Department of Engineering
MECH304 - Dynamic Control

Project Part 2 : Designing a PD Controller

June 3, 2022

Deniz Erdogan
Baris Aslan Turan
Defne Korkmaz

Supervisor: Prof.Cagatay Basdogan

Contents

1 Introduction **2**

2 PART 1 **2**

2.1 Deriving the Transfer Function of the Plant 2
2.2 Design of the PD Controller 4
2.3 Zero at the Positive Side 5
2.4 Zero Between to the Pole Pairs 5
2.5 Zero to the Left of Poles 6
2.6 Optimizing the Parameters 6
2.6.1 Smaller Overshoot 6
2.6.2 Aggressive but Oscillatory Configuration 7
2.6.3 Final Configuration 7

3 PART 2 **8**

3.1 Designing the Controller Under Constraint 8
3.2 Building The Simulink Model 9
3.3 Investigating the Effect of K_p and K_d 11
3.3.1 Marginally Stable Case 12
3.3.2 Underdamped 12
3.3.3 Overdamped 13
3.3.4 Plots and Maps 15
3.4 Greater Disturbances 18
3.5 Conclusion 18

1 Introduction

In this project, we are proposing a design of a PD controller to stabilize a ball on a tilting grooved system. The goal is to keep the ball in the vicinity of the origin that is around the shaft of the motor that controls the system. A sample illustration of this system can be seen from Figure 1. In order to accomplish this task, firstly, the transfer function of the plant is derived using Newton's law and transferred into frequency domain by Laplace transform.

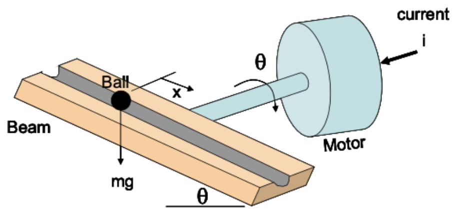


Figure 1: Illustration of the system

Following this, using the *Root Locus method*, the necessary PD parameters are derived in accordance with the design requirements that make the system not only stable but also compliant with various constraint such as using an upper bound for settling time or percent overshoot.

2 PART 1

2.1 Deriving the Transfer Function of the Plant

In this section, the derivation of the plant transfer function will be investigated. To start with, we focused on the mechanical part of the system and obtained the transfer function between X and θ. Figure 2 below illustrates our FBD. Further calculations are shown below.

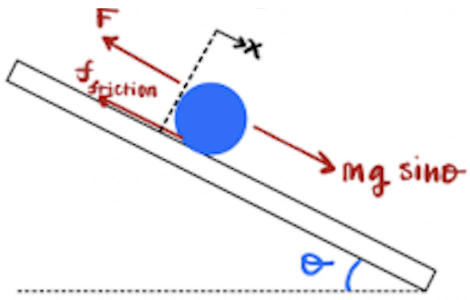


Figure 2: Calculations of the first part

$$\sum F = mg * sin(\theta) - F - f_{fric} = m\ddot{x}$$
$$sin(\theta) \approx \theta$$
$$\sum F = mg * \theta - m\ddot{x} - \mu\dot{x} - \mu_{min}x$$
Now, take the Laplace Transform:
$$m(s^2X(s)) + \mu sX(s) + \mu_{min}X(s) = mg\Theta(s)$$
$$X(s)\Big(ms^2 + \mu s + \mu_{min}\Big) = mg\Theta(s)$$
$$\frac{X(s)}{\Theta(s)} = \frac{mg}{ms^2 + \mu s + \mu_{min}}$$

Following these calculations, θ will be related to our current, I. After this step, two equations can be combined easily to obtain the transfer function between the current and X. Calculations below show the steps and derivations for these equations.

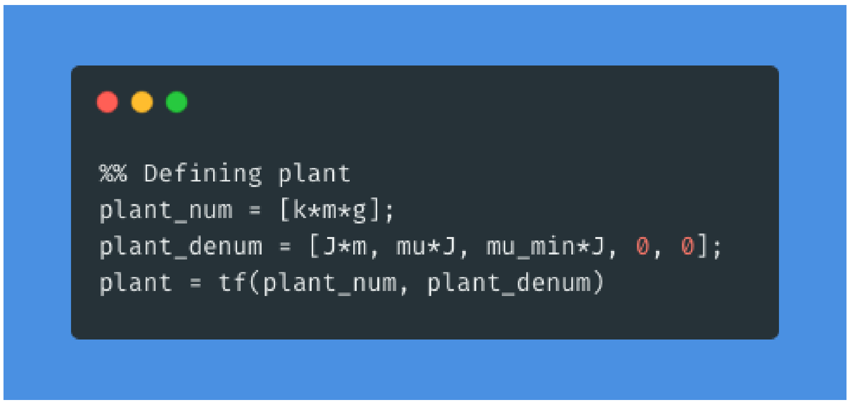
$$i(t) * k = T = J\ddot{\theta}$$
$$I(s) * k = J * \Theta(s) * s^2$$
$$\frac{\Theta(s)}{I(s)} = \frac{k}{Js^2}$$
We now combine the 2 transfer functions:
$$T(s) = \frac{X(s)}{I(s)} = \frac{mgk}{Jms^4 + J\mu s^3 + J\mu_{min}s^2}$$

Plugging in the values from the project prompt, our transfer function becomes as follows:

$$T(s) = \frac{0.0981}{s^4 + 5s^3 + 20s^2} \tag{1}$$

2.2 Design of the PD Controller

In this part, we switch to Matlab in order to design our parameters for the PD controller with the help of necessary tools. To define our transfer function there, we first initialize our parameters. Once the parameters are defined, we then create our plant by defining the numerator and denominator and passing them to tf() function. The obtained transfer function will be fed into rlocus() and sisotool() methods to be manipulated further. As can be seen from Figure 3.



```
%% Defining plant
plant_num = [k*m*g];
plant_denum = [J*m, mu*J, mu_min*J, 0, 0];
plant = tf(plant_num, plant_denum)
```

Figure 3: Creating the Plant

When we see the Root Locus plot of this plant, we get the following figure. As can be seen, this system is not stable since it has 2 roots on the imaginary axis. Our aim is to introduce another *zero* to the system in order to obtain a configuration where all roots have negative real parts.

