EEET 3046 Control Systems (2020)

Lecture 6: PID control

Dr. Yee Wei Law ⟨yeewei.law@unisa.edu.au⟩

Contents

Figure 1: These companies have one thing in common: they are among the world's five dominant vendors of PID control hardware [LAC06].

1 Introduction

In the previous lecture, we were introduced to the the *proportional-integral-derivative controller* — *PID controller* in short — a controller with two zeros for improving stability and transient response, and one pole at the origin for eliminating steady-state error. These controllers — also known as *three-mode controllers* — are of fundamental importance. This importance is reflected in a 2002 study done by Honeywell [DM02] which reported:

> Based on a survey of over eleven thousand controllers in the refining, chemicals and pulp and paper industries, 97% of regulatory controllers utilize a PID feedback control algorithm.

PID controllers are intuitive, easy to tune, and can even be used without knowledge of the plant model (although the plant should be SISO LTI). In fact, they (especially the PI variant) are by far the most widely adopted controllers in industry due to their advantageous cost/benefit ratio [VV12]. When not used by itself, a PID controller is often a part of a more sophisticated control system. For example, in Lecture 1, we saw how a PID controller can be used in the speed control loop of a direct torque controller for variable-speed drives.

Detail: History

The PID controller is also one of the earliest types of controllers to appear:

- In 1911, the first PID controller was used in Elmer Sperry's ship autopilot.
- In 1922, Nicholas Minorsky presented the first rigorous analysis of PID control.
- In the 1930s, general-purpose PID controllers became commercially available. The first theoretical papers on process control were published in the same period.
- In 1942, Ziegler and Nichols proposed their tuning rules, further propelling the development of PID control.
- By the 1980s, digital PID controllers have become widespread. Since the 1980s, digital hardware has been used on a routine basis and has had a tremendous impact on process control. Figure 1 shows the major industry players.

The fact that advances in PID control are tightly linked to developments in the process control industry explains why this lecture has a heavier process control flavor than the other lectures.

In this lecture, we shall look at PID control more closely in terms of

- **The properties of proportional action, integral action, and derivative action**: Knowing the properties/roles/effects of each of these actions is very useful in practice because it tells us how to troubleshoot and fine-tune a PID controller.
- **The forms/structures of PID controllers**: Different forms/structures have different pros and cons, and different commercial controllers support different forms/structures. Knowledge in this area helps us choose between products and use them efficiently.
- **Implementation of a PID controller**: This includes software and hardware implementations. The discussion on hardware implementation introduces lag, lead and lag-lead compensators.
- **Limitations of PID control**: PID controllers are simple, second-order (at most) controllers for SISO LTI systems, and naturally have limitations. Knowing this limitations helps us understand when we should not expect PID control to work well, if at all; and appreciate the need for more advanced control.

2 Proportional action

The proportional control action $u(t)$ is proportional to the current control error $e(t)$, i.e.,

$$u(t) = K_p e(t), \tag{1}$$

where K_p is the proportional gain. The control error is defined as $e(t) \overset{\text{def}}{=} r(t) - y(t)$, where $r(t)$ is the reference input or set-point, and $y(t)$ is the system output. In the process control literature, $r(t)$ and $y(t)$ are often denoted by *SP* ("Set-Point") and *PV* ("Process Variable") respectively. The transfer function of the proportional action is simply

$$\frac{U(s)}{E(s)} = K_p. \tag{2}$$

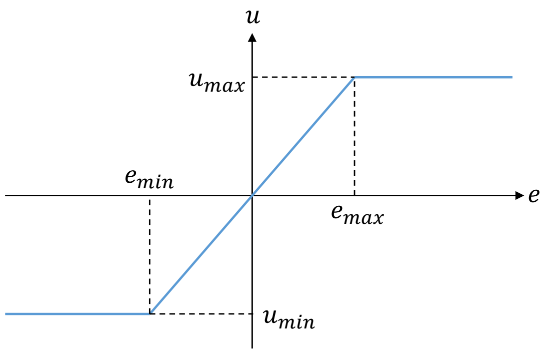


Figure 2: The definition of the proportional band originates in the observation that proportionality is restricted to a finite range of the control input.

In the process control industry, the proportional gain is frequently expressed as a percent *proportional band* (PB). In fact, some commercial controllers have a PB rather than a proportional gain setting [SMEDI10, p. 137]. The definition of the PB originates in the observation that due to actuator saturation, the control output (u) is only proportional to the control input (e) for a finite range of the control input (see Figure 2).

Definition: Proportional band

... is the minimum change required in the control input to drive a full-range change in the control output, expressed as a percentage of the control output range [Lip06, p. 116]. Using the symbols in Figure 2,

$$\text{PB} \overset{\text{def}}{=} \frac{e_{max} - e_{min}}{u_{max} - u_{min}} \times 100\% = \frac{100\%}{\frac{u_{max}-u_{min}}{e_{max}-e_{min}}} = \frac{100\%}{K_p}. \tag{3}$$

The term “wide band” is associated with a high PB value (low K_p). Likewise, the term “narrow band” is associated with a low PB value (high K_p).

Quiz 1

Is a P controller with a PB of 100% more or less sensitive than a P controller with a PB of 1000%?

2.1 Effects of proportional action

Consider the first-order plant $G(s) = b/(s + a)$, where $a, b \in \mathbb{R}^+$, and a proportional controller $C(s) = K$ in series with $G(s)$ in the unity feedback configuration. The closed-loop transfer function is

$$\frac{C(s)G(s)}{1 + C(s)G(s)} = \frac{Kb}{s + a + Kb}.$$

The rise time, settling time and step-response steady-state error are

$$t_r \approx \frac{2.2}{a + Kb}, \quad t_s \approx \frac{4}{a + Kb}, \quad e_{\text{step}}(\infty) = \frac{1}{1 + \lim_{s\to 0} C(s)G(s)} = \frac{a}{a + Kb} \quad \text{respectively.}$$

Hence, by increasing K, we can reduce the rise time, settling time and steady-state error, but the steady-state error is nonzero.

To generalize our understanding of the effects of proportional action to second- and higher-order systems, we shall make some empirical observations of these effects on the test systems in Table 1. The systems have been chosen for the variety of their root loci, as shown in Figure 3.

Table 1: Test systems.

Test system	Description	Plant transfer function
1	2nd-order system with two real poles and no zero	$G_1(s) = \dfrac{1}{(s+2)(s+3)}$
2	3rd-order system with three real poles and no zero	$G_2(s) = \dfrac{1}{(s+2)(s+3)(s+4)}$
3	2nd-order system with two complex poles and one zero	$G_3(s) = \dfrac{s+2}{(s^2+2s+2)}$
4	3rd-order system with two complex poles, one integrator pole and one zero	$G_4(s) = \dfrac{s+2}{s(s^2+2s+2)}$

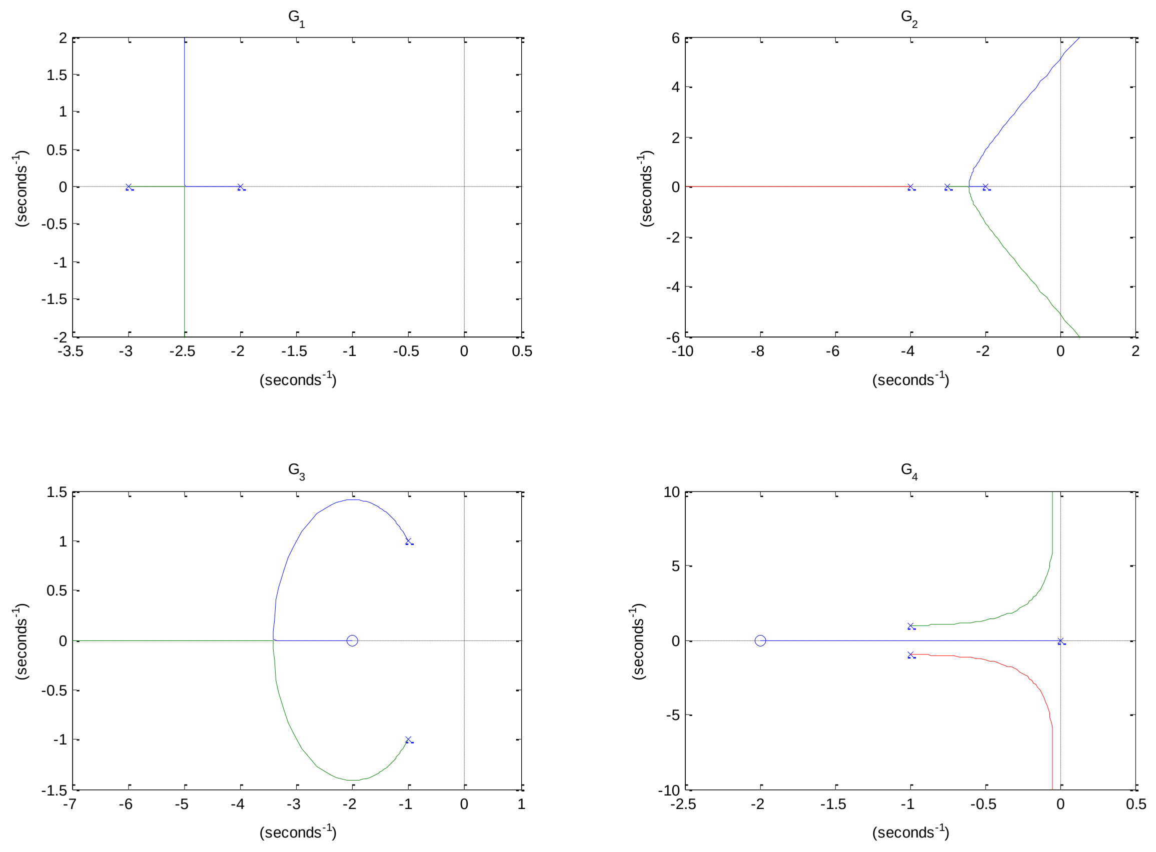


Figure 3: Root loci of the test systems in Table 1.

