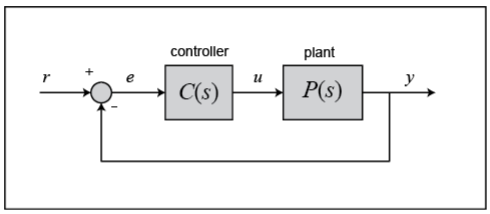
Classic Control Systems MATLAB Examples and Experiments : PID Controller

- This script overviews aspects of control, PID, Root Locus, Frequency Reponse, and other assessments involving Control System Analysis
- Several examples will be demonstrated using MATLAB as a tool for plotting poles/zeros, root locus, magnitude/phase response of system, step responses and more

```
clc; clear; close;
```

PID Overview

```
f = imread('feedback_block.png');
imshow(f)
```



The output of a PID controller, like one shown in image to the right, is:

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de}{dt} \qquad (1)$$

where $e(t)$ is the tracking error, which is the difference between desired output (r) and measured output (y). The error signal is fed to the PID controller, and the controller computes both the derivative and the integral of this error signal with respect to time.

The transfer function of a PID controller can be found by taking the Laplace Transform of (1)

$$K_p + \frac{K_i}{s} + K_d s = \frac{K_d s^2 + K_p s + K_i}{s}$$

where K_p = proportional gain, K_i = integral gain, and K_d = derivative gain.

```
% Define this PID controller using Control Systems Toolbox functions
Kp = 1; Ki = 1; Kd = 1;

s = tf('s');
C = Kp +Ki/s + Kd*s
```

```
C =

  s^2 + s + 1
  -----------
       s

Continuous-time transfer function.
```

We can also use MATLAB's **pid object** to generate an equivalent continuous controller:

```
C = pid(Kp,Ki,Kd)
```

```
C =

             1
  Kp + Ki * --- + Kd * s
             s

  with Kp = 1, Ki = 1, Kd = 1

Continuous-time PID controller in parallel form.
```

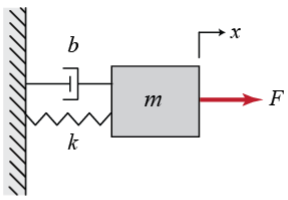
Now convert the **pid object** to a transfer function for verification with above:

```
tf(C)
```

```
ans =

  s^2 + s + 1
  -----------
       s

Continuous-time transfer function.
```

Example Problem: Mass-Spring-Damper System

```
f= imread('mass_spring_damper.png'); imshow(f)
```



The governing equation of this system is

$$m\ddot{x} + b\dot{x} + kx = F$$

Then using taking the Laplace Transform of the governing equation:

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

The transfer function between the input force $F(s)$ and the output displacement $X(s)$ becomes

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

For this example let

```
m = 1; % 1 kg
b = 10; % 10 N s/m
k = 20; % 20 N/m
F = 1; % 1 N
A = [m b k]; B = 1;
p = roots(A)
```

```
p = 2×1
   -7.2361
   -2.7639
```