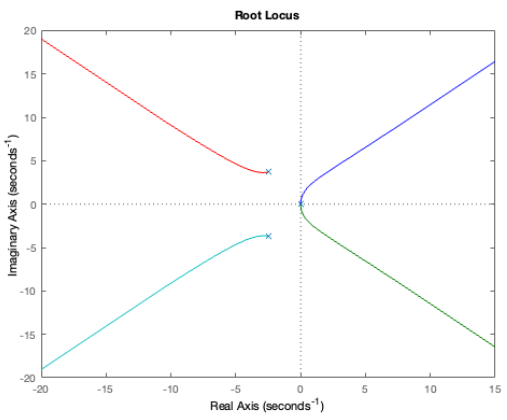


Figure 4: Root Locus Plot

Furthermore, the step response plot obtained from the sisotool() GUI confirms that the system is not stable. Figure 5 below shows this behavior.

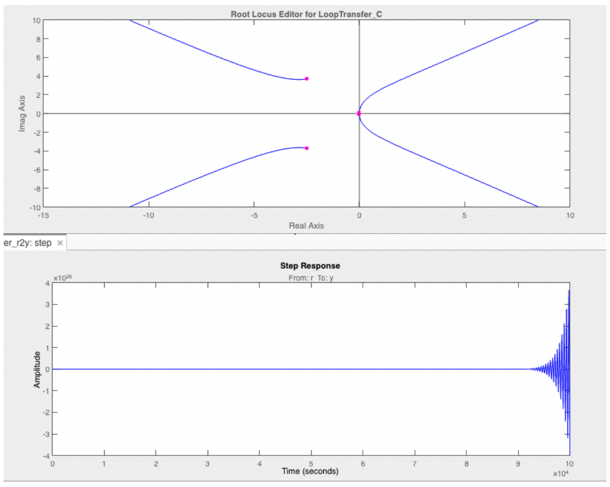


Figure 5: Sisotool Screen

The PD controller allows us to **introduce a new zero** to our system. This zero can be placed strategically in order to manipulate the *Root Locus plot* and obtain a configuration where all the root have negative real parts. There are 3 main places that our new zero can be placed. They are as follows:

- To the right of all poles
- Between the poles on the imaginary axis and the negative poles
- To the left of all poles

2.3 Zero at the Positive Side

When the zero is added to first mentioned location, this system does not benefit and continues to be unstable. A sample plot that shows this behavior is shown in Figure 6 below.

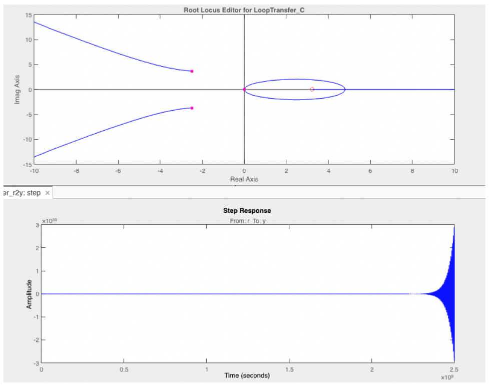


Figure 6: Zero added to the positive real part

2.4 Zero Between to the Pole Pairs

The second placement option for our zero actually allows a stable system. When the zero is placed between the two pole pairs, we obtain the option to select our K_d and K_p values to stabilize the system. A figure showing this behavior is given below on Figure 8. Although there is significant overshoot, this configuration can be stable and the design parameters can and will be further optimized throughout this report.

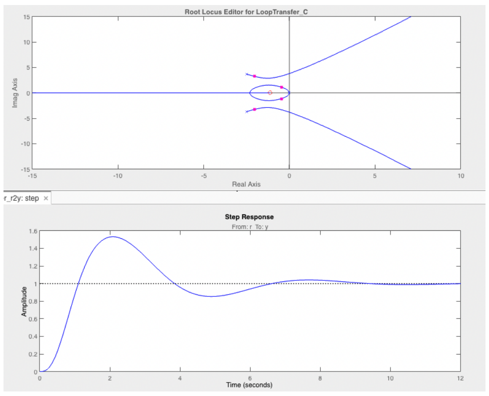


Figure 7: Zero added between the two pairs

2.5 Zero to the Left of Poles

This placement condition was first thought to not allow any stable configurations. However, it is then realized that there exists a small region to the left of the negative poles where the system in fact, can be stable. A sample configuration for the marginally stable case is shown below.

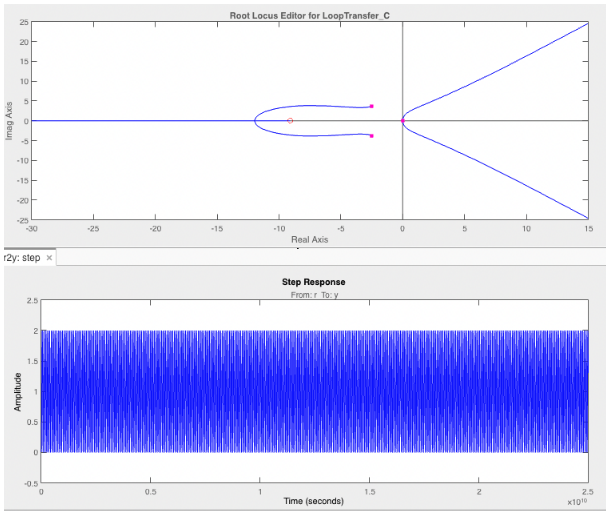


Figure 8: Zero added to the left the two pairs

2.6 Optimizing the Parameters

2.6.1 Smaller Overshoot

We continued by optimizing different design parameters individually. Our first optimization was keeping the overshoot at the minimum. This was observed when the zero value was close to the imaginary axis. This system is shown on the Figure 9 below.

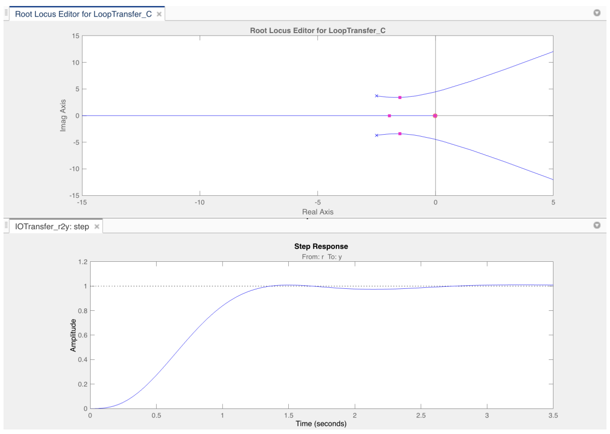


Figure 9: System with small overshoot

This system has a slower response time but it is very robust. The equation of the controller that corresponds to this system is as follows:

$$C(s) = 3.27 * (1 + 86s) \tag{2}$$

2.6.2 Aggressive but Oscillatory Configuration

We then optimized our system to have a more aggressive response. This, as expected, resulted in a oscillatory configuration that overshoots significantly more than the previous configuration. The response of this system is shown below:

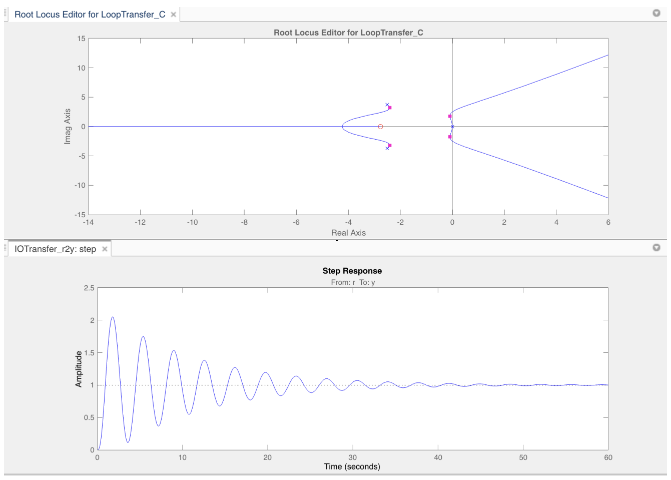


Figure 10: A more aggressive system

The controller that corresponds to this system is shown below:

$$C(s) = 502 * (1 + 36s) \tag{3}$$

2.6.3 Final Configuration

With our experience from the previous systems, we then continued with a new controller that we think is the best for our use in this project. It has very little overshoot, is fairly responsive and robust. A sample output of this configuration is shown below.

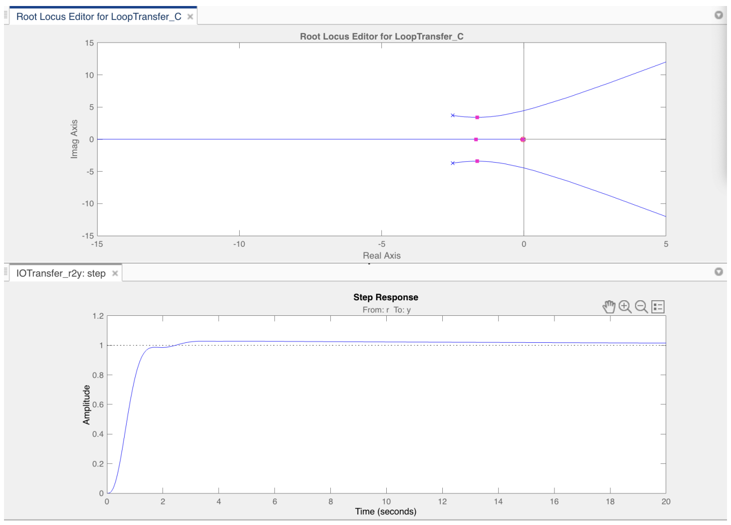


Figure 11: Proposed system

The controller that corresponds to this system is shown below:

$$C(s) = 9.75 * (1 + 26) \tag{4}$$

3 PART 2

3.1 Designing the Controller Under Constraint

In this part, we aim to find a PD controller that conforms with the design requirements needed. In order to achieve such a configuration, we used the Root Locus method and chose our K_p and K_d values accordingly. Figure 12 below shows the root locus with the vertical line showing the $T_s < 4$ threshold. I should be kept in mind that due to location of zero, we might not satisfy the condition even if all our poles are in the safe region. Therefore, we checked the settling time from the step response to be safe.

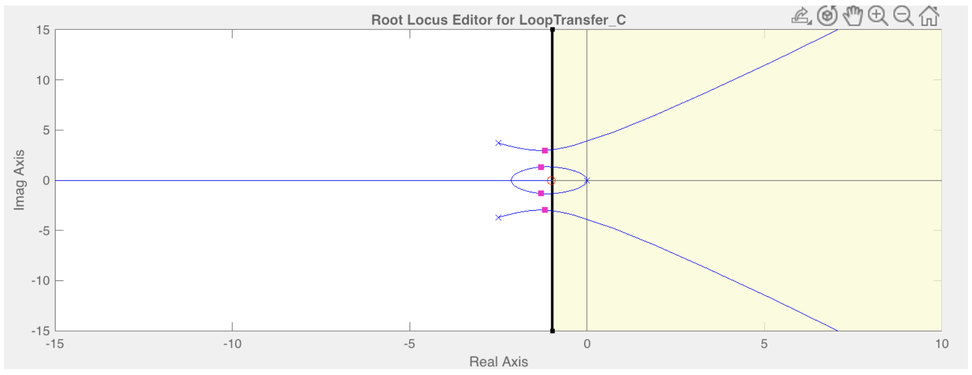


Figure 12: $T_s < 4$ seconds

Following this, we added an extra constraint on the percent overshoot metric. Since percent overshoot is a function of damping ratio, this constraint is not a vertical line, but 2 lines originating from zero in negative real axis quadrants. The snapshot of CSD showing the constraints together are shown on Figure 13.

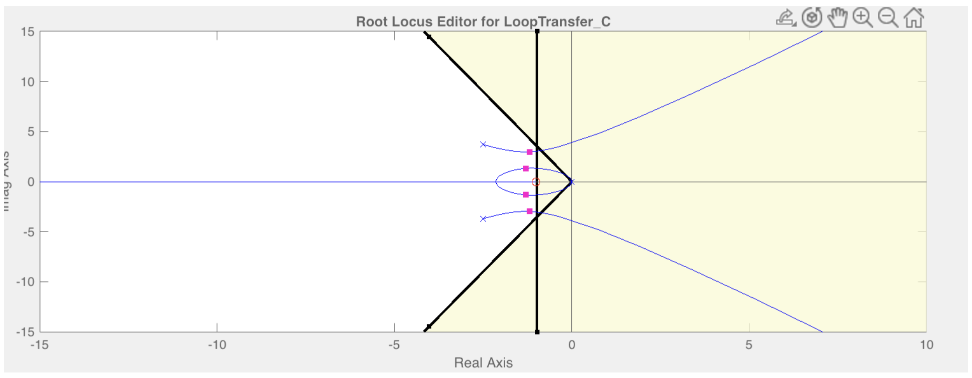


Figure 13: 2 Different Constraints

As can be seen above, our poles are in the safe zone for all conditions and the step response plot of the system verifies this condition. The step response which illustrates $T_s < 4$ and P.O ¡ 50% is shown below on Figure 14. In this case, $T_s = 3.7 < 4$ seconds.The K_p and K_d values that correspond to this controller are as follows:

- $K_p = 355.5$
- $K_d = 375.05$

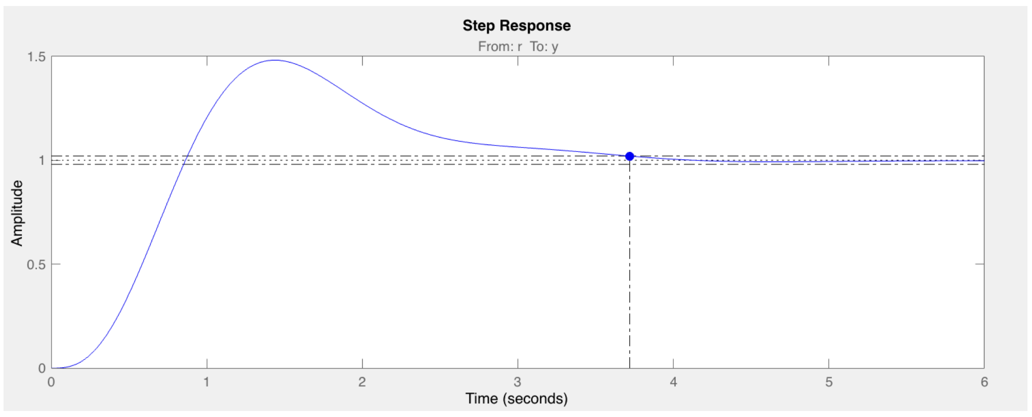


Figure 14: Step Response

3.2 Building The Simulink Model

Then, we moved on to Simulink and using block diagrams, we were able to build a model of our system that is connected to the variables on Matlab workspace, using this, we were able continuously update our parameters from Matlab side. Our block diagram model is shown below.

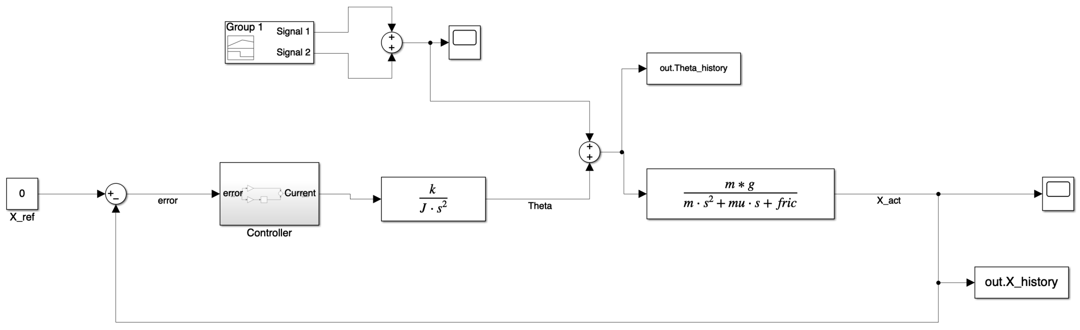


Figure 15: Simulink Model

In this model, our PD controller is created as a subsystem which comprises of the following model shown below. We manually created the controller by summing the K_p and K_d components. While K_p component is more direct, we took the derivative of the signal for K_d the part.

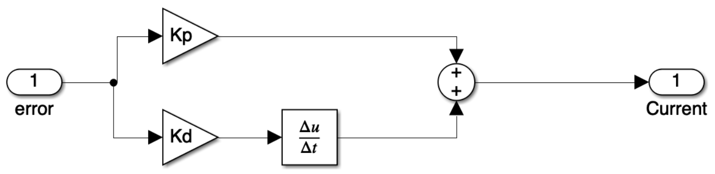


Figure 16: PD Subsystem

As Figure 15 illustrates, we have a disturbance signal on our system. This disturbance signal is built using the signal builder and summing that signal with Θ on Simulink. This allows us to test the performance of our controller better since without it the ball would be stable anyhow. The following figure illustrates our signal.

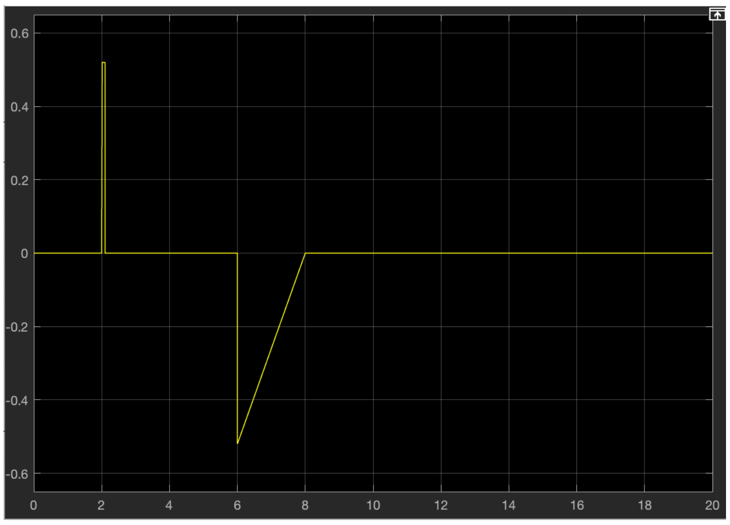
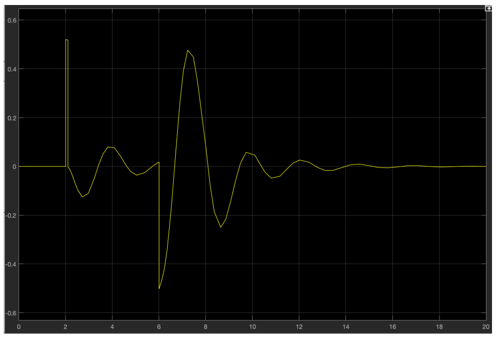
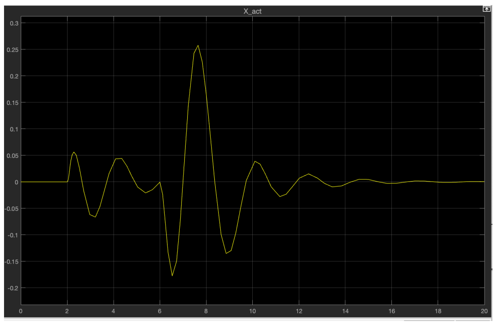


Figure 17: Disturbance Signal

Under the discussed disturbance signal, we used scopes to monitor our Θ and X to check if our system was able to stabilized once the disturbances have been made. Following figures show our Θ and X respectively.


(a) Scope of Θ



(b) Scope of X

Figure 18: Outputs

Following this, we exported these signals as timeseries datatype and fed into the animation function provided by our TA. With that visualization, we were able to more intuitively observe our system. A sample screenshot from the ending of that animation is shown below.