By connecting a P controller to each of the test systems and varying the gain, we can get the plots in Figure 4. From the plots, we can see the general effects of increasing the proportional gain (see also Table 2):

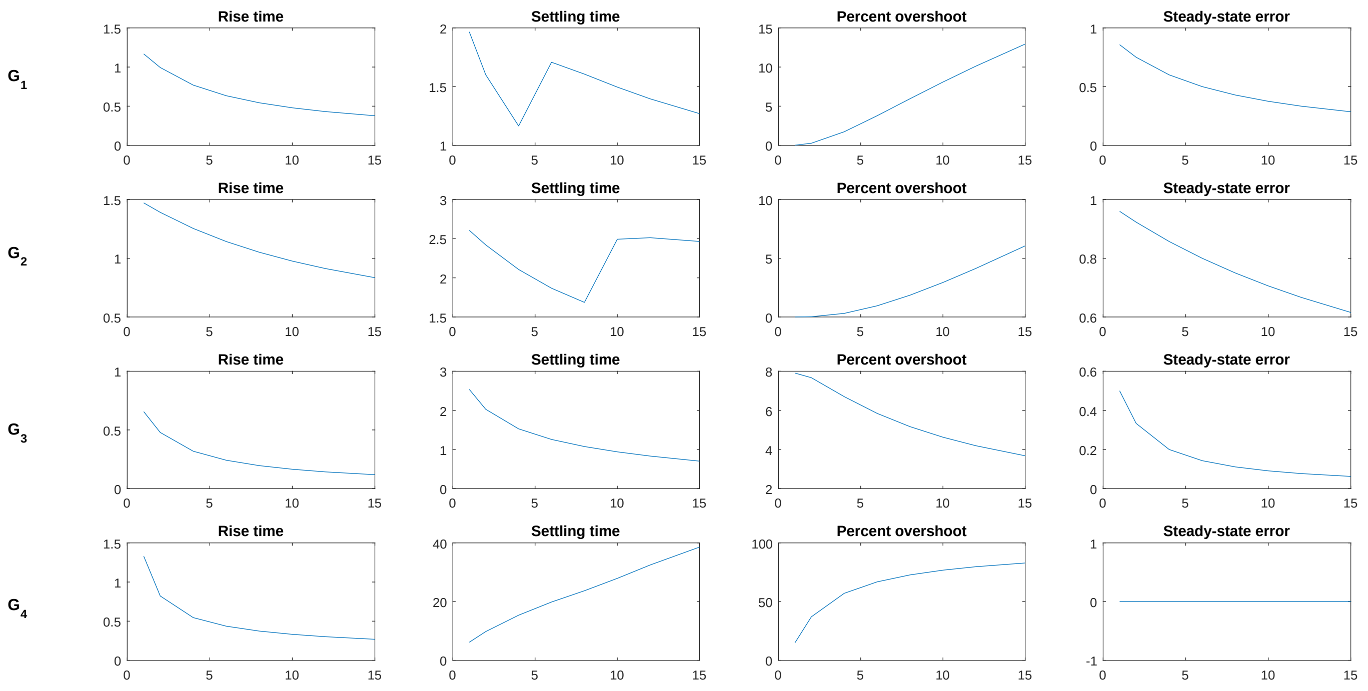


Figure 4: Plots of performance characteristics versus proportional gain for the test systems in Table 1.

- The rise time decreases.
- The settling time may increase or decrease, depending on the system.
- The overshoot may increase or decrease, depending on the system.
- The steady-state error decreases.

2.2 Manual reset

As seen in the preceding subsection, proportional action has the main drawback where by itself, it leaves a residual steady-state error, commonly called an *offset* in short. Even if the plant is a type 1 system, a nonzero offset will occur as a result of a step *disturbance*. This steady-state error can be reduced by increasing the gain, but this may destabilize the system (imagine the root locus branching into the right half plane). The exceptions are very slow processes, such as float-type valves, thermostats and humidostats [Lip06, p. 116].

The residual error motivates the addition of a *bias* term, $\bar{u}$, that is manually adjusted to eliminate the steady-state error:

$$u(t) = K_p e(t) + \bar{u}. \tag{4}$$

The process of adjusting $\bar{u}$ is called *manual reset*. Since the bias term is independent of the controller gain, it is an open-loop input to the plant.

Example 1

Consider the unity feedback system with plant $G(s)$ and controller $C(s)$:

$$G(s) = \frac{1}{(s+2)(s+3)}, \qquad C(s) = 60.$$

Calculate the steady-state error, $e_{r=10}(\infty)$, corresponding to a step input of 10. Furthermore,

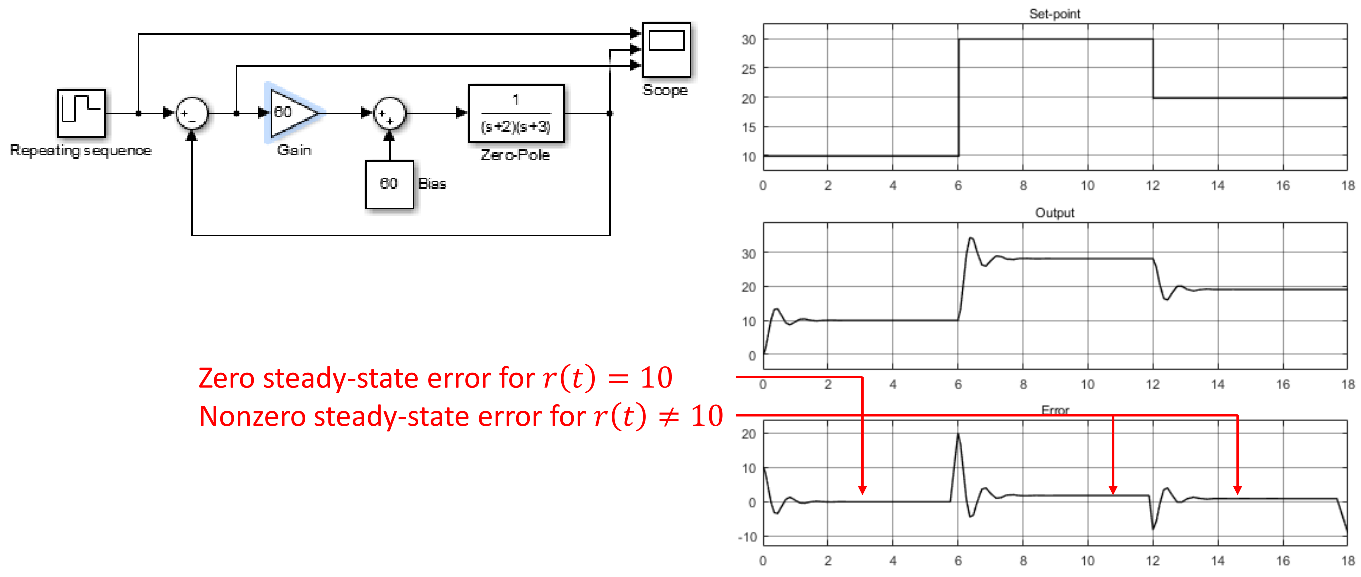


Figure 5: Block diagram and time responses of a P-controlled system with manual reset targeted at a set-point of 10.

determine the bias term $\bar{u}$ that reduces $e_{r=10}(\infty)$ to zero.

Solution: Prior to manual reset,

$$E(s) = \frac{1}{1 + C(s)G(s)} R(s) \qquad \text{(see Lecture 3).}$$

For $r(t) = 10$,

$$e_{r=10}(\infty) = \lim_{s \to 0} sE(s) = \lim_{s \to 0} s \frac{1}{1 + C(s)G(s)} \frac{10}{s} = \frac{10}{1 + \lim_{s \to 0} C(s)G(s)} = \frac{10}{1 + \frac{60}{2 \cdot 3}} = 0.9091,$$

which is small relative to the input but nonzero. To reduce $e_{\text{step}}(\infty)$ to zero, we must have $y_{ss}(\infty) = 10$, based on the Simulink model which doubles as the block diagram in Figure 5, we have

$$y_{ss}(\infty) = \lim_{s \to 0} sY(s) = \lim_{s \to 0} sG(s) \left(C(s)E(s) + \frac{\bar{u}}{s} \right) = \bar{u} \lim_{s \to 0} G(s) = \frac{\bar{u}}{2 \cdot 3} = 10 \implies \bar{u} = 60.$$

Figure 5 shows that manual reset can eliminate the steady-state error only for a fixed input. This is why we need integral control, to be discussed in the next section.

Nevertheless, for control applications where offsets can be tolerated, proportional-only control is adequate. For example, in some level control problems, maintaining the liquid level close to the set point is not as important as merely ensuring that the storage tank does not overflow or run dry.

3 Integral action

The integral control action $u(t)$ is proportional to the integral of the control error, i.e.,

$$u(t) = K_i \int_0^t e(\tau)\,\mathrm{d}\tau, \tag{5}$$

where K_i is the integral gain. The transfer function of the integral action is

$$\frac{U(s)}{E(s)} = \frac{K_i}{s}. \tag{6}$$

Recall from Lecture 3 that a type 1 system has a pole at the origin, which allows it to track step inputs with zero steady-state error, even in the presence of step disturbances. Therefore, integral action plays the role of the bias term in proportional action in eliminating steady-state error, without needing manual adjustment; this is why integral action is often called *automatic reset*.

Integral action is rarely used by itself, because the control action will remain small before the control error accumulates to a sizeable level. In contrast, proportional action is immediate, so integral action is normally used in conjunction with proportional action in the form of the proportional-integral (PI) controller, the most used controller in the industry. The PI controller is often written in the form:

$$\frac{U(s)}{E(s)} = K_p\left(1 + \frac{1}{T_i s}\right), \tag{7}$$

where T_i is called the *integral time* or *reset time*. In process control terminology, T_i is also called *seconds per repeat* if the time unit is seconds, or *minutes per repeat* if the time unit is minutes. The inverse of seconds/minutes per repeat, T_i^{-1}, is called *repeats per second/minute*.

Detail: Seconds/minutes per repeat

To understand the rationale behind the terminology, consider a step change in $e(t)$, i.e., $e(t) = h$ for some constant h. At any time t, the proportional action is $K_p h$, which is a constant value, while the integral action is $\frac{K_p}{T_i}\int_{\tau=0}^{t} h\,\mathrm{d}\tau = \frac{K_p}{T_i} ht$, which is a linear function of t. At time instants $t = T_i, 2T_i, \ldots, nT_i$, the integral action becomes

$$K_p h, 2K_p h, \ldots, nK_p h,$$

which are respectively 1 time the proportional action, 2 times the proportional action, …, n times the proportional action. In other words, T_i measures the time taken for the integral action to match the proportional action, and this matching repeats every T_i seconds/minutes [SMEDI10, p. 138].

Quiz 2

A flow controller is known to have a proportional band of 500% and 0.05 seconds per repeat. What are the proportional gain and integral gain of this controller?

