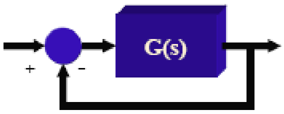
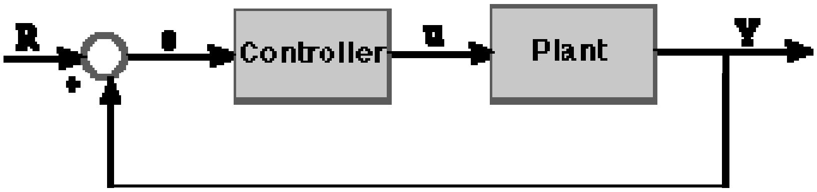
PID Controller Design

Ref: Chapter 9,10,11: **Dorf, R. C.** & **Bishop, R. H.** ,
Modern Control Systems
Chapter 9: **Nise, N. S.**
Control System Engineering





Plant: A system to be controlled
Controller: Provides the excitation for the plant;
Designed to control the overall system behavior

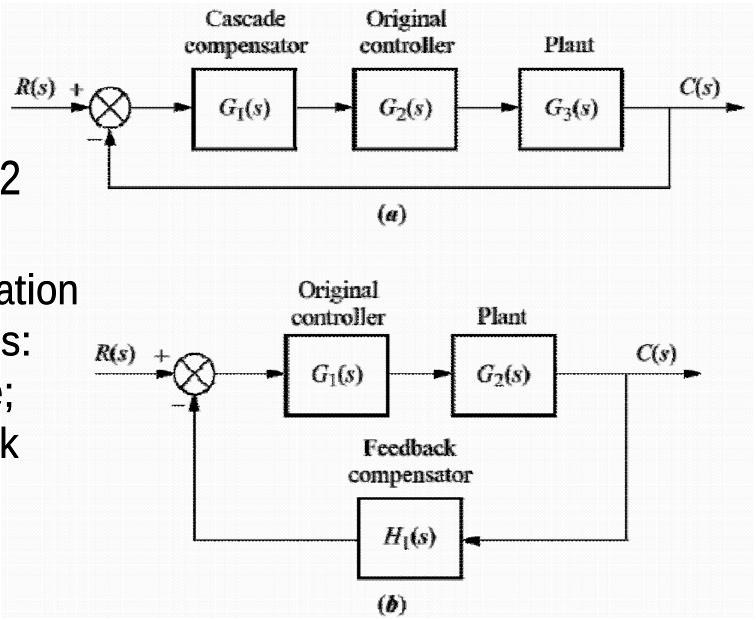
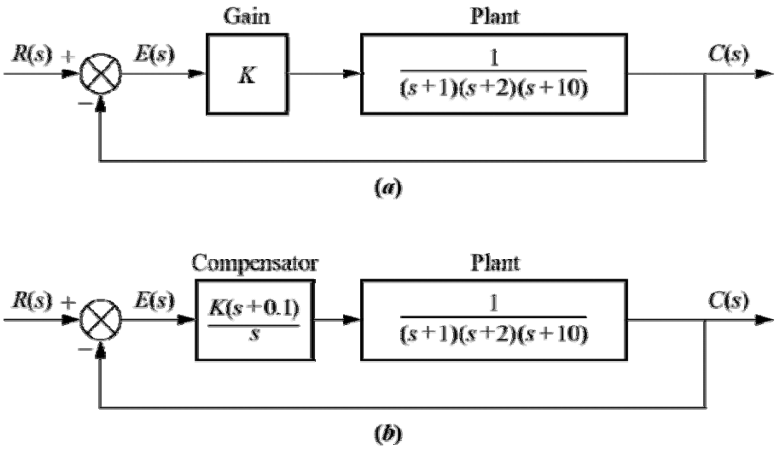