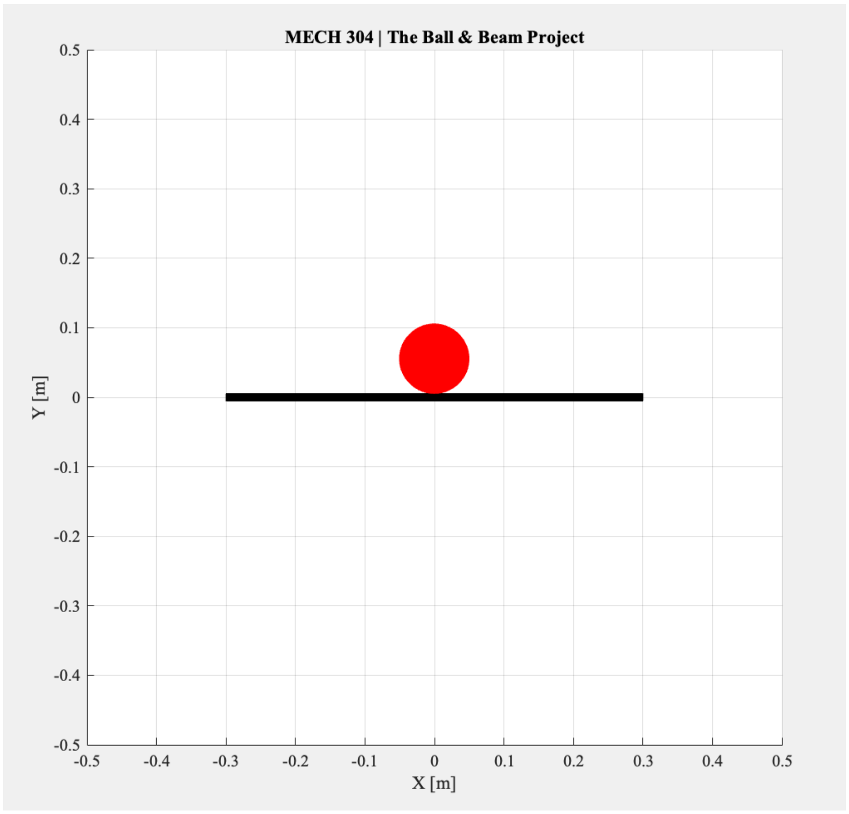


Figure 19: Stabilized Animation

3.3 Investigating the Effect of K_p and K_d

On top the controller proposed in previous sections, in this section 3 new different controllers will be that create different systems. While stabilizing these conditions, we manipulated the $/K_p$ and K_d values using the root locus to obtain a controller for the systems. Then we exported these controllers from CSD to MATLAB workspace.

3.3.1 Marginally Stable Case

In this part of our project, we obtained the marginally stable condition by making changes on the root locus graph. Due to the PD controller, we were able to introduce a zero to our system. Using this, we were able to position our roots in the negative zone. Then, we positioned the two roots in the system coincidentally on the imaginary axis. So, we get a marginally stable condition that oscillates forever, and the oscillations continue with a constant magnitude. Figure 20 below shows our Root Locus and step response for this configuration. Also, the corresponding K_p and K_d values are as follows:

- K_p = 2.4413e-18
- K_d = 9.7701e-19

As can be seen, the values are very close to zero, this is due to the fact that in order to put the poles on the imaginary axis, we chose the option to put our zero in the vicinity of the origin.

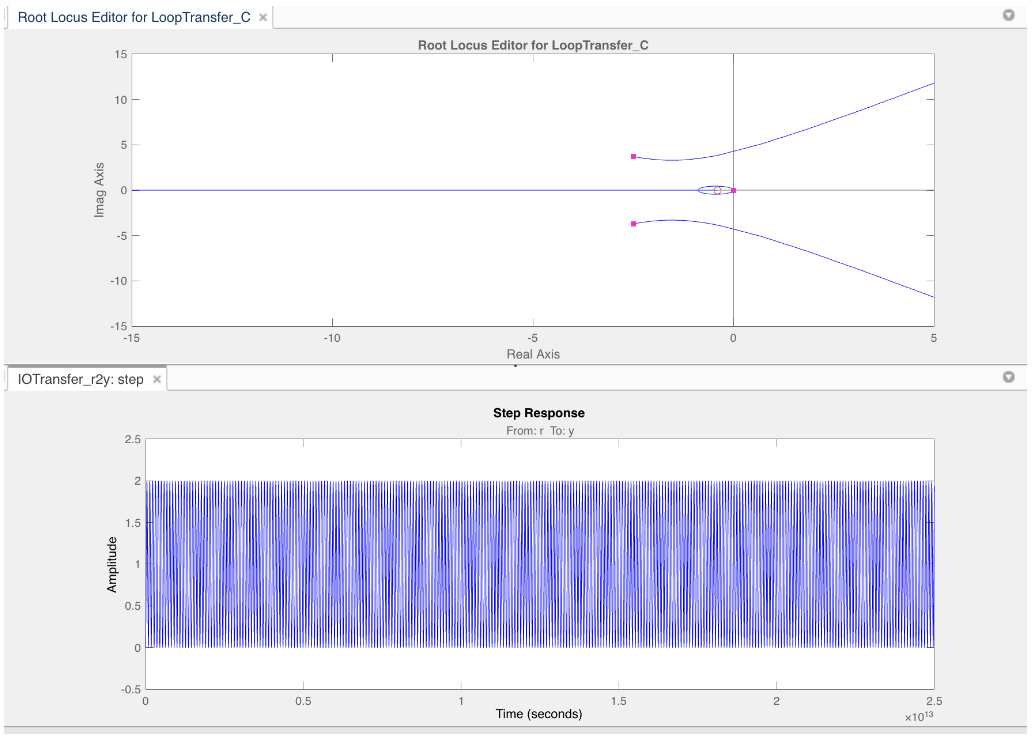


Figure 20: Marginally Stable

3.3.2 Underdamped

We positioned 4 poles on the root locus graph. Here we use 4 roots as 2 conjugate root pairs. Since the first of these conjugate root pairs is closer to the imaginary axis than the other, they affect the system as a more significant pair. This feature causes oscillations in the system. While these oscillations were observed in second order systems, we can approximate and observe the similar affect in our system. Although high oscillations are observed in this system at first, these oscillations decay over time. So, oscillations converge to the steady state value. The parameters that correspond to this system is as follows:

- K_p = 721.94
- K_d = 496.26

Unlike the overdamped case which is discussed later in this report, in this configuration, Kp value is greater than Kd and Kp is quite large. Since Kp is the "springy" term in our system, we can expect our system to show an oscillatory behavior as it increases.