Example 2

Consider again the first-order plant $G(s) = b/(s+a)$, where $a, b \in \mathbb{R}^+$, and two types of controllers:

1. Integral controller $C(s) = K/s$: The closed-loop transfer function is
$$\frac{C(s)G(s)}{1+C(s)G(s)} = \frac{Kb}{s^2+as+Kb}.$$
When the closed-loop system is underdamped, the settling time and step-response steady-state error are
$$t_s = \frac{8}{a} \quad \text{and} \quad e_{\text{step}}(\infty) = 0 \quad \text{respectively.}$$
Compared to the the proportional-only controller, the integral controller is able to eliminate the steady-state error but at the expense of worse settling time (compared to $4/a$ of the open-loop system). Moreover, this settling time cannot be reduced by increasing the gain. It is exactly for this reason that we combine "proportional action" and "integral action" in the form of a PI controller.

2. Proportional-integral (PI) controller $C(s) = K(s+z)/s$: The closed-loop transfer function is
$$\frac{C(s)G(s)}{1+C(s)G(s)} = \frac{Kb(s+z)}{s^2+(a+Kb)s+Kbz}.$$
When the closed-loop system is underdamped, the settling time and step-response steady-state error are
$$t_s = \frac{8}{a+Kb} \quad \text{and} \quad e_{\text{step}}(\infty) = 0 \quad \text{respectively.}$$
This is a clear improvement over the proportional controller and the integral controller, because it is able to eliminate the steady-state error and yet allows the settling time to be reduced by increasing the gain.

3.1 Effects of integral action

As in Sect. 3.1, let us make some empirical observations of the effects of integral action on the test systems in Table 1. By connecting a PI controller (rather than a pure I controller) to each of the test systems and varying the integral gain, we can get the plots in Figure 11. From the plots, we can see the general effects of increasing the integral gain (see also Table 2):

- The rise time decreases.
- The settling time may increase or decrease, depending on the system.
- The overshoot increases.
- The steady-state error disappears.

3.2 Integrator windup

Besides being less responsive than proportional action, integral action has one more drawback: *reset windup*, also known as *integrator windup*. In the practical on voltage-based DC motor position

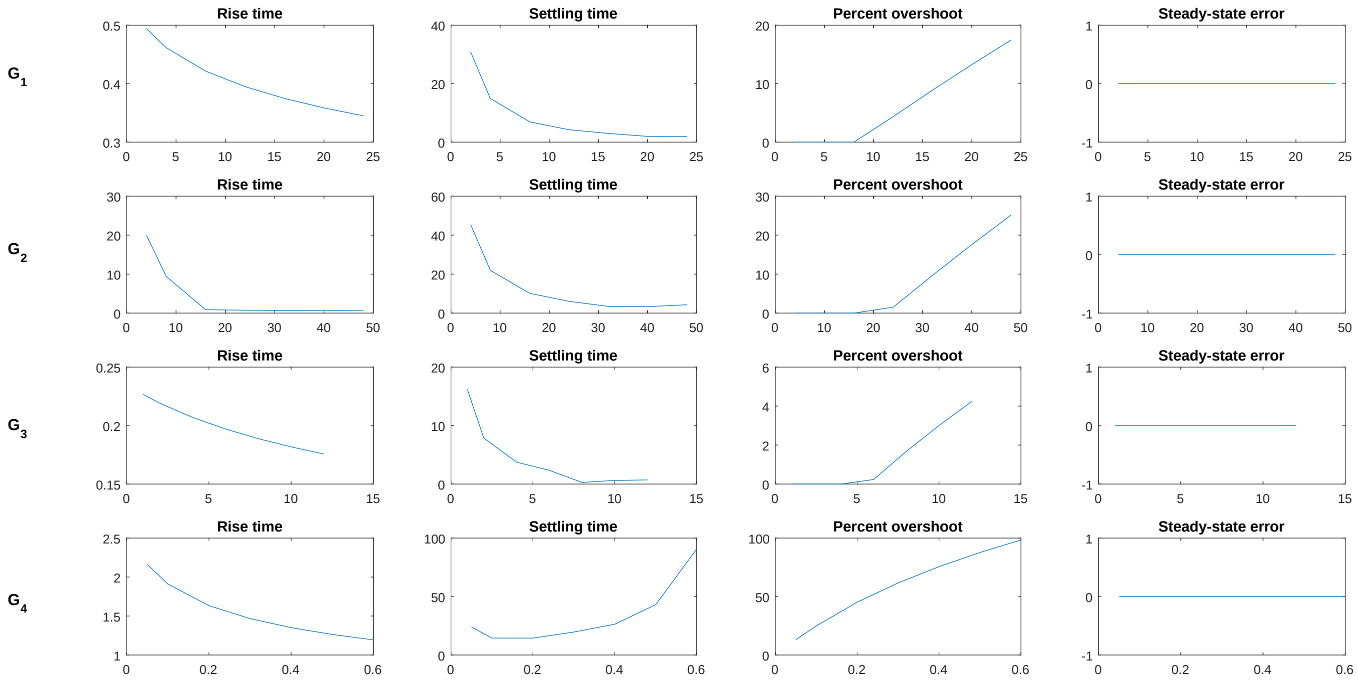


Figure 6: Plots of performance characteristics versus integral gain for the test systems in Table 1.

control, we have witnessed the *actuator saturation* phenomenon, where the input to the motor has to be limited to a certain voltage range. Actuator saturation then leads to integrator windup if the controller has an integral mode. Here, let us look at integrator windup in more detail.

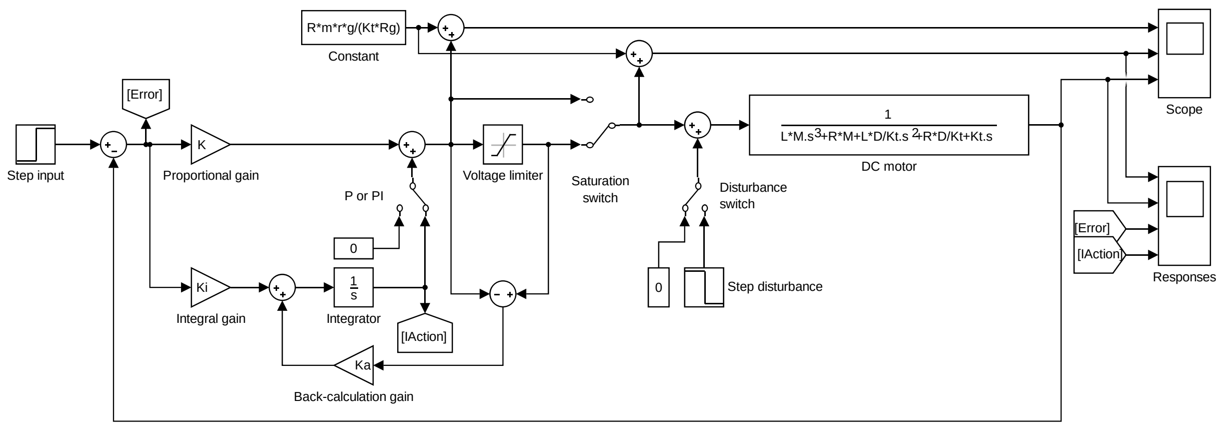


Figure 7: Simulink model of a DC motor position control system.

In the Simulink model in Figure 7, if we (i) enable the PI controller, (ii) disable the voltage limiter, and (iii) set the back-calculation gain (to be defined later) to zero, we will observe the responses in Figure 8. In this case, which is free of actuator saturation, the motor reaches the desired angle rapidly, and the integral error dies away quickly, but meeting the input voltage demand of a few hundred volts is impractical.

Now, if we enable the voltage limiter, we get the responses in Figure 9, which shows that actuator saturation causes the control error $e(t)$ to decrease linearly rather than exponentially. This causes the integral error and hence integral action to increase negative-quadratically (because the integral of a linear function is a quadratic function). Even after reaching the desired angle, the motor keeps

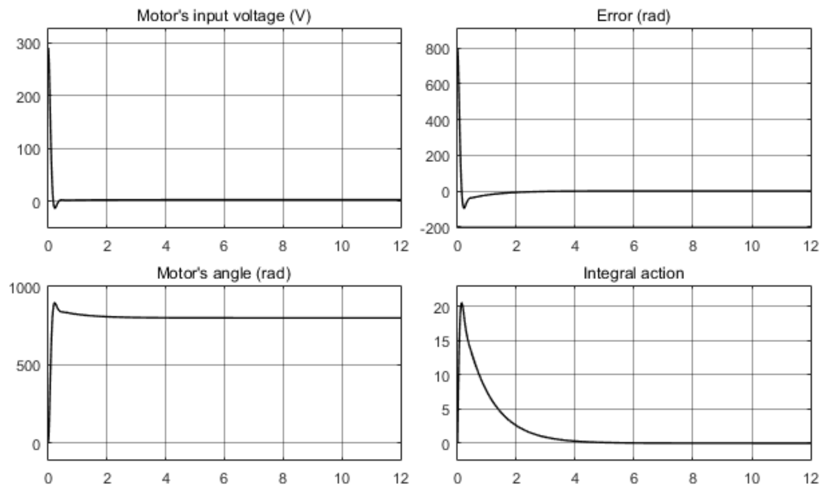


Figure 8: Responses of the system in Figure 7 to a step input of 800, in the absence of actuator saturation.

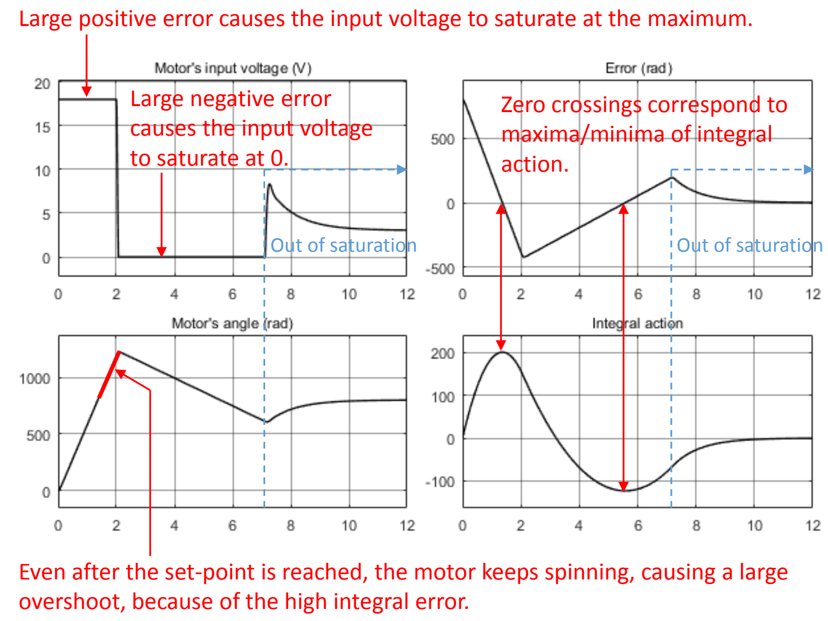


Figure 9: Responses of the system in Figure 7 to a step input of 800, in the presence of actuator saturation but no integrator anti-windup.