After substitution, the transfer function then looks like

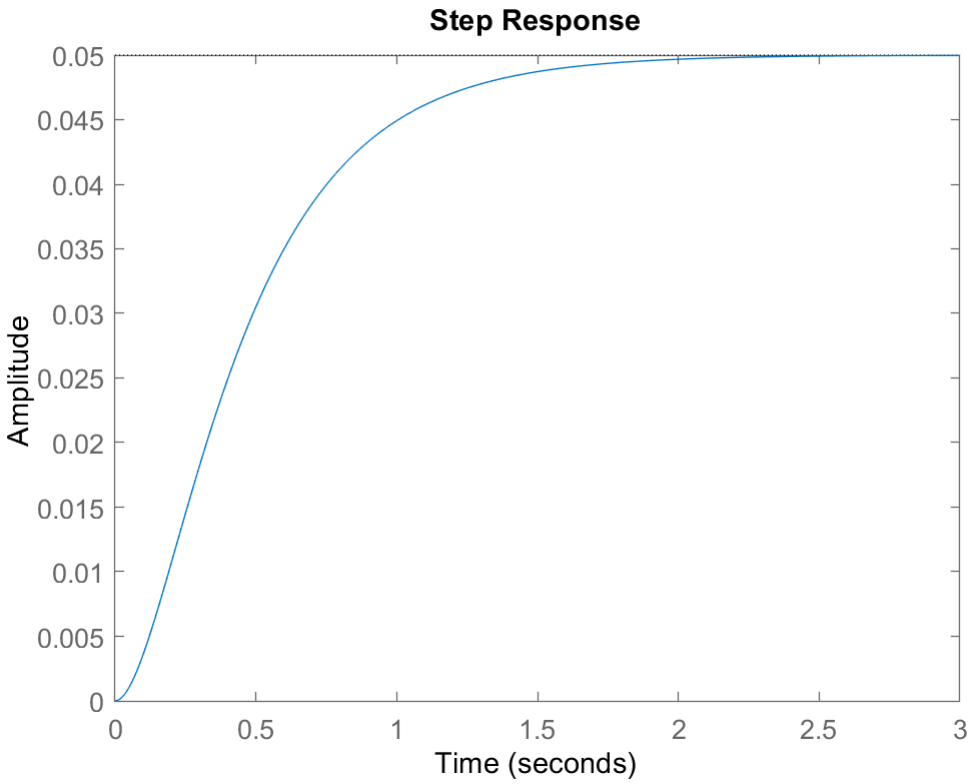
$$\frac{X(s)}{F(s)} = \frac{1}{s^2 + 10s + 20}$$

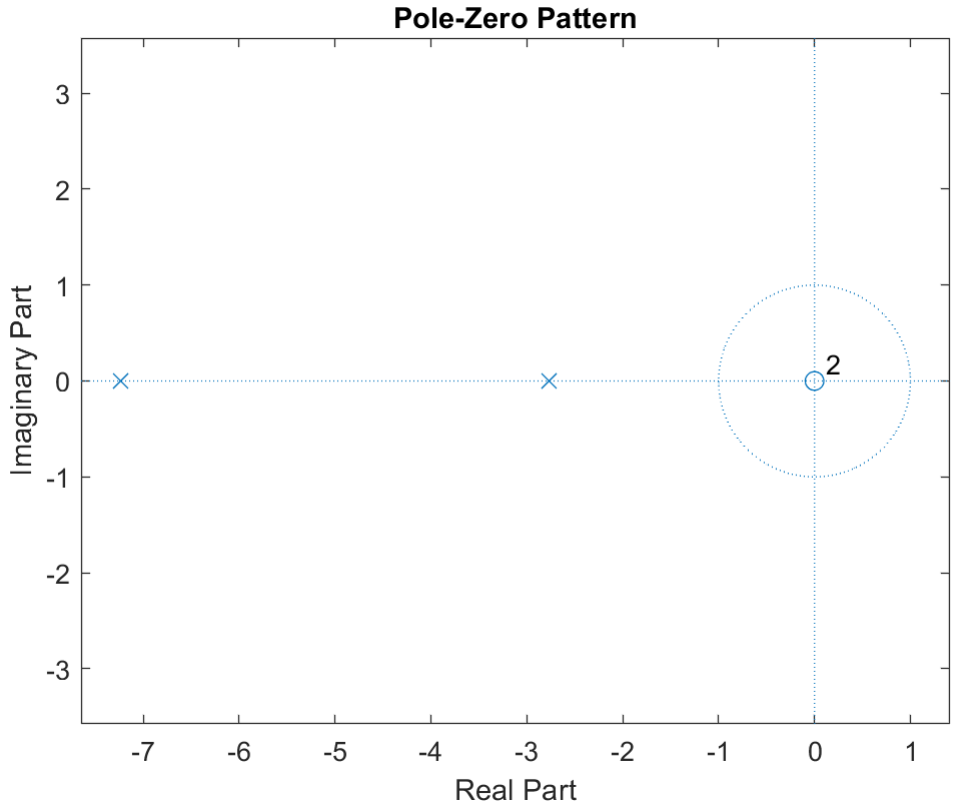
The goal of this problem is to show how each of the terms, K_p, K_i, and K_d contribute to goals of:

- Fast rise time
- Minimal Overshoot
- Zero steady-state error

Open-Loop Response

```
s = tf('s');
P = 1/(m*s^2 + b*s + k);
figure, step(P), figure, zplane(B,A), title('Pole-Zero Pattern')
```