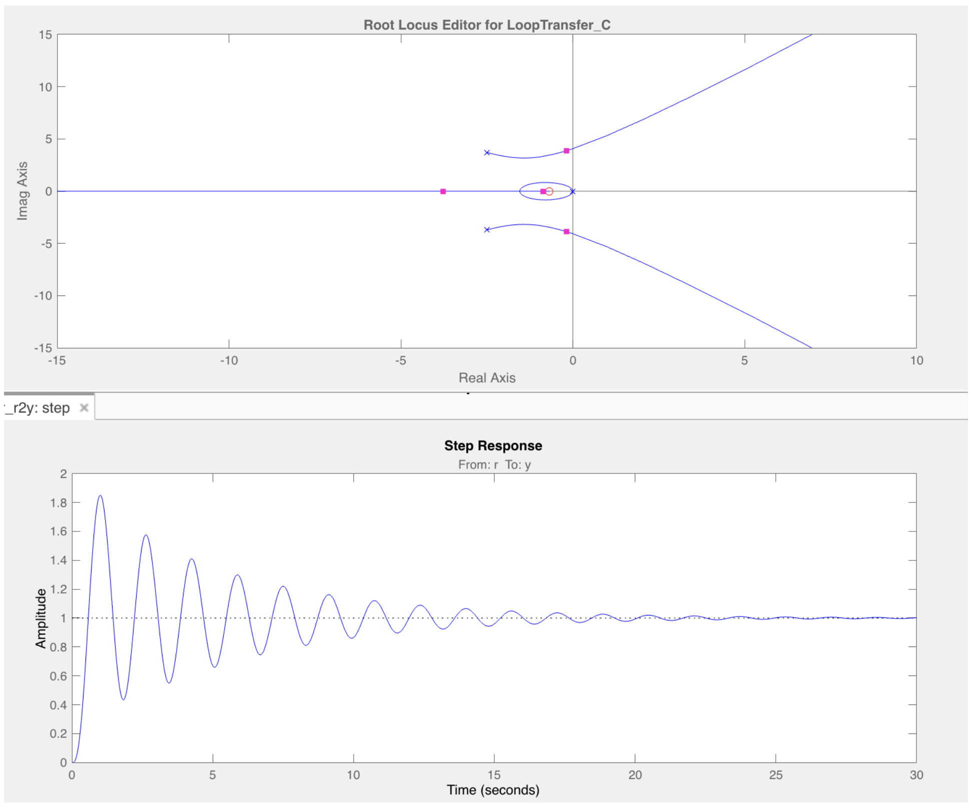


Figure 21: Underdamped System

3.3.3 Overdamped

In this part of our project, we wanted to make our system overdamped. However, overdamped characteristic could not be fully established due to the 4th degree of our system. When the graph is observed initially, the system seems to be overdamped. However, we saw that our system could not fully provide the overdamped condition even though it looks to be overdamped in step response in first glance.

Further investigation revealed that the graph went above the steady state value with a small value when zoomed. When we rate this value and evaluate it as a percentage overshoot, we get less than 2% percent overshoot value. For this reason, we consider our system to be overdamped. Root locus and step response of this system is shown on Figure 22 Controller parameters are as follows:

- $K_p = 3.8$
- $K_d = 255.5$

This high ratio between Kp and Kd is expected since Kd is the derivative term which acts as the damping term in our system. As we increase the Kd value, we can expect the system to be more damped.

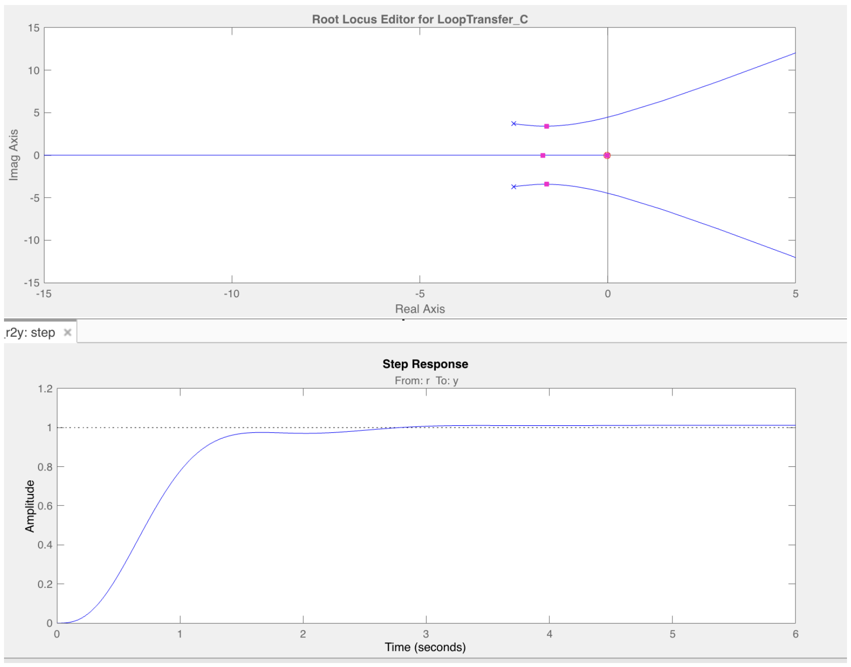
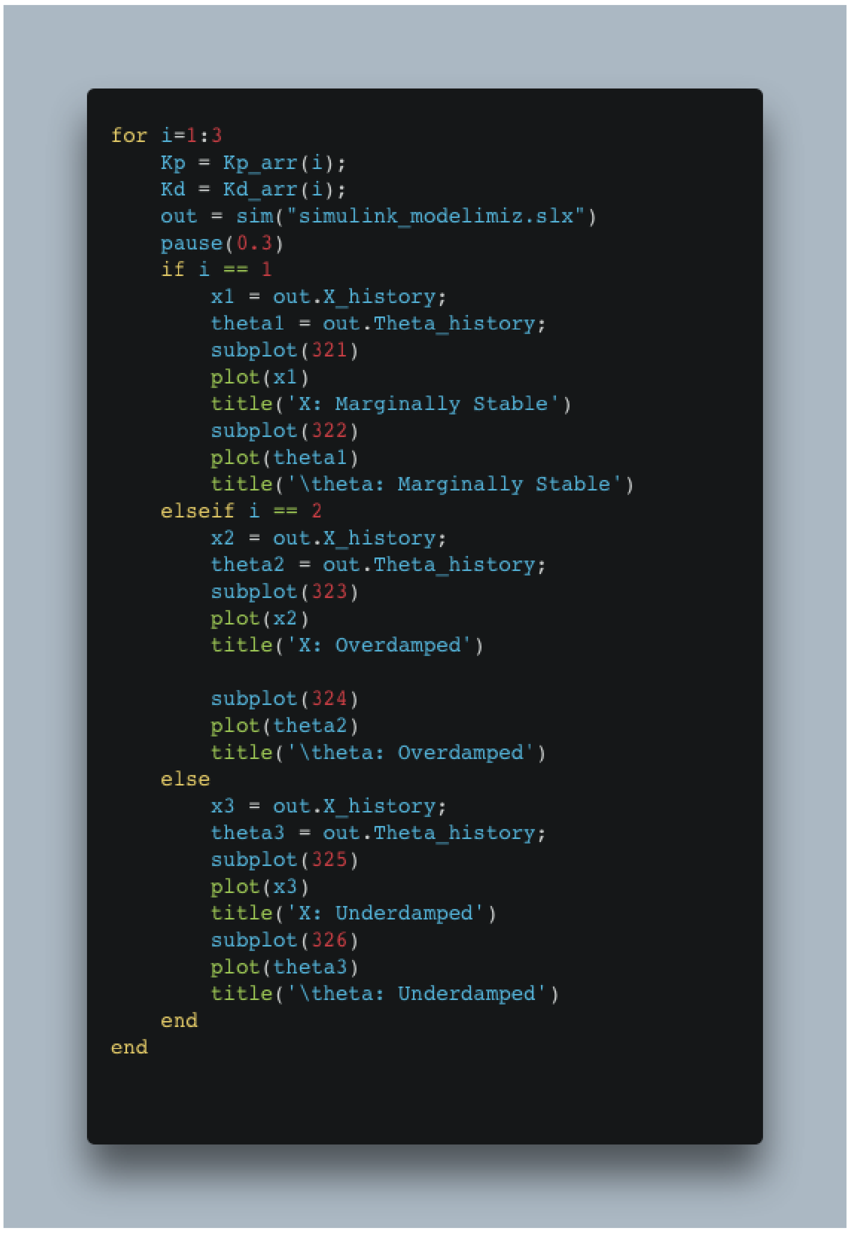