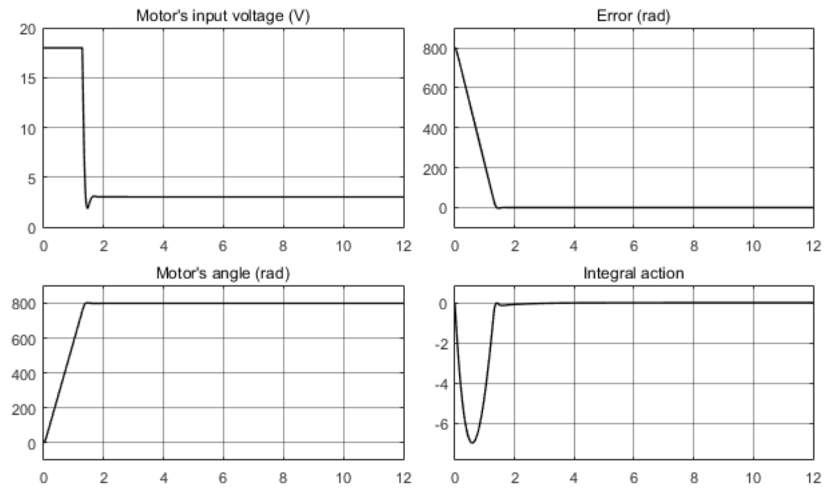


Figure 10: Responses of the system in Figure 7 to a step input of 800, in the presence of actuator saturation and integrator anti-windup.

spinning because the integral action remains large. Once $e(t)$ becomes negative, the integral action starts decreasing. At some point when the control action (proportional action + integral action) becomes small enough, the system comes out of saturation. It is only then can the magnitude of the integral error start decreasing exponentially to zero. This is the reset windup or integrator windup phenomenon that was alluded to at the beginning of this subsection.

The remedy is *anti-reset windup* or *integrator anti-windup*. We know that as a remedy, we need to suppress integration of $e(t)$ during actuator saturation. One of the most widely used anti-windup schemes is *back-calculation anti-windup*, which is exactly the anti-windup scheme implemented in Figure 7. Instead of disabling integration, back-calculation resets integration dynamically at a time constant called the *tracking time constant*, denoted T_t [ÅH06, Ch. 3]. The shorter T_t is, the faster the integral error is reset once saturation commences. The inverse of T_t, denoted K_a, is called the *tracking gain* or *back-calculation gain*.

Detail: Tracking time constant

From Figure 7, we can see the integral action is

$$U_i(s) = \frac{K_i E(s) + K_a[U_{sat}(s) - U(s)]}{s}, \tag{8}$$

where $U_{sat}(s)$ is the saturated control action, defined as

$$U_{sat}(s) = \begin{cases} \dfrac{u_{min}}{s} & \text{if } u(t) < u_{min}, \\ U(s) & \text{if } u_{min} \le u(t) \le u_{max}, \\ \dfrac{u_{max}}{s} & \text{if } u_{max} < u(t). \end{cases} \tag{9}$$

In the absence of saturation, the integral action is the normal integral action defined in Eq. (5). When $u(t)$ saturates at u_{min},

$$\begin{aligned} U_i(s) &= \frac{K_i}{s}E(s) - \frac{K_a}{s}[U_p(s) + U_i(s)] + \frac{K_a u_{min}}{s^2} \\ \implies \left(1 + \frac{K_a}{s}\right) U_i(s) &= \frac{K_i}{s}E(s) - \frac{K_a}{s}U_p(s) + \frac{K_a u_{min}}{s^2} \\ \implies U_i(s) &= \frac{K_i E(s) - K_a U_p(s)}{s + K_a} + \frac{K_a u_{min}}{s(s + K_a)} \\ \implies u_i(t) &= \mathcal{L}^{-1}\left\{\frac{K_i E(s) - K_a U_p(s)}{s + K_a}\right\} + u_{min}\left(1 - e^{-t/T_t}\right), \end{aligned}$$

where $U_p(s)$ and $U_i(s)$ represent the proportional and integral actions respectively. The second term of the equation above approaches u_{min} (a negative value) at a time constant of T_t, thereby validating our earlier statement about how T_t determines the response of the anti-windup.

Back to the system in Figure 7. Once we set the back-calculation gain to a suitable value, we can get the responses in Figure 10, which shows the integral action is two orders of magnitude smaller than the previous case without anti-windup. More importantly, overshoot has been flattened out. Other anti-windup schemes such as the integrator clamp and conditional integration schemes will be introduced in a later practical — these other schemes are not covered here because they are more ad hoc. As some of you might find out in EEET 4071 Advanced Control, state-space methods provide a more integrated approach to anti-windup.

4 Derivative action

The derivative control action $u(t)$ is proportional to the rate of change of the control error, i.e.,

$$u(t) = K_d \dot{e}(t), \tag{10}$$

where K_d is the derivative gain. To see why/how derivative action works, observe:

- Large positive rate of change indicates $y(t)$ is falling fast below $r(t)$, and the need for a strong positive control action.
- Large negative rate of change indicates $y(t)$ is growing fast above $r(t)$, and the need for a strong negative control action.

The transfer function of the derivative action is

$$\frac{U(s)}{E(s)} = K_d s. \tag{11}$$

Derivative action is often paired with proportional action because a proportional action based on the predicted error at time $t + T_d$ can be expressed as

$$K_p e(t + T_d) = K_p \left[e(t) + T_d \dot{e}(t) + \frac{T_d^2}{2!} \ddot{e}(t) + \cdots \right] \approx K_p[e(t) + T_d \dot{e}(t)], \tag{12}$$

by first-order Taylor series approximation [Vis06, p. 6]. The right-hand side of the equation above is the sum of a proportional term and a derivative term. The constant T_d is called the *derivative time*.

4.1 Effects of derivative action

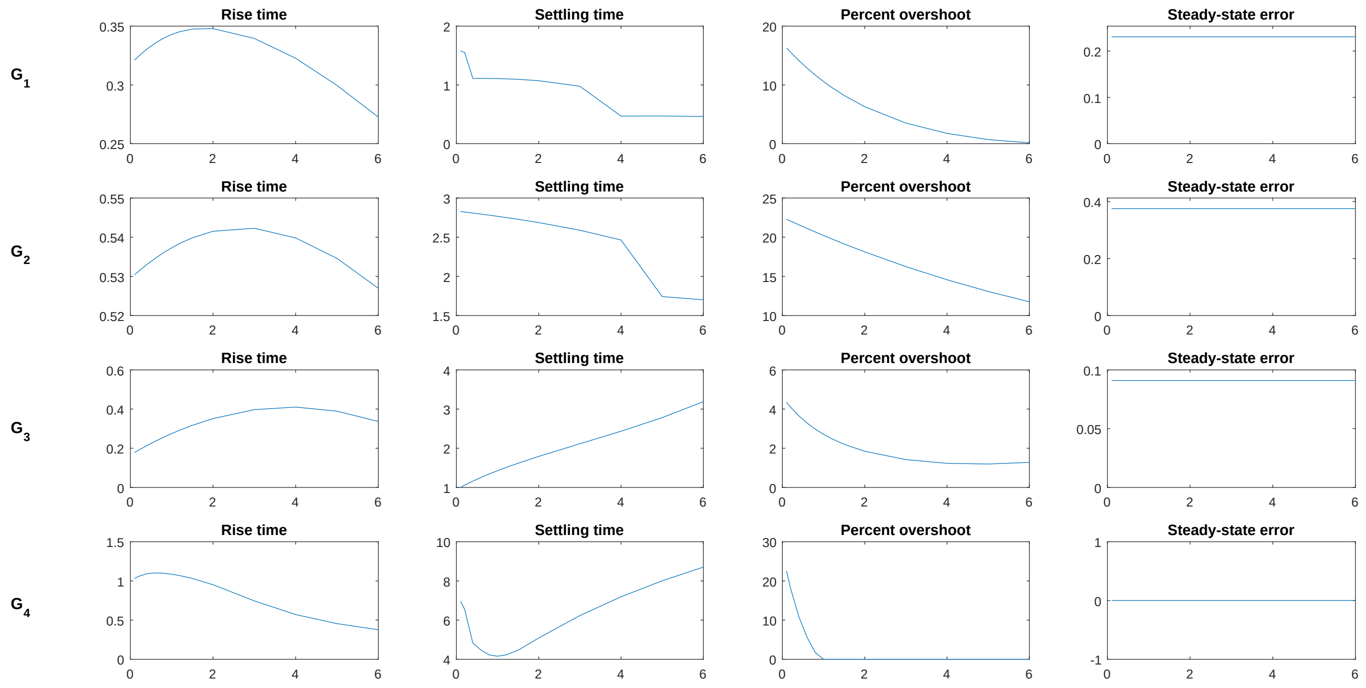


Figure 11: Plots of performance characteristics versus derivative gain for the test systems in Table 1.

As before, we shall make some empirical observations of the effects of derivative action on the test systems in Table 1, By connecting a PD controller (rather than a pure D controller) to each of the test systems and varying the derivative gain, we can get the plots in Figure 11. From the plots, we can see the general effects of increasing the derivative gain (see also Table 2):

- The rise time generally decreases.
- The settling time may increase or decrease, depending on the system.
- The overshoot decreases.
- The steady-state error does not change.

4.2 Noise and time delay

As mentioned in the previous lecture, differentiation amplifies noise, so derivative action by itself does not work well with noisy signals. For example, derivative action is seldom used for flow control, because flow control loops respond quickly and flow measurements tend to be noisy. For noise suppression, derivative action is typically paired with a *derivative (mode) filter* which is typically a first-order low-pass filter:

$$\frac{U(s)}{E(s)} = K_p \left(1 + \frac{T_d s}{\alpha T_d s + 1}\right), \qquad \alpha < 1. \tag{13}$$

Detail: Derivative filter

Alternative equivalent forms of the derivative filter $\frac{1}{\alpha T_d s + 1}$ include

- $\frac{1}{T_d s/N + 1}$, where $N > 1$ and is typically between 8 and 16 [ACL05];
- $\frac{1}{T_f s + 1}$, which is the form used by the MATLAB function `pid`.