Step Response
Amplitude
Time (seconds)
0
0.005
0.01
0.015
0.02
0.025
0.03
0.035
0.04
0.045
0.05
0
0.5
1
1.5
2
2.5
3


Pole-Zero Pattern
Imaginary Part
Real Part
2
3
2
1
0
-1
-2
-3
-7
-6
-5
-4
-3
-2
-1
0
1

DC gain of the Process transfer function is about 0.05 which is steady-state error of about 95%. The rise time is about 1 second, and the transient response lasts for about 1.5 secs until reaching steady-state. Now, let's design a controller that will:

- reduce the rise time
- reduce the transient response time
- eliminate the steady-state error

Proportional Control

The proportional controller (K_p):

- reduces the rise time
- increases the overshoot
- eliminates the steady-state error

The closed-loop transfer function of our unity-feedback system with proportional controller is

$$T(s) = \frac{Y(s)}{R(s)} = \frac{K_p}{s^2 + 10s + (20 + K_p)}$$

where $Y(s)$is the output, $R(s)$ is the referenced signal, and $T(s)$ is our transfer function

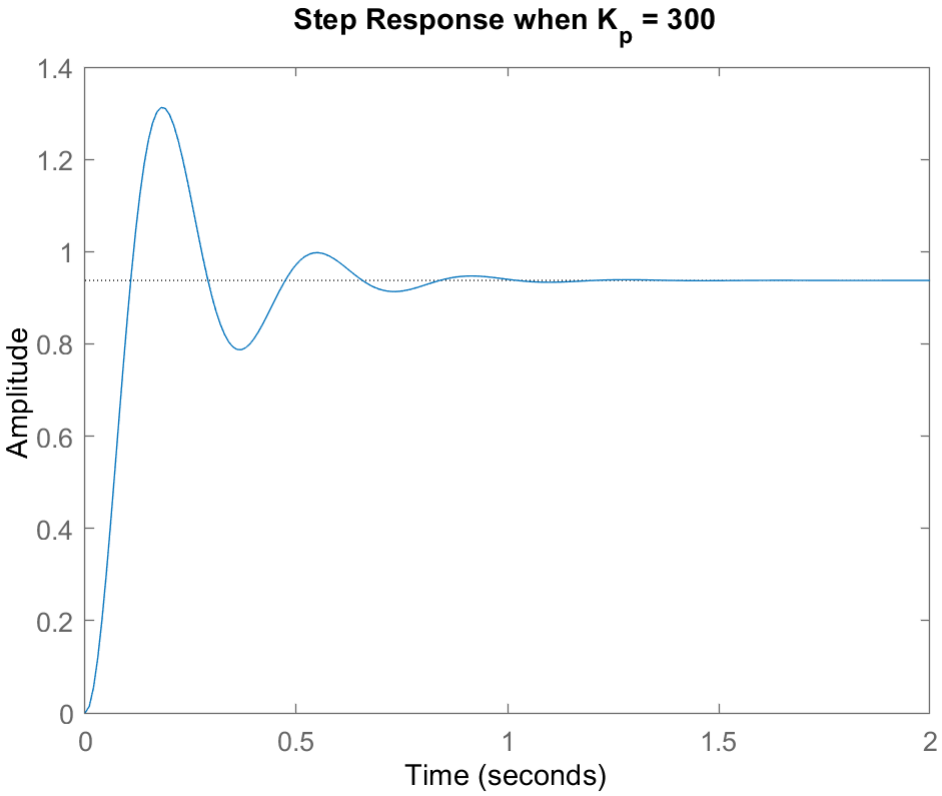
```
% Now let K_p equal to 300 and observe the step response
Kp = 300;
C = pid(Kp)
```

```
C =

  Kp = 300

P-only controller.
```

