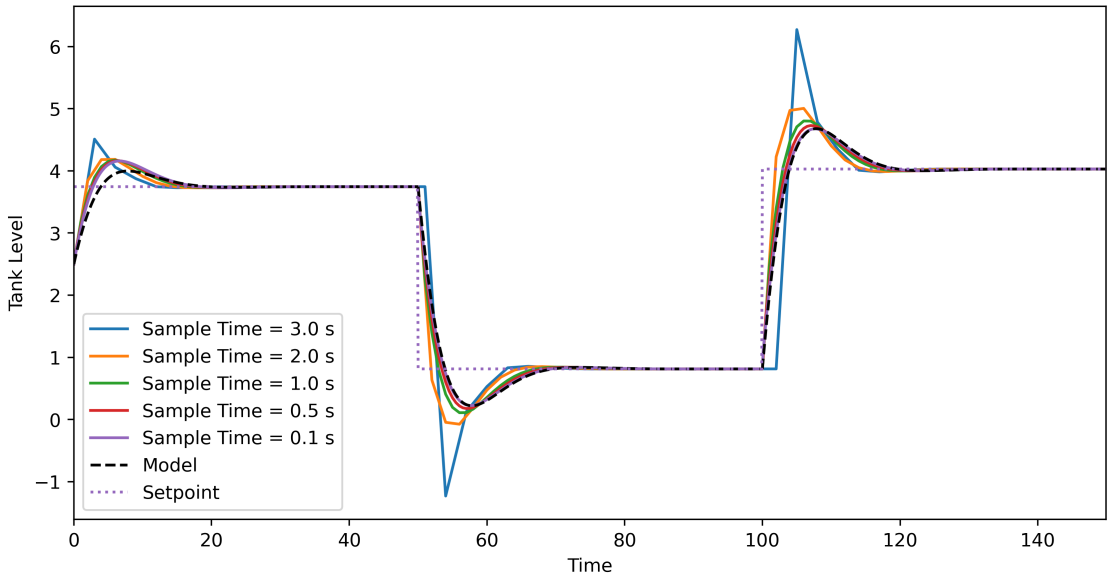


Figure 4.6: The simulation increases in accuracy with smaller sampling times.

This tells that that our simulation is indeed relatively accurate to what we would expect from theory. We can now move on to simulating more complicated systems with confidence

that our results will make sense. For the remainder of simulations, we will use a sample time of 0.1 seconds.

4.2.1 Verifying Simulation Stability

One final test to verify our simulation is to see how well it follows stability criteria, namely seeing if it has the expected behavior for over-, critically-, and under-damped systems. Remember that a PID controlled system will oscillate depending on the denominator of the transfer function; take our transfer function in eq. (4.1.4)

$$\frac{H'(s)}{H'_{sp}(s)} = \frac{K_1 s^2 + K_2 s^2 + 1}{K_3 s^2 + K_2 s + 1}$$

The denominator should take the form

$$\tau^2 s^2 + 2\xi\tau s + 1$$

which means that $\tau = \sqrt{K_3}$ and $\xi = K_2/(2\sqrt{K_3})$, and the poles of the denominator are found by

$$s = \frac{-\xi \pm \sqrt{\xi^2 - 1}}{\tau}$$

if they are real, and by

$$s = \frac{-\xi \pm i\sqrt{1 - \xi^2}}{\tau}$$

if they are complex. An oscillatory system requires that $\xi < 1$. There are infinitely-many combinations for K_p, K_i, and K_d that will theoretically make the system oscillatory, and the table below shows a number of them.

Table 4.1: P, I, and D gains that are expected to give specific damping behavior.

Damping	K_p	K_i	K_d
Over-damped	6	0.6	0.3
Critically-damped	3	0.2788594	1
Under-damped	5	1	1

These values yield the Root Locus diagrams in figure 4.7 and the responses in figure **??**

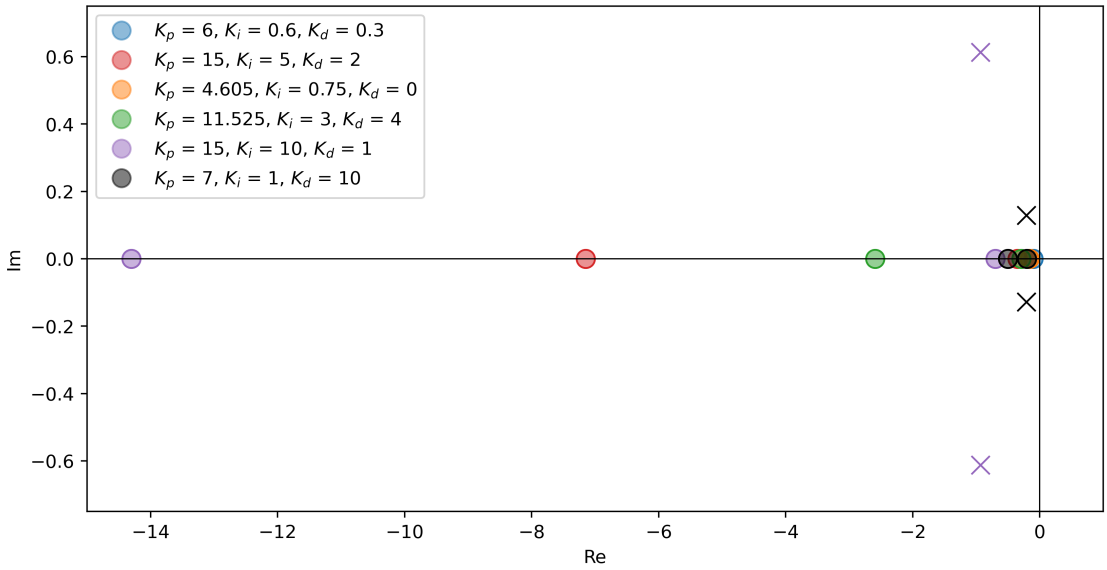


Figure 4.7: Root Locus diagrams for all values in table 4.1.