For plants with time delay (also called transport lag, transport delay, or dead time, as mentioned in Lecture 2), the derivative gain needs to be carefully tuned, because certain values of the gain can destabilize these systems [ACL05]. This is why derivative control is not widely used in the process control industry, where time-delay processes are common; for example, whenever some fluid is transported through a pipe, there is a notable time delay. In fact, a survey [ACL05] shows that 80% of the PID controllers in use have their derivative action disabled.

5 PID actions in a nutshell

At this point, it is clear that a PID controller is essentially a controller that acts on past, present and future/predicted control errors.

Following our observations in Sect. 2.1, 3.1 and 4.1, Table 2 summarizes the effects of P/I/D actions. The saying "too much of a good thing" is applicable to the gain values, because despite the performance benefits,

- Increasing $K_p/K_i/K_d$ does not necessarily improve the settling time.
- Increasing K_p or K_i generally decreases the *stability margins* (to be defined rigorously in a later lecture).

Table 2: Summary of effects of P/I/D actions.

PID gain	Rise time	Settling time	% Overshoot	Steady-state error
Increasing K_p	*Decreases*	Depends	Depends	*Decreases*
Increasing K_i	Decreases	Depends	*Increases*	*Disappears*
Increasing K_d	Generally decreases	Depends	*Decreases*	Does not change

- Increasing K_d may increase or reduce the stability margins depending on the system, and the current PD gains [ACL05].

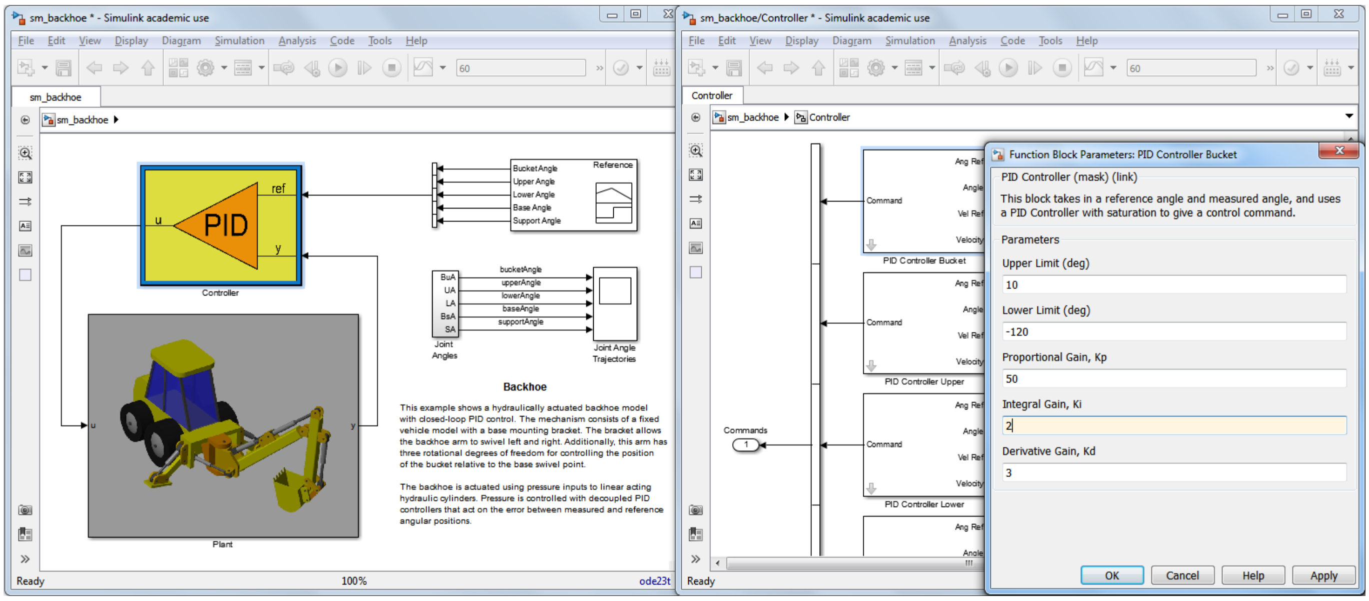
Naturally, determining the right gains (called tuning) is important and will be discussed in subsequent lectures.

Example 3

To visualize the impact of excessive K_p or K_i, run the MATLAB demo `sm_backhoe` by typing "`sm_backhoe`" on a MATLAB Command Window. Double click the "PID Controller Bucket" block (see Figure 12) to modify the PID gains.

(a) Increase K_p to 4000, and observe the angle response of the bucket. Figure 13 shows the bucket suffers a high-frequency ringing (i.e., oscillation) around 150° that renders the backhoe useless.

(b) Set K_p back to its default value, increase K_i to 2000, and observe the angle response of the bucket. Figure 14 also shows the bucket suffers excessive oscillations.



Backhoe

This example shows a hydraulically actuated backhoe model with closed-loop PID control. The mechanism consists of a fixed vehicle model with a base mounting bracket. The bracket allows the backhoe arm to swivel left and right. Additionally, this arm has three rotational degrees of freedom for controlling the position of the bucket relative to the base swivel point.

The backhoe is actuated using pressure inputs to linear acting hydraulic cylinders. Pressure is controlled with decoupled PID controllers that act on the error between measured and reference angular positions.

Function Block Parameters: PID Controller Bucket

PID Controller (mask) (link)

This block takes in a reference angle and measured angle, and uses a PID Controller with saturation to give a control command.

Parameters

Upper Limit (deg): 10

Lower Limit (deg): -120

Proportional Gain, Kp: 50

Integral Gain, Ki: 2

Derivative Gain, Kd: 3

Figure 12: The MATLAB demo called `sm_backhoe` uses PID control to control the movement of the bucket. Watch out for the animation of the backhoe.

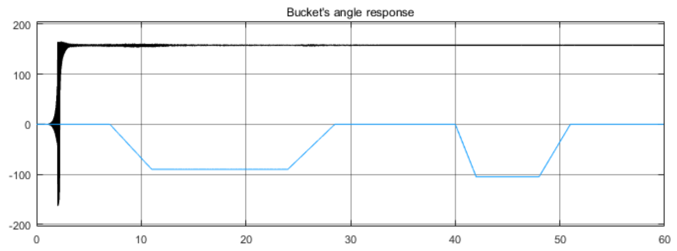


Figure 13: The angle response of the backhoe's bucket in Example 3, when $K_p = 4000$, $K_i = 2$, $K_d = 3$. Black line is the response; blue line is the set-point.

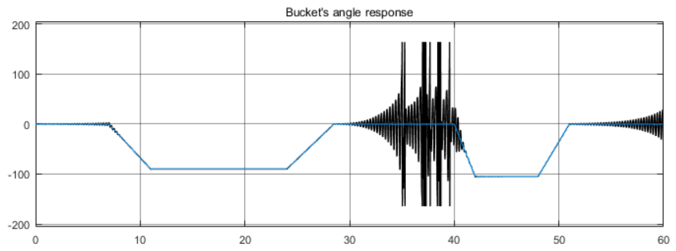


Figure 14: The angle response of the backhoe's bucket in Example 3, when $K_p = 50$, $K_i = 2000$, $K_d = 3$. Black line is the response; blue line is the set-point.

Quiz 3

State 'true' or 'false' for each of the statements below:

- Proportional action generally improves rise time and settling time.
- Integral action can reduce steady-state error to zero in spite of actuator saturation, as long as the actuator is able to get out of saturation.
- A PD controller can not only improve the transient response but also the stability of a delay-free LTI system.
- A PID controller without a derivative filter is physically unrealizable because of noise.

6 Forms/structures of PID controllers

A PID controller can combine proportional action, integral action and derivative action in various forms/structures. The terms "forms" and "structures" are used interchangeably in the literature (e.g., [ACL05, Vis06, O'D09]), and can be understood as (i) different ways of writing the basic PID control law; or (ii) variants/extensions of the basic PID control law. The next two subsections introduce some common PID forms using the nomenclature from [ÅH95] and [ÅH06], informally classified into "simple" and "advanced".

6.1 Simple forms

The simple forms are those that use only the control error as input.

- **Parallel form**: This is also called the *textbook form*. A PID control law in this form is written as

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \dot{e}(t), \tag{14}$$

  with the associated transfer function

$$C(s) = K_p + \frac{K_i}{s} + K_d s, \tag{15}$$

  which we have already seen. Note:

  - MATLAB provides the function `pid` that supports the creation of PID controllers in the parallel form coupled with a derivative filter:

$$C(s) = K_p + \frac{K_i}{s} + \frac{K_d s}{T_f s + 1}. \tag{16}$$

  - Some authors call this the "expanded form" [SMEDI10, p. 140], but this is unconventional.

- **Ideal form**: This is also known as the **noninteracting/noninteractive form**, the **ISA form**, or the **standard form** [ÅH06, Sect. 3.2]. Although the parallel form of PID controllers seems to offer the flexibility of tuning individual gains independently, most of the established tuning rules apply to the ideal form and series form. Paired with a derivative filter, a PID controller in the ideal form has the transfer function

$$C(s) = K_p \left(1 + \frac{1}{T_i s} + \frac{T_d s}{\alpha T_d s + 1}\right), \qquad \alpha < 1, \tag{17}$$

  where T_i is the integral time, and T_d is the derivative time. Note:
  - MATLAB provides the function `pidstd` for converting a PID transfer function of any other form to the standard form.
  - Some authors call this the "parallel form" [SMEDI10, p. 140], but this is unconventional.

- **Internal model control form**: This is obtained by applying the derivative filter to the whole transfer function rather than just the derivative term in the ideal form:

$$C(s) = K_p \left(1 + \frac{1}{T_i s} + T_d s\right)\left(\frac{1}{T_f s + 1}\right). \tag{18}$$

  This form is so-called because the internal model control tuning rule [SMEDI10, Sect. 12.2.2] generates controllers of this form, but this tuning rule is not covered in this course.