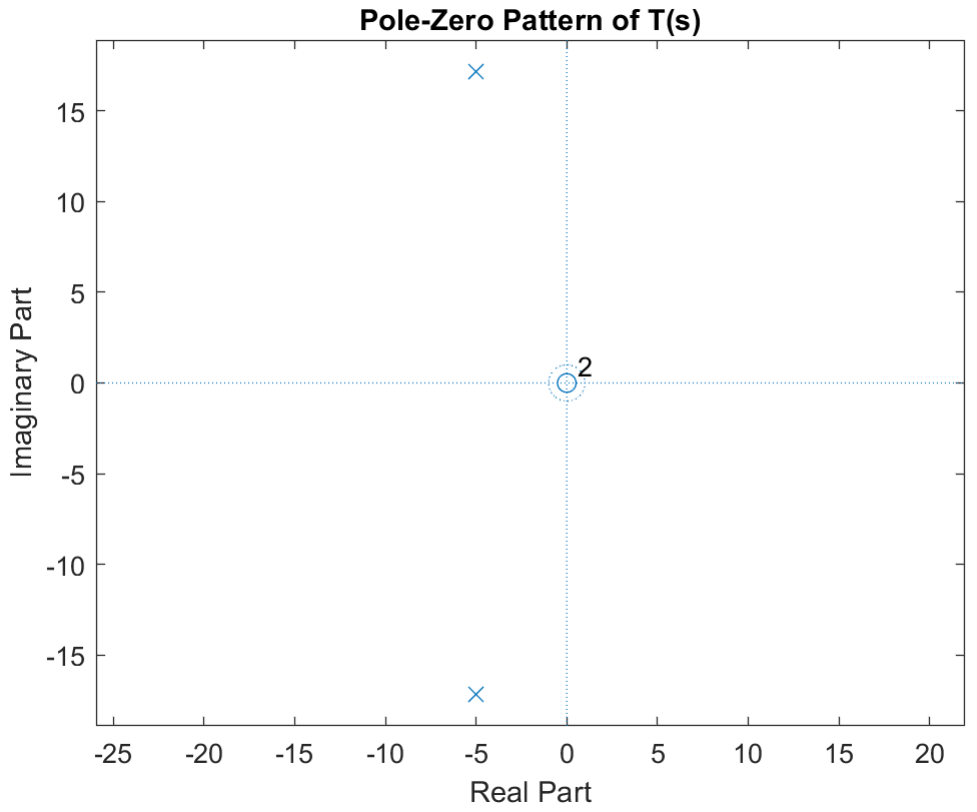
```
T = feedback(C*P,1)
```

```
T =

        300
  ----------------
  s^2 + 10 s + 320

Continuous-time transfer function.
```

```
t = 0:1/100:2; figure, step(T,t), title('Step Response when K_p = 300')
```




```
figure, zplane(Kp,[m b k+Kp]), title('Pole-Zero Pattern of T(s)')
```



Proportional-Derivative Control

The derivative controller (K_d):

- reduces both the overshoot and the transient response time

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

$$T(s) = \frac{Y(s)}{R(s)} = \frac{K_d s + K_p}{s^2 + (10 + K_d)s + (20 + K_p)}$$

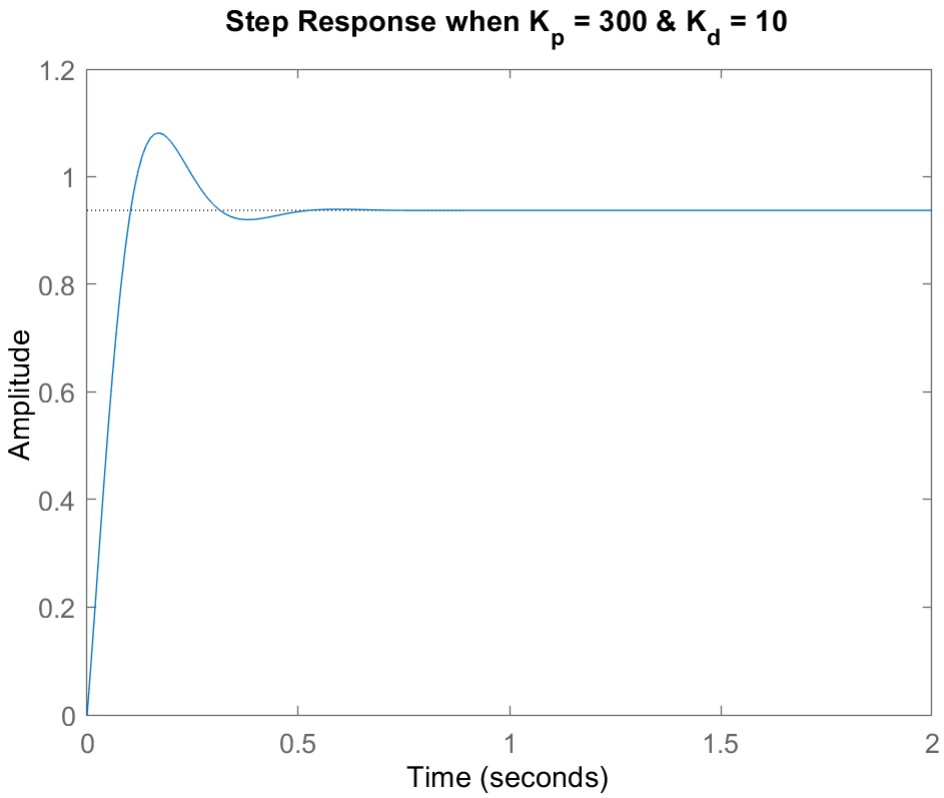
```
% Let Kp = 300 and Kd = 10
Kp = 300;
Kd = 10;
C = pid(Kp,0,Kd)
```

```
C =


  Kp + Kd * s


  with Kp = 300, Kd = 10

Continuous-time PD controller in parallel form.
```