Figure 9.2 (Nise)
Compensation techniques:
a.cascade;
b.feedback

Figure 9.4 (Nise) : Closed-loop system for Example 9.1:

a. Before compensation;

b. After ideal integral compensation



Control Tutorials for Matlab

PID Tutorial

Introduction

The three-term controller

The characteristics of P, I, and D controllers

Example Problem

Open-loop step response
Proportional control
Proportional-Derivative control
Proportional-Integral control
Proportional-Integral-Derivative control

General tips for designing a PID controller

Basic Control Actions: u(t)

Proportional control: $u(t) = K_p e(t)$ $\qquad \dfrac{U(s)}{E(s)} = K_p$

Integral control: $u(t) = K_i \int_0^t e(t)dt$ $\qquad \dfrac{U(s)}{E(s)} = \dfrac{K_i}{s}$

Differential control: $u(t) = K_d \dfrac{d}{dt} e(t)$ $\qquad \dfrac{U(s)}{E(s)} = K_d s$

Effect of Control Actions

- **Proportional Action**
  - Adjustable gain (amplifier)
- **Integral Action**
  - Eliminates bias (steady-state error)
  - Can cause oscillations
- **Derivative Action (“rate control”)**
  - Effective in transient periods
  - Provides faster response (higher sensitivity)
  - Never used alone

Basic Controllers

- Proportional control is often used by itself
- Integral and differential control are typically used in combination with at least proportional control
  - eg, Proportional Integral (PI) controller:

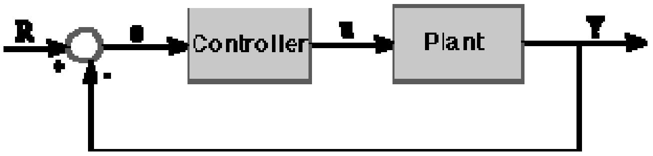
$$\boxed{G(s)=\frac{U(s)}{E(s)}=K_p+\frac{K_I}{s}=K_p\left(1+\frac{1}{T_i s}\right)}$$

Summary of Basic Control

- **Proportional control**
  - Multiply e(t) by a constant
- **PI control**
  - Multiply e(t) and its integral by separate constants
  - Avoids bias for step
- **PD control**
  - Multiply e(t) and its derivative by separate constants
  - Adjust more rapidly to changes
- **PID control**
  - Multiply e(t), its derivative and its integral by separate constants
  - Reduce bias and react quickly

Introduction

This tutorial will show you the characteristics of the each of **proportional (P), the integral (I),** and the **derivative (D)** controls, and how to use them to obtain a desired response. In this tutorial, we will consider the following unity feedback system:



Plant: A system to be controlled

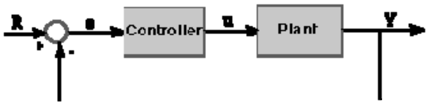
Controller: Provides the excitation for the plant;
Designed to control the overall system behavior

The three-term controller

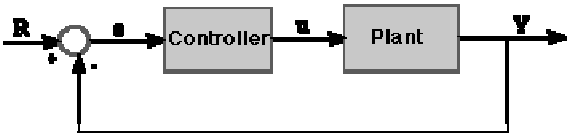
The transfer function of the PID controller looks like the following:

$$K_P + \frac{K_I}{s} + K_D s = \frac{K_D s^2 + K_P s + K_I}{s}$$

- Kp = Proportional gain
- KI = Integral gain
- Kd = Derivative gain



First, let's take a look at how the PID controller works in a closed-loop system using the schematic shown above. The variable (e) represents the tracking error, the difference between the desired input value (R) and the actual output (Y).



This error signal (e) will be sent to the PID controller, and the controller computes both the derivative and the integral of this error signal.

The signal (u) just past the controller is now equal to the proportional gain (Kp) times the magnitude of the error plus the integral gain (Ki) times the integral of the error plus the derivative gain (Kd) times the derivative of the error.

$$u = K_P e + K_I \int e\,dt + K_D \frac{de}{dt}$$

This signal (u) will be sent to the plant, and the new output (Y) will be obtained. This new output (Y) will be sent back to the sensor again to find the new error signal (e). The controller takes this new error signal and computes its derivative and its integral again. This process goes on and on.