- **Series form**: This is also known as the **interacting/interactive form** or **classical form** [ÅH06, Sect. 3.2]. Paired with a derivative filter, a PID controller in the series form has the transfer function

$$C(s) = K_p \left(1 + \frac{1}{T_i s}\right)\left(\frac{T_d s + 1}{\alpha T_d s + 1}\right), \qquad \alpha < 1. \tag{19}$$

  The series form, being a PI controller and a PD controller (filtered or unfiltered) connected in series, was historically the first to be implemented. Implementations of the series form can be found in early analog controllers as well as contemporary digital control systems. However, the ideal form is more general/flexible than the series form because it supports complex conjugate zeros. Recall from the previous lecture how essential complex conjugate zeros are for stabilizing systems with oscillatory modes.

Quiz 4

Can the series form be converted to the ideal form? Can the ideal form be converted to the series form?

Traditionally, a process automation system is either a network of Programmable Logic Controllers (PLCs) or a Distributed Control System (DCS, see Figure 15) [Sie07]. For PLCs, Ladder Logic is the

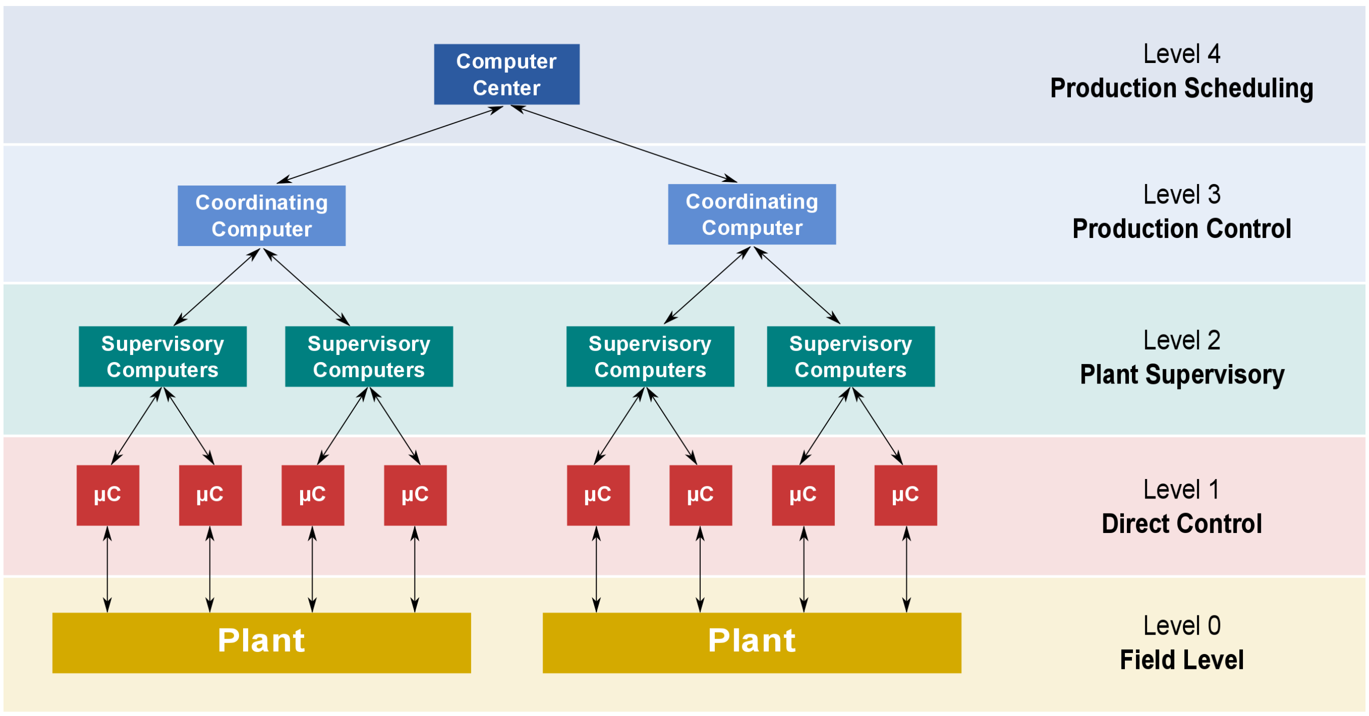


Figure 15: The Distributed Control System architecture. "Distributed" is a bit of a misnomer, and "hierarchical" is a more appropriate adjective. Image by Daniele Pugliesi - Own work, CC BY-SA 3.0, `https://commons.wikimedia.org/w/index.php?curid=31527335`.

standard configuration language, while for DCS, Function Block Diagram is the norm. In either language, PID control functionalities are typically available through dedicated Function Blocks. Some Function Blocks support both the ideal form and series form, but some support only the ideal form. For example, in Honeywell's Experion LX[1] Control Builder software, the PID Function Block supports only the ideal form [Hon14].

6.2 Advanced forms

Looking at the description of the PID block in the Control Builder manual (as in Figure 16), we can see that the block takes an argument called CTLEQN, which selects the equation/form of the PID controller [Hon14, pp. 380-382]:

- Equation A is the ideal form.
- Equations B and C are the advanced forms that we shall now discuss.

The advanced forms exist because of the following problems with the simple forms:

- "**Derivative kick**": This refers to a spike in the derivative error caused by a jump (i.e., step change) in the set-point. Mathematically, since

$$e(t) = r(t) - y(t) \implies \dot{e}(t) = \dot{r}(t) - \dot{y}(t),$$

a spike in $\dot{r}(t)$ creates a spike in $\dot{e}(t)$, which in turn creates a spike in the derivative action.

[1] `https://youtu.be/HoT7SEthuLA`

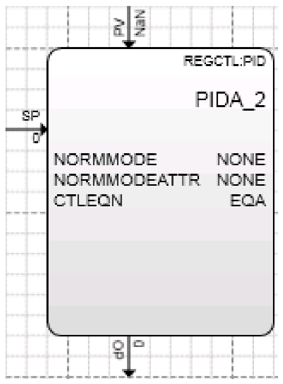


Figure 16: The PID block in Honeywell's Experion LX Control Builder library [Hon14, p. 347].

- "**Proportional kick**": This refers to a large change in the control error caused by a step change in the set-point. The large change in control error triggers a large change in the proportional action.

The "kicks" are undesirable because they inflict excessive wear and tear on the actuators; for example, valves deteriorate rapidly with time if exposed to high-pressure pulses repeatedly. The advanced forms are improvements to the basic PID forms because they can buffer the actuators from these "kicks". The advanced forms include

- The **Type B** or **PI-D** controller, which takes $-\dot{y}(t)$ instead of $\dot{e}(t)$ as input to the derivative term:

$$u(t) = K_p \left[e(t) + \frac{1}{T_i} \int_0^t e(\tau)\,\mathrm{d}\tau - T_d \dot{y}(t) \right]. \tag{20}$$

  This controller softens the derivative kick.

- The **Type C** or **I-PD** controller, which takes $-y(t)$ instead of $e(t)$ as input to the proportional term, and $-\dot{y}(t)$ instead of $\dot{e}(t)$ as input to the derivative term:

$$u(t) = K_p \left[-y(t) + \frac{1}{T_i} \int_0^t e(\tau)\,\mathrm{d}\tau - T_d \dot{y}(t) \right]. \tag{21}$$

  This controller softens both the derivative kick and proportional kick.

- The **beta-gamma** controller, which takes $e_p(t)$ instead of $e(t)$ as input to the proportional term, and $\dot{e}_d(t)$ instead of $\dot{e}(t)$ to the derivative term:

$$u(t) = K_p \left[e_p(t) + \frac{1}{T_i} \int_0^t e(\tau)\,\mathrm{d}\tau + T_d \dot{e}_d(t) \right], \tag{22}$$

  where

$$\begin{aligned} e_p(t) &\overset{\text{def}}{=} \beta r(t) - y(t), \quad 0 \le \beta \le 1; \\ e_d(t) &\overset{\text{def}}{=} \gamma r(t) - y(t), \quad 0 \le \gamma \le 1. \end{aligned}$$

  The tunable parameters β and γ enable *set-point weighting*, i.e., striking different trade-offs between responsiveness and suppression of derivative kicks and proportional kicks:

  – Setting $\beta = 1$ and $\gamma = 1$ reduces the beta-gamma controller to the ideal-form PID controller.

– Setting $\beta = 1$ and $\gamma = 0$ reduces the beta-gamma controller to the PI-D controller.

– Setting $\beta = 0$ and $\gamma = 0$ reduces the beta-gamma controller to the I-PD controller.

The beta-gamma controller is also called the *parallel PID controller with proportional and derivative mode weighting* [SMEDI10, Sect. 8.3.1]. In the Laplace domain, Eq. (22) is equivalent to

$$U(s) = K_p \left(\beta + \frac{1}{T_i s} + \gamma T_d s\right) R(s) - K_p \left(1 + \frac{1}{T_i s} + T_d s\right) Y(s).$$

Coupled with a derivative filter, the equation above becomes

$$U(s) = C_r(s)R(s) - C_y(s)Y(s), \tag{23}$$

where

$$C_r(s) = K_p \left(\beta + \frac{1}{T_i s} + \gamma \frac{T_d s}{\alpha T_d s + 1}\right), \tag{24}$$

$$C_y(s) = K_p \left(1 + \frac{1}{T_i s} + \frac{T_d s}{\alpha T_d s + 1}\right). \tag{25}$$

Eq. (23) is known as the *two degree-of-freedom PID* (2DOF PID) control law, an implementation of the beta-gamma controller. Figure 17 shows how $C_r(s)$ and $C_y(s)$ are connected in a block diagram. From the block diagram,

$$Y(s) = \underbrace{\frac{C_r(s)G(s)}{1 + C_y(s)G(s)}}_{\text{Servo control transfer function}} R(s) + \underbrace{\frac{G(s)}{1 + C_y(s)G(s)}}_{\text{Regulatory control transfer function}} D(s). \tag{26}$$

Closed-loop stability with respect to disturbance depends only on $C_y(s)$, and not on the set-point weighting factors β and γ in $C_r(s)$. Thus, when tuning the beta-gamma controller, $C_y(s)$ should be tuned first for good *regulatory control* performance (i.e., disturbance rejection), then β and γ can be tuned for good *servo control* performance (i.e., good reference tracking with quick transient response) [AV16]. The servo control transfer function in Eq. (26) is also called a set-point filter or reference filter in Lecture 3.