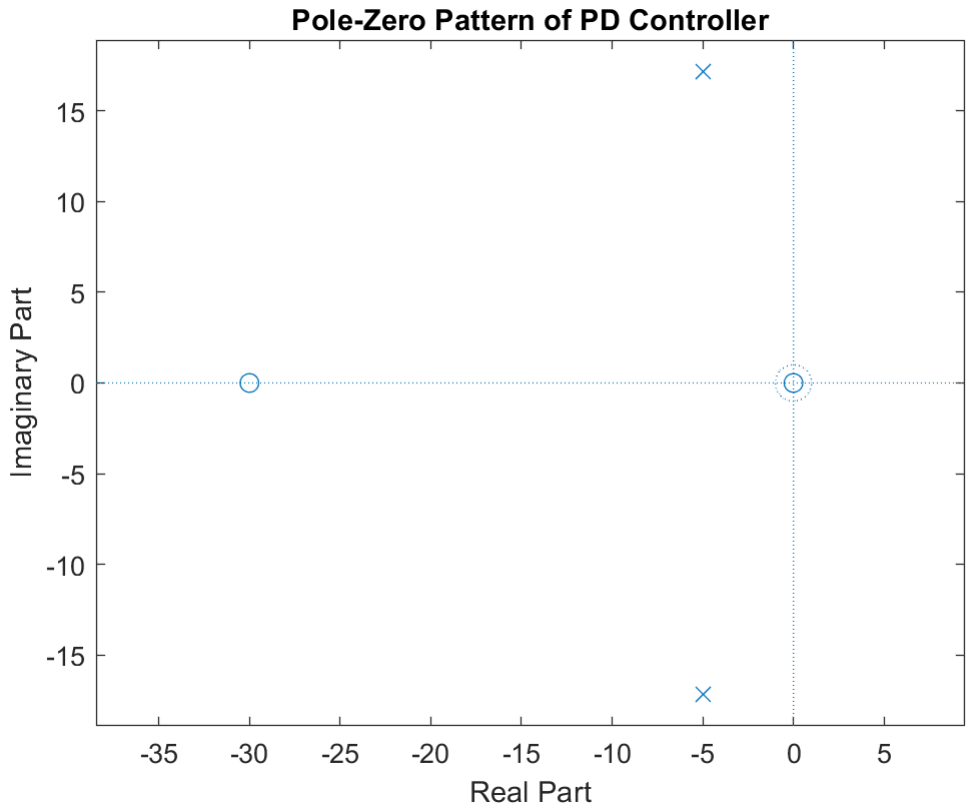
```
P = 1/(s^2 + 10*s + 20);
T = feedback(C*P,1)
```

```
T =

     10 s + 300
  ----------------
  s^2 + 20 s + 320

Continuous-time transfer function.
```

```
t = 0:1/100:2; figure, step(T,t), title('Step Response when K_p = 300 & K_d = 10')
```




```
B = [Kd Kp]; A = [m +Kd k+Kp]; figure, zplane(B,A), title('Pole-Zero Pattern of PD Controller')
```



The resulting step response shows that the addition of the derivative term reduces both the overshoot and the transient response time, and had a negligible effect on the rise time and steady-state error

Proportional-Integral Control

The integral controller (K_i):

- decreases the rise time
- increases both the overshoot and transient reponse
- reduces the steady-state error