Figure 22: Overdamped

3.3.4 Plots and Maps

After this step, closed-loop pole-zero map, angular position and ball's position are plotted. To achieve this, we automated the steps by putting all Kp and Kd values into an array and running simulations from Matlab side in a for loop plotting the values as they are exported. Following figure shows how we did this.



```
for i=1:3
    Kp = Kp_arr(i);
    Kd = Kd_arr(i);
    out = sim("simulink_modelimiz.slx")
    pause(0.3)
    if i == 1
        x1 = out.X_history;
        theta1 = out.Theta_history;
        subplot(321)
        plot(x1)
        title('X: Marginally Stable')
        subplot(322)
        plot(theta1)
        title('\theta: Marginally Stable')
    elseif i == 2
        x2 = out.X_history;
        theta2 = out.Theta_history;
        subplot(323)
        plot(x2)
        title('X: Overdamped')

        subplot(324)
        plot(theta2)
        title('\theta: Overdamped')
    else
        x3 = out.X_history;
        theta3 = out.Theta_history;
        subplot(325)
        plot(x3)
        title('X: Underdamped')
        subplot(326)
        plot(theta3)
        title('\theta: Underdamped')
    end
end
```

Figure 23: Running Simulink From Matlab

Here are the said plots:

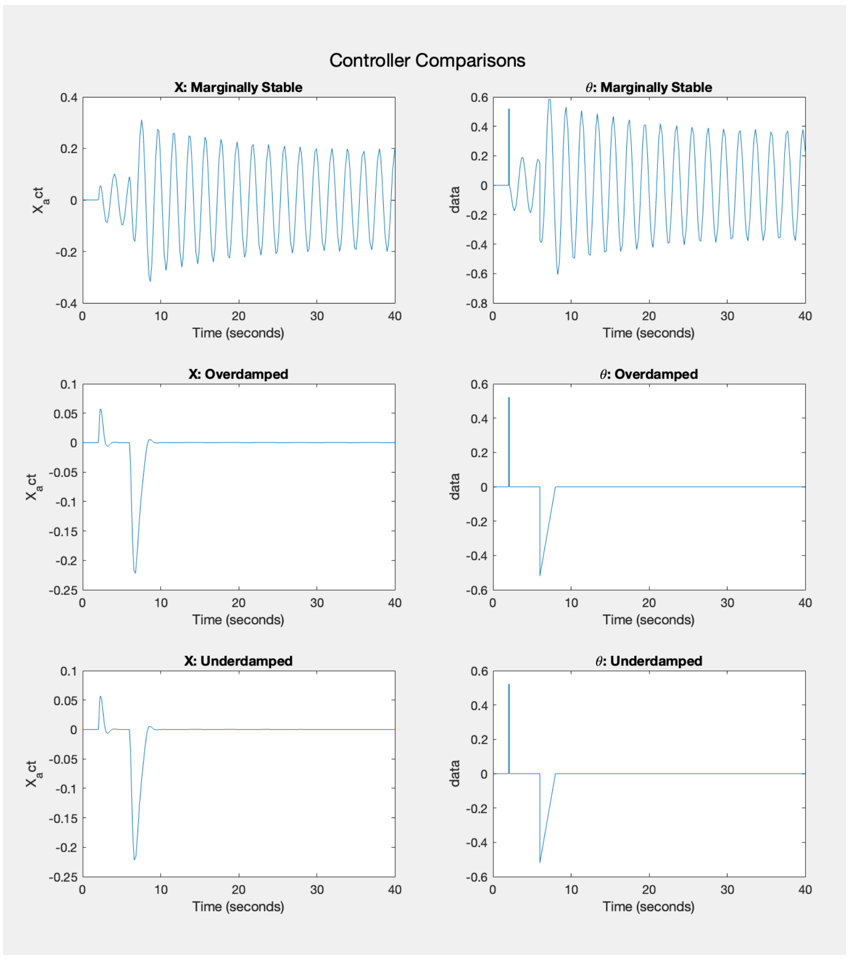


Figure 24: Responses for 3 different cases.

In the pole zero map figure, the subplots show the underdamped, overdamped and marginally stable case respectively.