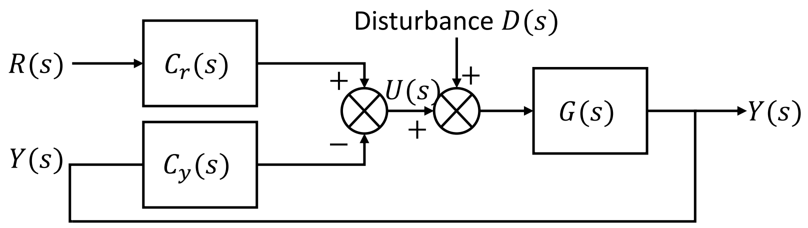


Figure 17: A system with a beta-gamma controller in the two degree-of-freedom configuration, where $C_r(s)$ and $C_y(s)$ are as defined in Eqs. (24) and (25).

Example 4

To visualize set-point weighting, run the Simulink demo `sldemo_pid2dof` by typing "`sldemo_pid2dof`" on a MATLAB Command Window (see Figure 18). The plots in Figure 19 are obtained by setting $\beta = 1$, $\gamma = 1$, thereby disabling the beta-gamma controller. Observe the spikes in the control action. The plots in Figure 20 are obtained by setting $\beta = 0$, $\gamma = 0$, thereby suppressing proportional kicks and derivative kicks to the fullest.

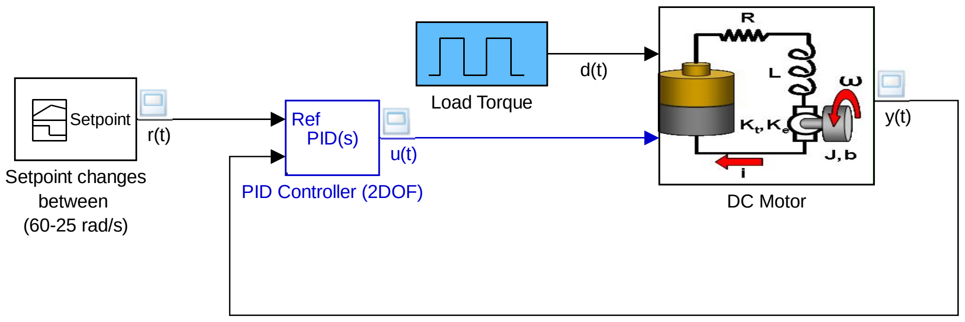


Figure 18: Block diagram of the Simulink demo `sldemo_pid2dof`, which uses the tunable built-in block "PID Controller (2DOF)".

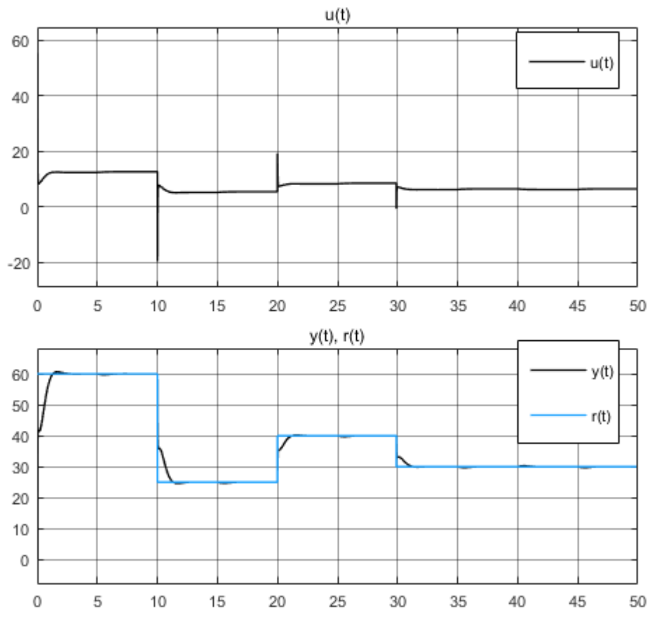


Figure 19: Plots for Example 4 with $\beta = 1$, $\gamma = 1$.

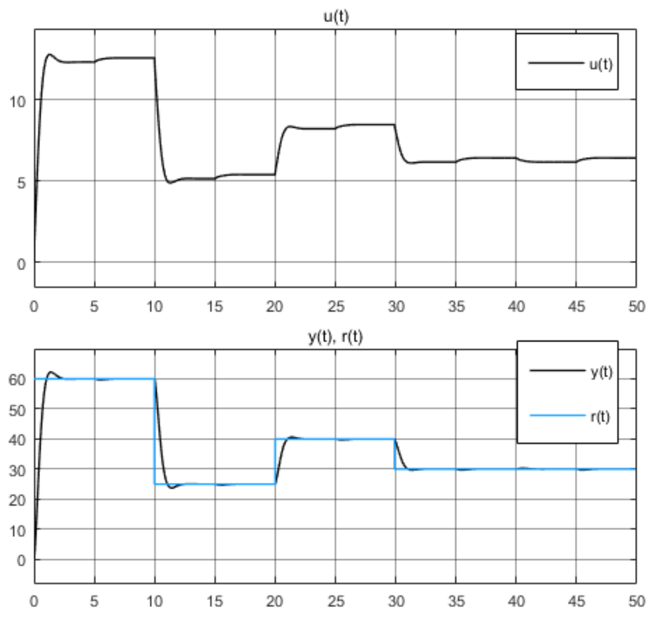


Figure 20: Plots for Example 4 with $\beta = 0$, $\gamma = 0$.

7 Software implementation

So far, we have seen various forms of the PID control law, all in continuous time, but in software, these laws have to be implemented in discrete time.

When implementing a PID controller in software, we expect it to be given in the parallel form. If given in another form, say the series form, we can readily convert it to the parallel form:

$$C_{pid}(s) = K_{pid}\left(\frac{s + z_{pi}}{s}\right)(s + z_{pd}) = K_p + \frac{K_i}{s} + K_d s, \tag{27}$$

where $K_p = K_{pid}(z_{pi} + z_{pd})$ is the proportional gain, $K_i = K_{pid} z_{pi} z_{pd}$ is the integral gain, and $K_d = K_{pid}$ is the derivative gain. Thus in continuous time, a PID control law in the parallel form looks like

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\,\mathrm{d}\tau + K_d \dot{e}(t), \tag{28}$$

where $u(t)$ is the control signal, and $e(t)$ is the control error (reference input minus output). In discrete time, the above translates to the following so-called *positional algorithm* [Vis06, Eq. (1.36)]:

$$u[k] = K_p e[k] + \underbrace{K_i T \sum_{i=0}^{k} e[i]}_{u_i[k]} + \underbrace{K_d \frac{e[k] - e[k-1]}{T}}_{u_d[k]}, \tag{29}$$

where T is the sample period, and we have applied the *backward rectangular version of Euler's method* [FPW98, p. 69] to approximate differentiation:

$$\dot{e}(t = kT) \approx \frac{e[k] - e[k-1]}{T}, \qquad \text{for small } T.$$

Note that by convention, we write $e[k]$ to mean $e(t = kT)$ to emphasize we are dealing with a discrete-time value. Clearly, both integration and differentiation are approximations. By subtracting $u[k-1]$ from $u[k]$, we get the following so-called *incremental/velocity algorithm* [Vis06, Eq. (1.37)]:

$$u[k] = u[k-1] + \left(K_p + K_i T + \frac{K_d}{T}\right) e[k] - \left(K_p + \frac{2K_d}{T}\right) e[k-1] + \frac{K_d}{T} e[k-2],$$

which can be used to derive the controller transfer function in the z domain (detail omitted).

In practice, the PID controller is often implemented as a hybrid of the positional and velocity algorithms, where the proportional term is positional, while the integral and derivative terms ($u_i[k]$ and $u_d[k]$) are incremental. Furthermore,

- $u_i[k]$ is coupled with an anti-windup scheme for suppressing integrator windup, whereas
- $u_d[k]$ is coupled with a derivative filter, i.e., a low-pass filter for suppressing noise.

To implement back-calculation anti-windup, the integral action in discrete time is readily derivable from Eq. (8) as

$$u_i[k] = u_i[k-1] + T\left\{K_i e[k] + K_a(u_{\text{sat}}[k] - u[k])\right\}, \tag{30}$$

where K_a is the back-calculation gain, $u_{\text{sat}}[k]$ is the saturated version of $u[k]$, and $u[k]$ is as defined in Eq. (29).

To implement a derivative filter, consider the transfer function

$$\begin{aligned} \frac{U_d(s)}{E(s)} = \frac{K_p T_d s}{T_f s + 1} &\implies U_d(s)(T_f s + 1) = K_p T_d s E(s) \\ &\implies T_f \dot{u}_d(t) + u_d(t) = K_p T_d \dot{e}(t), \end{aligned} \tag{31}$$

where $u_d(t)$ denotes derivative action, and $e(t)$ denotes control error. To implement Eq. (31), we need to discretize the derivatives, for which we can apply the backward rectangular version of Euler's method again. Discretizing Eq. (31) accordingly, we have

$$\begin{aligned} T_f \frac{u_d[k] - u_d[k-1]}{T} + u_d[k] &= K_p T_d \frac{e[k] - e[k-1]}{T} \\ \implies (T_f + T) u_d[k] &= T_f u_d[k-1] + K_p T_d (e[k] - e[k-1]). \end{aligned} \tag{32}$$

Therefore, the filtered derivative action in discrete time is

$$u_d[k] = \frac{T_f}{T_f + T} u_d[k-1] + \frac{K_p T_d}{T_f + T}(e[k] - e[k-1]). \tag{33}$$

8 Hardware implementation

Improper transfer functions are not physically realizable, for example, a PD controller is not physically realizable. Any physically unrealizable transfer function is naturally not *practically* realizable in hardware. For example, A PD controller is physically unrealizable because it has an improper transfer function; and not practically realizable in hardware because it requires pure differentiation.

Pure differentiation requires an *ideal differentiator*, but the gain of the ideal differentiator increases with frequency, and past some high enough frequency, nevermind the noise completely overwhelms the differentiated output, the differentiator also becomes oscillatory [BG10, Sect. 2.32.4]. This is the reason why the *practical differentiator*, rather than the ideal differentiator, is used in practice, providing the basis for the *lead compensator*. A lead compensator has transfer function of the form

$$C(s) = \frac{K(s+z)}{s+p}, \quad \text{where } z < p \text{ and } z, p \in \mathbb{R}^+, \tag{34}$$

which has a negative zero, and a pole to the left of the zero (see Figure 21). A lead compensator is in fact a PD controller with integrated first-order low-pass filtering.

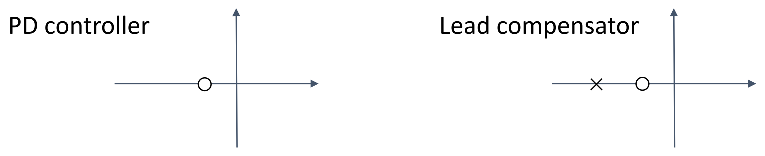


Figure 21: PD controller vs. lead compensator in terms of pole and zero.