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

$$T(s) = \frac{Y(s)}{R(s)} = \frac{K_p s + K_i}{s^3 + 10s^2 + (20 + K_p)s + K_i}$$

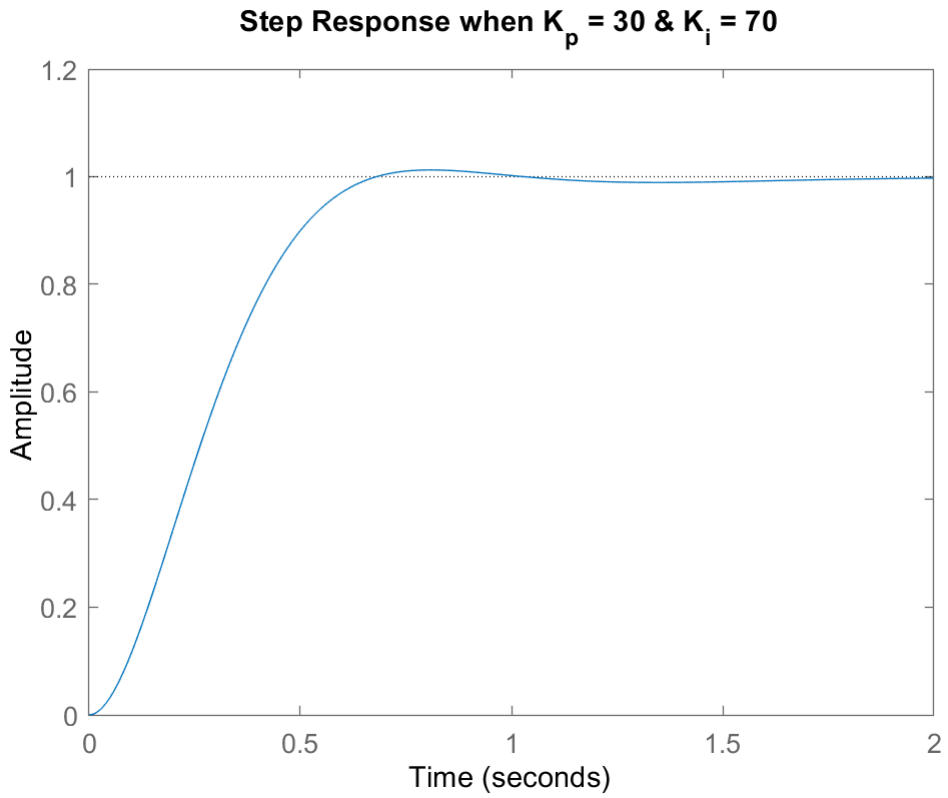
```
% Let Kp = 30 and Ki = 70
Kp = 30;
Ki = 70;
C = pid(Kp,Ki)
```

```
C =

            1
  Kp + Ki * ---
            s

  with Kp = 30, Ki = 70

Continuous-time PI controller in parallel form.
```