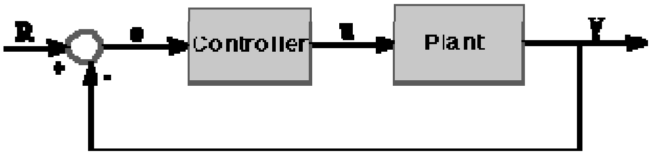


Figure 9.8 (Nise) PI controller

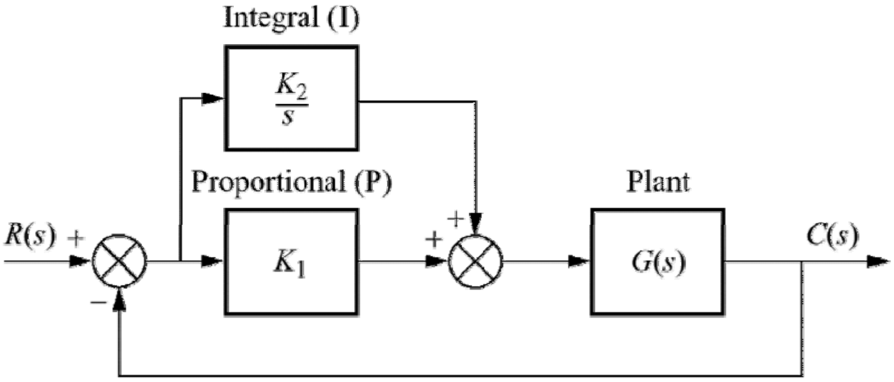


Figure 9.23 (Nise)
PD controller

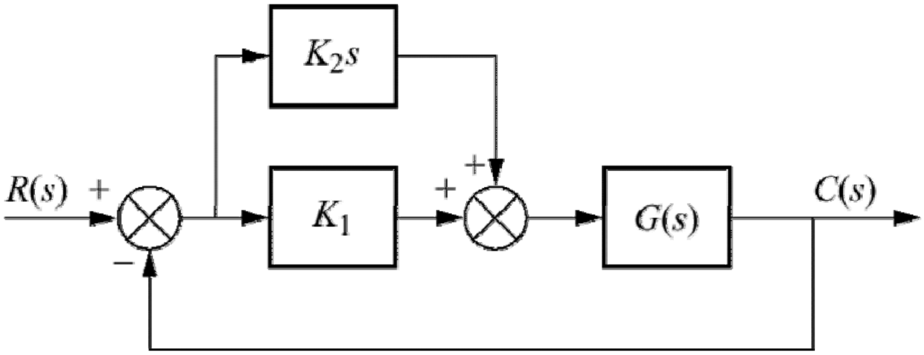
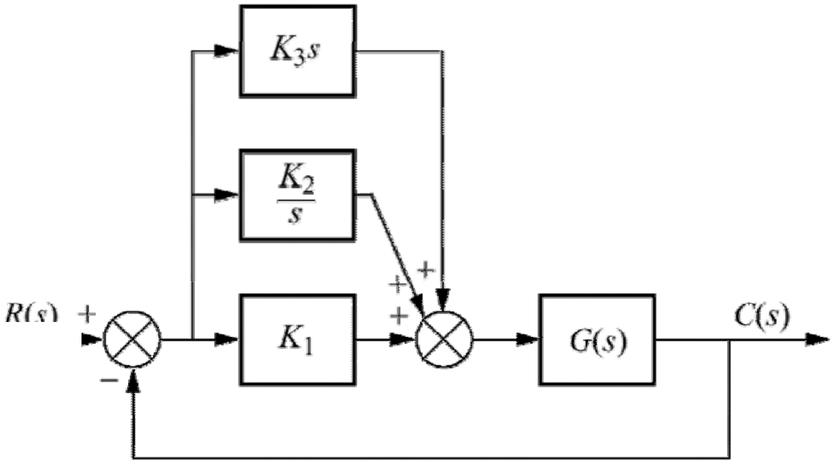


Figure 9.30 (Nise)
PID controller



The characteristics of P, I, and D controllers

A proportional controller (Kp) will have the effect of reducing the rise time and will reduce ,but never eliminate, the **steady-state error**.