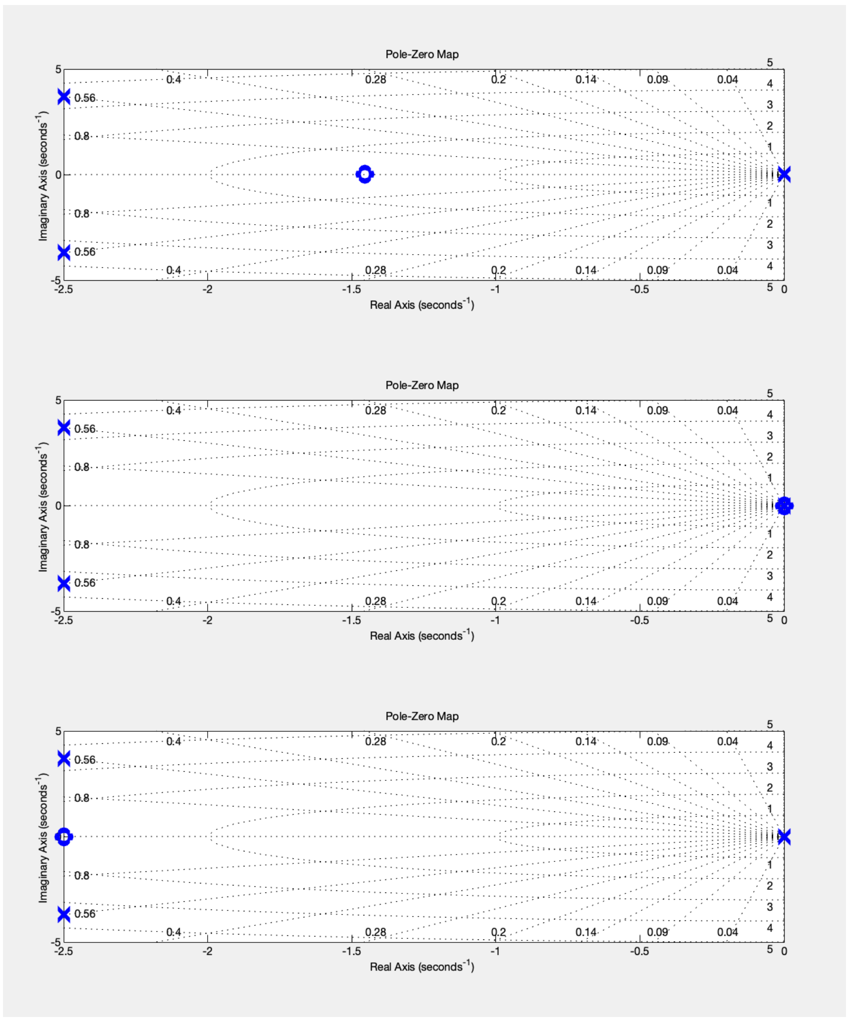


Figure 25: PZ Maps

Also, we plotted the pole zero maps using pzplot() command inside a for loop iterating through our Kd and Kp arrays. The piece of code that does this part is shown below.

```
for i=1:3
     subplot(3,1,i);
     Kp = Kp_arr(i);
     Kd = Kd_arr(i);
     numG = [Kd*m*g*k, Kp*m*g*k];
     denG = [J*m, J*mu, J*fric , 0, 0];
     G = tf(numG, denG)

     plotoptions = pzoptions;
     plotoptions.Grid = 'on';
     pzplot(G, plotoptions , "b")
     hm = findobj(gca, 'Type', 'Line');
     hm(2).MarkerSize = 10;
     hm(3).MarkerSize = 10;
     hm(2).LineWidth = 4;
     hm(3).LineWidth = 4;
end
```

3.4 Greater Disturbances

We then wanted to test our system even more and introduced extra disturbances to our system. To do this, we added 2 unit step inputs to our system. One of these is added to our current, and the other is added to our output X. The magnitudes of the unit steps are 0.5 and they get activated after the 8 second mark to not interfere with the previous signal. Our model with new disturbances added is as follows.

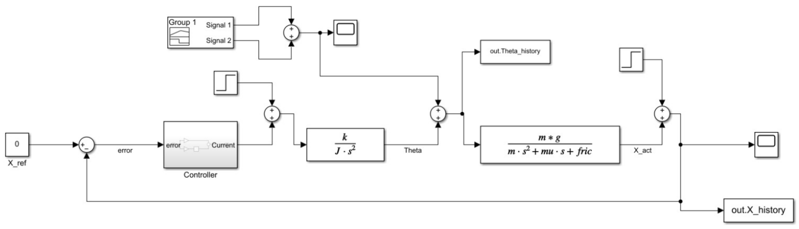


Figure 26: Simulink Model w/ Extra Signals

Since we are adding another unit input to out output X, we expect the steady state value of out Θ output to be different than 0, which is -1 in our case. This effect is similar to applying a force on the ball. Due to this, the paddle on which the ball stabilizes is not level after the system stabilizes. Figure 27 shows the output responses of Θ and X.

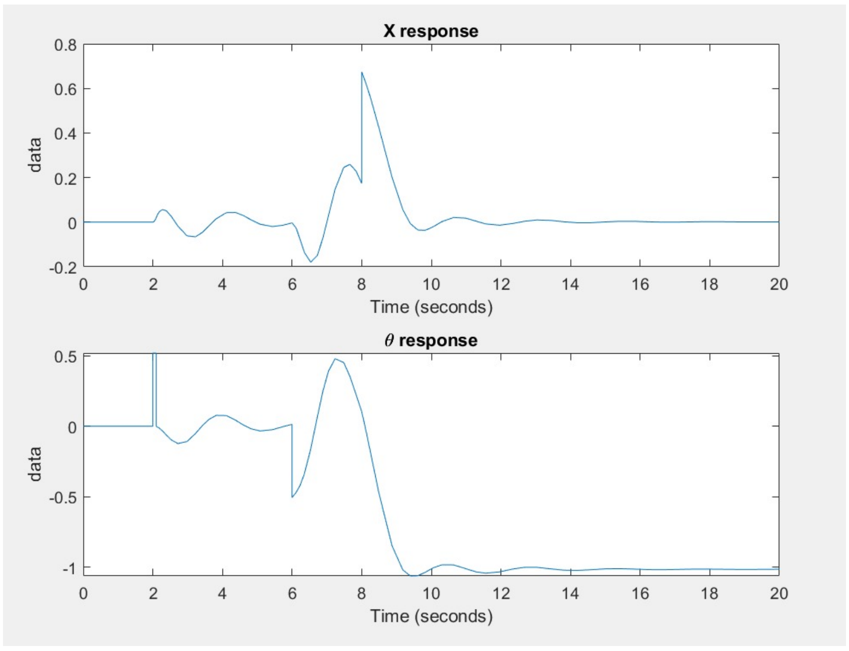


Figure 27: Outputs w/ extra disturbances

3.5 Conclusion

In this project we gained a hands on experience with utilizing the Root Locus, using CSD module and Simulink and designing controllers. We investigated the effects of controller parameters and created different configurations. We also testes our controllers under disturbances and saw them rejecting the signals and stabilizing quickly. Furthermore, we designed some of these controllers under various constraints to conform with design parameters.