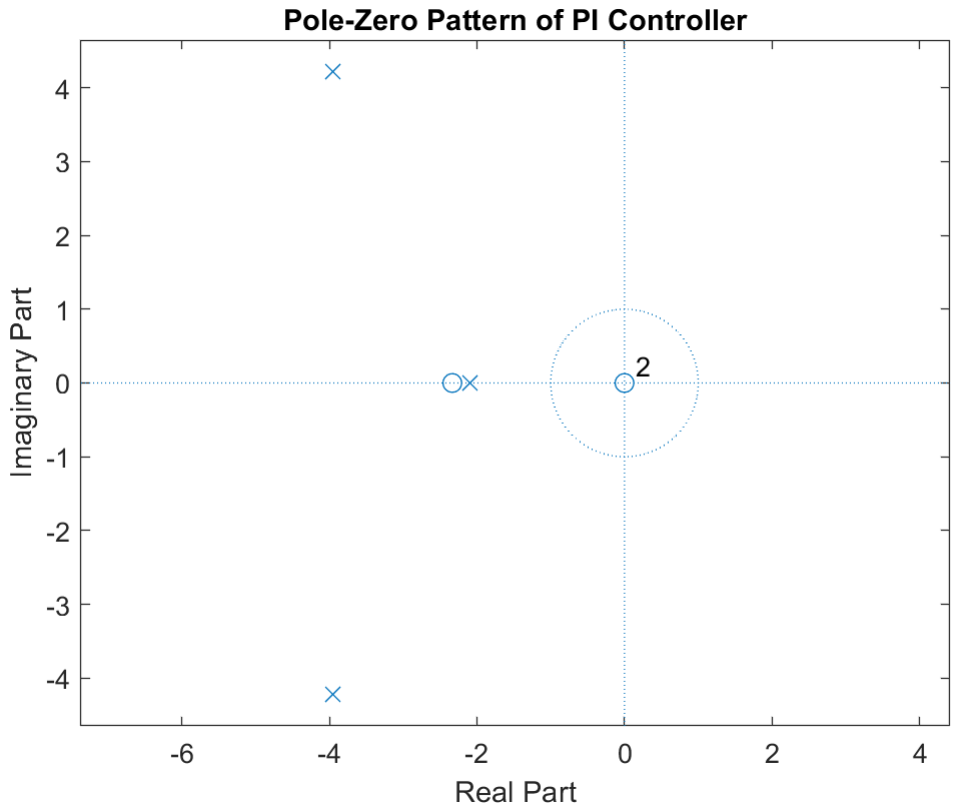
```
T = feedback(C*P,1)
```

```
T =

         30 s + 70
  ------------------------
  s^3 + 10 s^2 + 50 s + 70

Continuous-time transfer function.
```

```
t = 0:1/100:2; figure, step(T,t), title("Step Response when K_p = 30 & K_i = 70")
```




```
B = [Kp Ki]; A = [1 b (20+Kp) Ki]; figure, zplane(B,A), title('Pole-Zero Pattern of PI Controll
```



The proportional gain (K_p) has been reduced because the integral controller also reduces the rise time and increases the overshoot just as the proportional controller does as well. The resulting step response shows that the integral controller elminated the steady-state error.

Proportional-integral-Derivative Controller

Now let's combine all the previous methods into one.

The closed-loop system with a PID controller would be:

$$T(s) = \frac{Y(s)}{R(s)} = \frac{K_d s^2 + K_p s + K_i}{s^3 + (10 + K_d)s^2 + (20 + K_p)s + K_i}$$

After some experimenting, the desired response is provided when

$K_p = 350$, $K_i = 300$, and $K_d = 50$

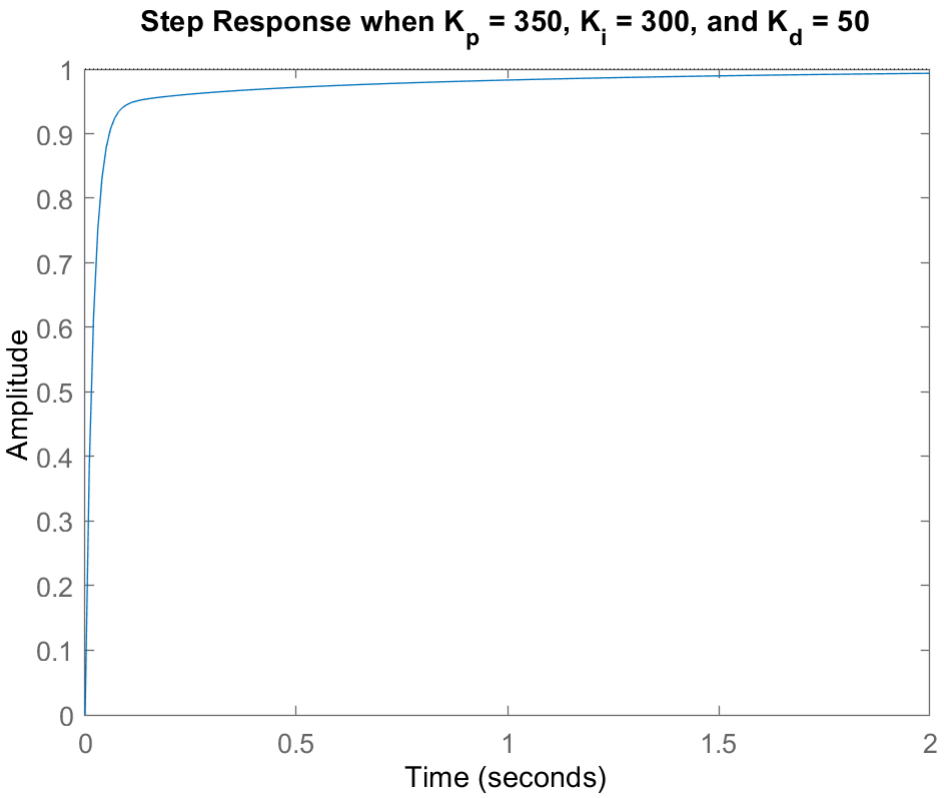
```
Kp = 350; Ki = 300; Kd = 50;
C = pid(Kp,Ki,Kd)
```

```
C =

           1
  Kp + Ki * --- + Kd * s
           s

  with Kp = 350, Ki = 300, Kd = 50

Continuous-time PID controller in parallel form.
```