An integral control (Ki) will have the effect of eliminating the steady-state error, but it may make the transient response worse.

A derivative control (Kd) will have the effect of increasing the stability of the system, reducing the overshoot, and improving the transient response.

Effects of each of controllers Kp, Kd, and Ki on a closed-loop system are summarized in the table shown below.

CL RESPONSE	RISE TIME	OVERSHOOT	SETTLING TIME	S-S ERROR
Kp	Decrease	Increase	Small Change	Decrease
Ki	Decrease	Increase	Increase	Eliminate
Kd	Small Change	Decrease	Decrease	Small Change

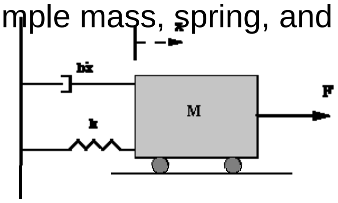
Note that these correlations may not be exactly accurate, because Kp, Ki, and Kd are dependent of each other.

In fact, changing one of these variable can change the effect of the other two.

For this reason, the table should only be used as a reference when you are determining the values for Ki, Kp and Kd.

Example Problem

Suppose we have a simple mass, spring, and damper problem.



The modeling equation of this system is

$$M\ddot{x} + b\dot{x} + kx = F \qquad (1)$$

Taking the Laplace transform of the modeling equation (1)

$$Ms^2X(s) + bsX(s) + kX(s) = F(s)$$

The transfer function between the displacement **X(s)** and the input **F(s)** then becomes

$$\frac{X(s)}{F(s)} = \frac{1}{Ms^2 + bs + k}$$

Let

- ***M = 1kg***
- ***b = 10 N.s/m***
- ***k = 20 N/m***
- ***F(s) = 1***

Plug these values into the above transfer function

$$\frac{X(s)}{F(s)} = \frac{1}{s^2 + 10s + 20}$$

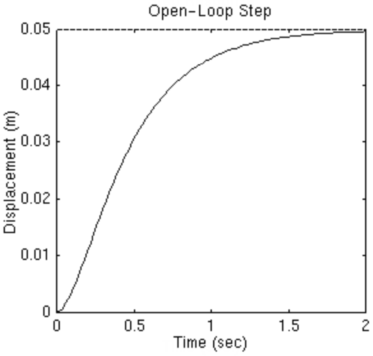
The goal of this problem is to show you how each of Kp, Ki and Kd contributes to obtain

- Fast rise time
- Minimum overshoot
- No steady-state error

Open-loop step response

Let's first view the open-loop step response. Create a new m-file and add in the following code:

```
num=1;
den=[1 10 20];
step(num,den)
```

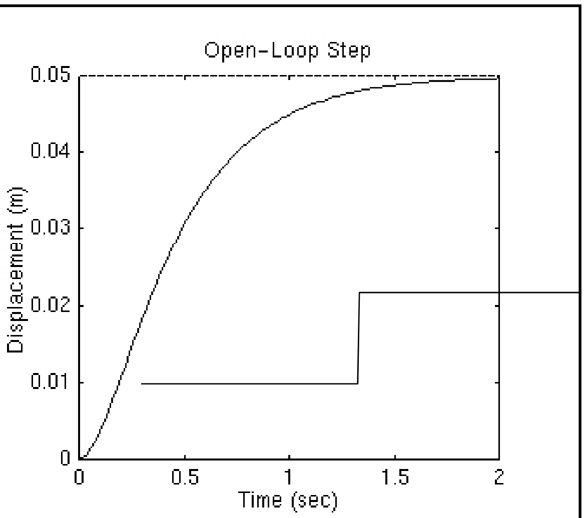


Running this m-file in the Matlab command window should give you the plot shown above.

The DC gain of the plant **transfer function** is 1/20, so 0.05 is the final value of the output to an unit step input.