On top of that, some physically realizable transfer functions are not practically realizable in hardware; for example, any transfer function with an integrator pole (i.e., pole at the origin) is not practically realizable in hardware because that requires pure integration.

Pure integration requires an *ideal integrator*, but the "bandwidth" of an ideal integrator is so small that it can only be used for a small input frequency range [SB08, p. 205]; this limitation makes the ideal integrator impractical. The result of approximating pure integration in hardware is the *practical integrator* (aka *lossy integrator*) [SB08, p. 205], which provides the basis for the *lag compensator*. A lag compensator has transfer function of the form

$$C(s) = \frac{K(s+z)}{s+p}, \quad \text{where } z > p \text{ and } z, p \in \mathbb{R}^+, \tag{35}$$

which has a *small* negative pole, and a zero to the left of the pole (see Figure 22). Therefore, a lag compensator is a PI controller with its pole slightly left-shifted from origin. When connecting a lag compensator and a lead compensator in series, we get a *lag-lead compensator*.

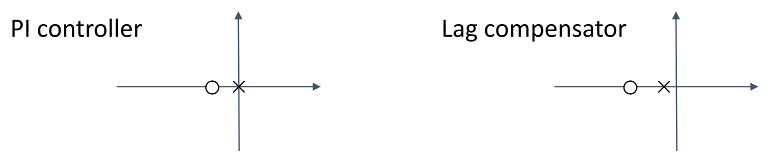


Figure 22: PI controller vs. lag compensator in terms of pole and zero.

In summary,

- A P controller is both physically realizable and practically realizable in hardware.
- A PI controller is physically realizable but not practically realizable in hardware. As a solution, it can be implemented as a *lag compensator* in hardware, which has a small negative rather than zero pole.
- A PD controller is not physically realizable. As a solution, it can be implemented as a *lead compensator* in hardware, which is essentially a PD controller with a derivative filter.
- A PID controller is not physically realizable. As a solution, it can be implemented as a *lag-lead compensator* in hardware.
- A PIDF controller is physically realizable but not practically realizable in hardware. As a solution, it can be implemented as a *lag-lead compensator* in hardware.

In terms of hardware components, lag, lead and lag-lead compensators can be implemented using operational amplifiers (op-amps). There are two types of op-amps: inverting and noninverting — the primary difference between the two is that the inverting type inverses the polarity of the input voltage at the output. Figure 23 shows a Simscape model consisting of an inverting op-amp and a noninverting op-amp. Notice for both types of op-amps, the output voltage is fed back to the negative input. We shall discuss inverting and noninverting op-amps in turn.

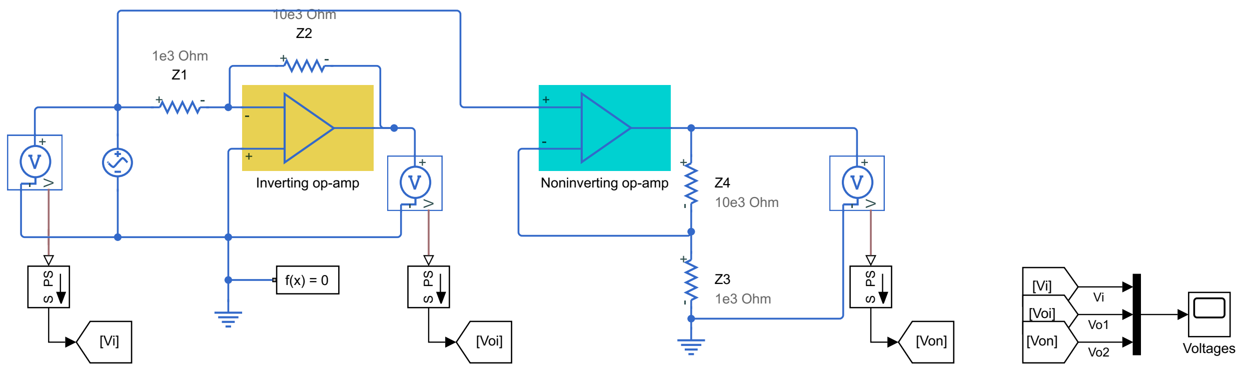


Figure 23: A Simscape model consisting of an inverting op-amp and a noninverting op-amp.

Based on the derivation in [SS15, Sect. 2.2.1], the transfer function of the inverting op-amp in Figure 23 is

$$\frac{V_{oi}(s)}{V_i(s)} = -\frac{Z_2(s)}{Z_1(s)}. \tag{36}$$

We can implement a lag or lead compensator using an inverting op-amp by implementing impedances $Z_1(s)$ and $Z_2(s)$ as parallel RC circuits. The impedances are given by

$$Z_j(s) = \frac{1}{\frac{1}{R_j} + C_j s} = \frac{R_j}{R_j C_j s + 1}, \qquad \text{where } j = 1, 2. \tag{37}$$

Therefore,

$$\frac{V_{oi}(s)}{V_i(s)} = -\frac{Z_2(s)}{Z_1(s)} = -\frac{\frac{R_2}{R_2C_2s+1}}{\frac{R_1}{R_1C_1s+1}} = -\left(\frac{C_1}{C_2}\right)\frac{s+(R_1C_1)^{-1}}{s+(R_2C_2)^{-1}}. \tag{38}$$

By choosing R_1C_1 smaller than R_2C_2, we get a lag compensator; otherwise, we get a lead compensator. However, inverting op-amps have the disadvantage of having a low input impedance.

Based on the derivation in [SS15, Sect. 2.3.1], the transfer function of the noninverting op-amp in Figure 23 is

$$\frac{V_{on}(s)}{V_i(s)} = 1 + \frac{Z_4(s)}{Z_3(s)}. \tag{39}$$

We can implement a lag or lead compensator using a noninverting op-amp by implementing $Z_3(s)$ and $Z_4(s)$ as parallel RC circuits, but the detail is omitted here. Noninverting op-amps have the advantage of having a high input impedance, but Eq. (39) implies they cannot implement a gain of less than 1.

9 Limitations of PID control

We first discuss PI controllers, then PID controllers.

PI control is adequate for all processes whose dynamics is essentially of the first order [ÅH06, Sect. 3.6], e.g., level control for single tanks, a car's brake system. However, it cannot control double integrating processes, i.e., processes with two poles at the origin — for these processes, derivative action is necessary. An example of a double integrating process is the ball and beam system (see Figure 24).

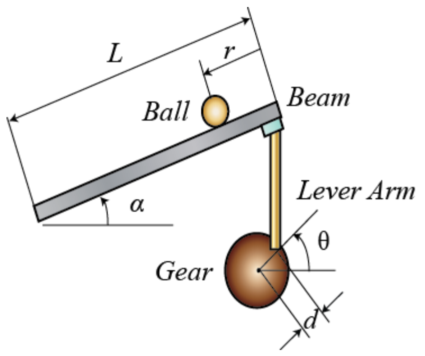


Figure 24: The ball and beam system, where the objective is to balance the ball on the beam, is a double integrating process and a classic test system for control schemes. Model and image available at `http://ctms.engin.umich.edu/CTMS/index.php?example=BallBeam&section=SystemModeling`.

PID controllers are second-order (at most) controllers, so there exist unstable plants that cannot be stabilized by any member of the PID family. In particular, PID controllers are problematic [Cor04, Sect. 4.4.2] for

- systems with substantial time delay;
- systems with oscillatory modes (poles on the imaginary axis);
- systems with large parameter variations.

Acknowledgement

David Challender proofread the 2016 version of this manuscript.

References

[ACL05] Kiam Heong Ang, Gregory. Chong, and Yun Li. PID control system analysis, design, and technology. *IEEE Transactions on Control Systems Technology*, 13(4):559–576, July 2005.

[ÅH95] K.J. Åström and T. Hägglund. *PID Controllers: Theory, Design and Tuning*. ISA Press, 1995.

[ÅH06] Karl Johan Åström and Tore Hägglund. *Advanced PID control*. ISA-Instrumentation, Systems, and Automation Society, 2006.

[AV16] Victor M. Alfaro and Ramon Vilanova. *Model-Reference Robust Tuning of PID Controllers*. Springer International Publishing, 2016.

[BG10] U.A. Bakshi and A.P. Godse. *Linear integrated circuits & applications*. Technical Publications, 2010.

[Cor04] J.-P. Corriou. *Process Control: Theory and Applications*. Springer-Verlag London Ltd., 2004.

[DM02] Lane Desborough and Randy Miller. Increasing customer value of industrial control performance monitoring—Honeywell's experience. In *AIChE symposium series*, pages 169–189. New York; American Institute of Chemical Engineers; 1998, 2002.

[FPW98] Gene Franklin, J. David Powell, and Michael Workman. *Digital Control of Dynamic Systems*. Addison Wesley Longman, Inc., third edition, 1998.

[Hon14] Honeywell Process Solutions. *Experion LX Control Builder Components: Theory*, release 110 edition, February 2014.

[LAC06] Y. Li, K.H. Ang, and G.C.Y. Chong. Patents, software, and hardware for pid control: an overview and analysis of the current art. *IEEE Control Systems*, 26(1):42–54, 2006.

[Lip06] Béla Lipták. *Instrument Engineers' Handbook: Process Control and Optimization: Volume II*. ISA-Instrumentation, Systems, and Automation Society, 2006.

[O'D09] Aidan O'Dwyer. *Handbook of PI and PID controller tuning rules*, volume 57. World Scientific, 2009.

[SB08] S. Salivahanan and V.S. Kanchana Bhaaskaran. *Linear Integrated Circuits*. Tata McGraw-Hill, 2008.

[Sie07] Siemens. DCS or PLC? Seven Questions to Help You Select the Best Solution. white paper, 2007.

[SMEDI10] Dale E Seborg, Duncan A Mellichamp, Thomas F Edgar, and Francis J Doyle III. *Process dynamics and control.* John Wiley & Sons, 3rd edition, 2010.

[SS15] Adel S. Sedra and Kenneth C. Smith. *Microelectronic Circuits.* Oxford University Press, 7th edition, 2015.

[Vis06] Antonio Visioli. *Practical PID control.* Springer-Verlag London Limited, 2006.

[VV12] Ramon Vilanova and Antonio Visioli, editors. *PID Control in the Third Millennium: Lessons Learned and New Approaches.* Springer-Verlag London, 2012.