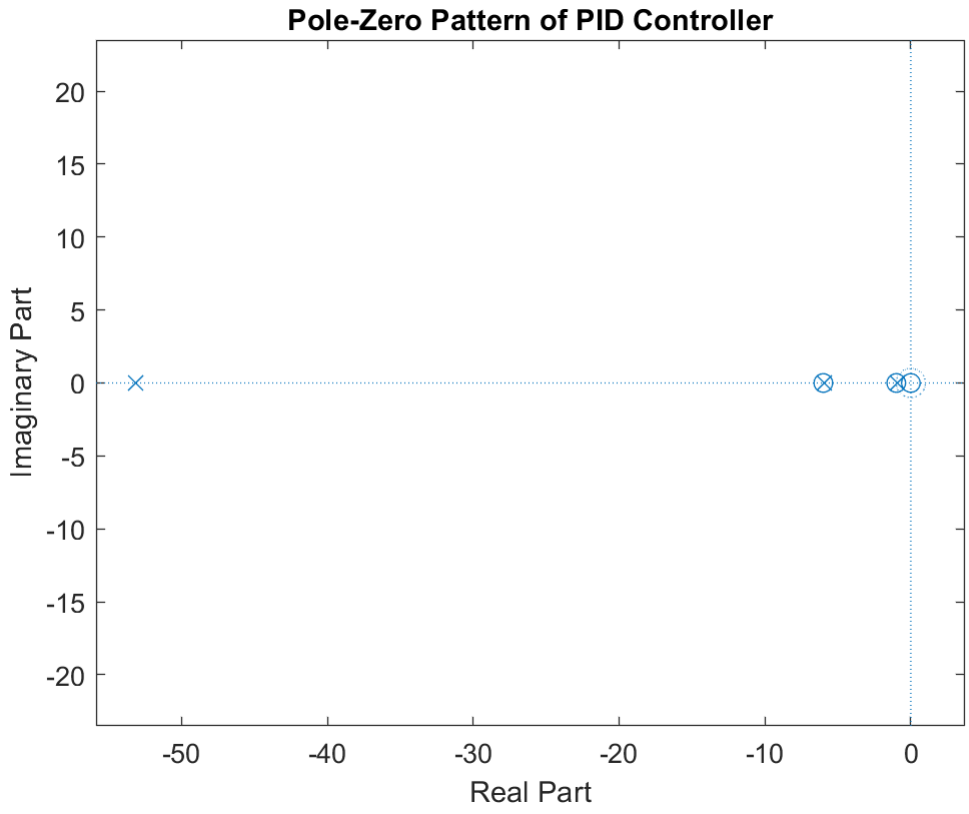
```
T = feedback(C*P,1)
```

```
T =

     50 s^2 + 350 s + 300
  --------------------------
  s^3 + 60 s^2 + 370 s + 300

Continuous-time transfer function.
```

```
t = 0:1/100:2; figure, step(T,t), title('Step Response when K_p = 350, K_i = 300, and K_d = 50
```




```
B = [Kd Kp Ki]; A = [1 10+Kd 20+Kp Ki]; figure, zplane(B,A), title('Pole-Zero Pattern of PID Co
```



General Tips for Designing a PID Controller

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

1. Obtain an open-loop response and determine improvements
2. Add proportional control to improve the rise time
3. Add derivative control to reduce the overshoot
4. Add integral control to reduce the steady-state error
5. Adjust of the terms, K_p, K_i, and K_d until desired output is achieved

Don't necessarily need to implement all three controllers into a single system