This corresponds to the **steady-state error of 0.95**, quite large indeed.

Furthermore, the rise time is about one second, and the **settling time** is about 1.5 **seconds**.



Let's design a controller that will reduce the rise time, reduce the settling time, and eliminates the steady-state error.

Proportional control (P Controller)

From the table (shown bellow), we see that the proportional controller (Kp)
reduces the **rise time**,
increases the **overshoot**, and
reduces the **steady-state error**.

CL RESPONSE	RISE TIME	OVERSHOOT	SETTLING TIME	S-S ERROR
Kp	Decrease	Increase	Small Change	Decrease
Ki	Decrease	Increase	Increase	Eliminate
Kd	Small Change	Decrease	Decrease	Small Change

Proportional control (P Controller)

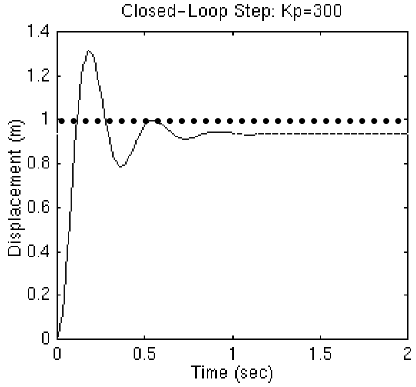
The closed-loop transfer function of the system with a proportional controller is:

$$\frac{X(s)}{F(s)} = \frac{K_p}{s^2 + 10s + (20 + K_p)}$$

Let the proportional gain (Kp) equals 300 and change the m-file to the following:

```
Kp=300; num=[Kp];
den=[1 10 20+Kp];
t=0:0.01:2;
step(num,den,t)
```

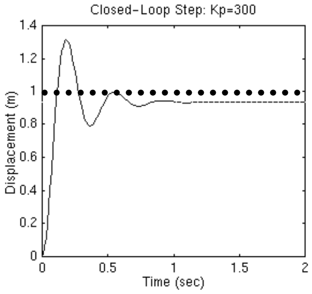
Running this m-file in the Matlab command window should gives you the following plot.



Note: The Matlab function called “cloop” can be used to obtain a closed-loop transfer function directly from the open-loop transfer function (instead of obtaining closed-loop transfer function by hand).

The following m-file uses the cloop command that should give you the identical plot as the one shown above.

```
num=1; den=[1 10 20];
Kp=300;
[numCL,denCL] = cloop(Kp*num,den);
t=0:0.01:2;
step(numCL, denCL,t)
```

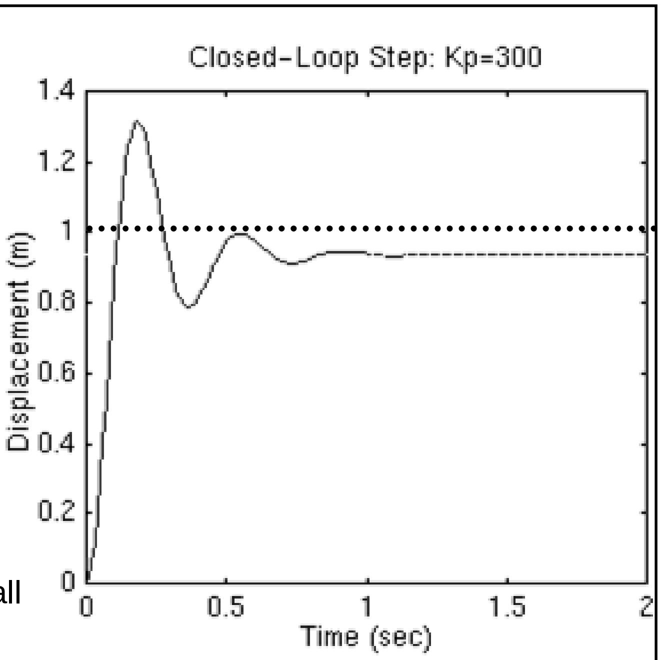


The plot shows that the proportional controller

reduced both the **rise time** and the **steady-state** error,

increased the **overshoot**, and

decreased the **settling time** by small amount.



Proportional-Derivative control (PD Controller)

Now, let's take a look at a PD control. From the table, we see that the derivative controller (Kd) reduces both **the overshoot** and **the settling time**.

CL RESPONSE	RISE TIME	OVERSHOOT	SETTLING TIME	S-S ERROR
Kp	Decrease	Increase	Small Change	Decrease
Ki	Decrease	Increase	Increase	Eliminate
Kd	Small Change	Decrease	Decrease	Small Change

Proportional-Derivative control (PD Controller)

Now, let's take a look at a PD control. From the table, we see that the derivative controller (Kd) reduces both **the overshoot** and **the settling time**.

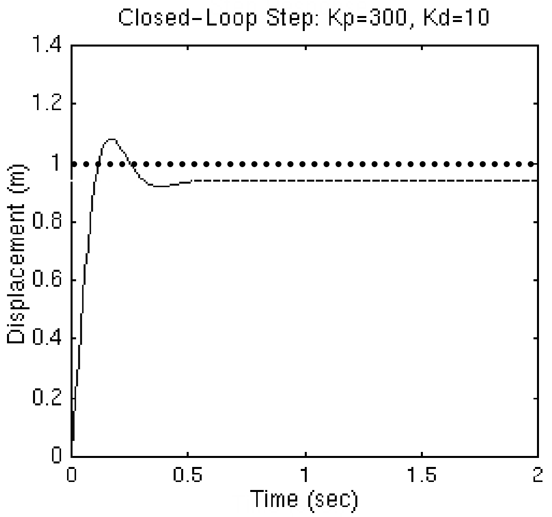
The closed-loop transfer function of the given system with a PD controller is:

$$\frac{X(s)}{F(s)} = \frac{K_D s + K_P}{s^2 + (10 + K_D)s + (20 + K_P)}$$

Let Kp equals to 300 as before and let Kd equals 10.

Enter the following commands into an m-file and run it in the Matlab command window.

```
Kp=300;
Kd=10;
num=[Kd Kp];
den=[1 10+Kd 20+Kp];
t=0:0.01:2;
step(num,den,t)
```



*This plot shows that the derivative controller **reduced** both the **overshoot** and the **settling time**, and had **small effect** on the **rise time** and the **steady-state error**.*

Proportional-Integral control (PI Controller)

Before going into a PID control, let's take a look at a **PI control**. From the table, we see that an integral controller (Ki) decreases the **rise time**,
increases both the **overshoot** and the **settling time**,
and eliminates the **steady-state error**.

CL RESPONSE	RISE TIME	OVERSHOOT	SETTLING TIME	S-S ERROR
Kp	Decrease	Increase	Small Change	Decrease
Ki	Decrease	Increase	Increase	Eliminate
Kd	Small Change	Decrease	Decrease	Small Change

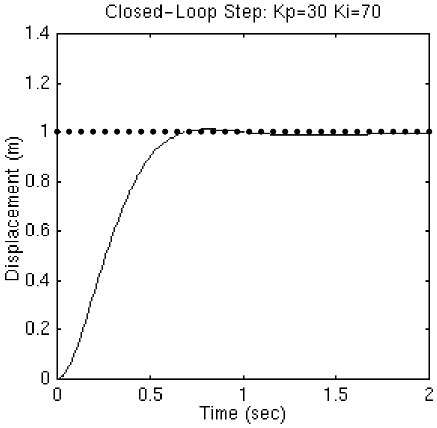
For the given system, the closed-loop transfer function with a PI control is:

$$\frac{X(s)}{F(s)}=\frac{K_p s+K_I}{s^3+10s^2+(20+K_p)s+K_I}$$

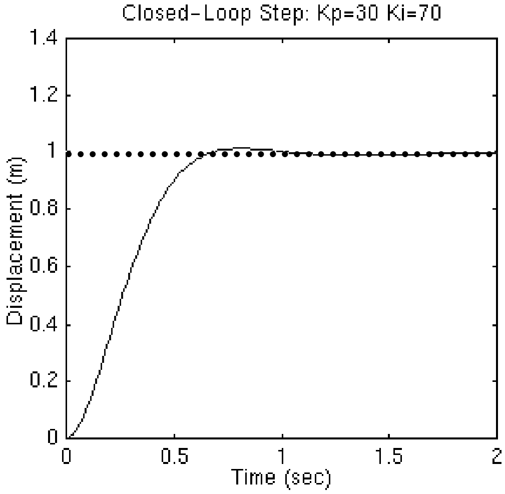
Let's reduce the Kp to 30, and let Ki equals to 70.
Create an new m-file and enter the following commands.

```
Kp=30;
Ki=70;
num=[Kp Ki];
den=[1 10 20+Kp Ki];
t=0:0.01:2;
step(num,den,t)
```

Run this m-file in the Matlab command window, and you should get the following plot.



We have reduced the proportional gain (Kp) *[from 300→30]* because the integral controller also **reduces the rise time** and **increases the overshoot** as the proportional controller does (double effect).



The above response shows that the integral controller **eliminated the steady-state error.**

Proportional-Integral-Derivative control (PID Controller)

Now, let's take a look at a PID controller. The closed-loop transfer function of the given system with a PID controller is:

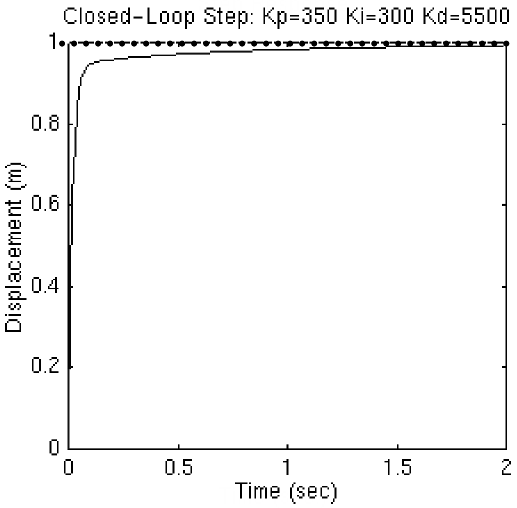
$$\frac{X(s)}{F(s)} = \frac{K_D s^2 + K_P s + K_I}{s^3 + (10 + K_D)s^2 + (20 + K_P)s + K_I}$$

CL RESPONSE	RISE TIME	OVERSHOOT	SETTLING TIME	S-S ERROR
Kp	Decrease	Increase	Small Change	Decrease
Ki	Decrease	Increase	Increase	Eliminate
Kd	Small Change	Decrease	Decrease	Small Change

After several trial and error runs, the gains Kp=350, Ki=300, and Kd=50 provided the desired response.

To confirm, enter the following commands to an m-file and run it in the command window. You should get the following step response.

```
Kp=350;
Ki=300; Kd=50;
num=[Kd Kp Ki];
den=[1 10+Kd 20+Kp Ki];
t=0:0.01:2;
step(num,den,t)
```



Now, we have obtained the system with no overshoot, fast rise time, and no steady-state error.

General tips for designing a PID controller

When you are designing a PID controller for a given system, follow the steps shown below to obtain a desired response.

1. Obtain an open-loop response and determine what needs to be improved
2. Add a proportional control to improve the rise time
3. Add a derivative control to improve the overshoot
4. Add an integral control to eliminate the steady-state error
5. Adjust each of Kp, Ki, and Kd until you obtain a desired overall response. You can always refer to the table shown in this "PID Tutorial" page to find out which controller controls what characteristics.

Lastly, please keep in mind that
you do not need to implement all three controllers (proportional, derivative, and integral) into a single system, if not necessary.

For example, if a PI controller gives a good enough response (like the above example), then you don't need to implement derivative controller to the system.

Keep the controller as simple as possible.

This presentation is made from:

http://www.library.cmu.edu/ctms/ctms/matlab42/pid/pid.htm