Electrical and Computer Engineering

School of Engineering

Wentworth Institute of Technology

Feedback and Control – ELEC4475

Summer 2024

Lab Experiment Name: Feedback and Control Final Project

Team Member 1 Name: Dylan Cadigan

Team Member 2 Name: Robert Nguyen

Date of Lab Experiment Demo: 8/8/2024

Table of Contents

Summary of the Design Problem

The objective of this design project is to use the root locus method in order to design a controller for a Qube 3 servo motor. The controller must have zero steady-state error, keep the percent overshoot under 10%, and make sure the settling time is under 0.6 seconds. Additionally, the DC motor must rotate at an angular velocity of 15 rad/s.

In order to achieve these requirements, one or more controllers may be implemented. This includes PI, PD, PID, lead compensators, and/or lag compensators. The choice for any of these controllers is up to the discretion of the designer.

Design Solution

Step 1: Picking a Controller

To decide which controller(s) should be implemented, the original 2nd-order voltage-to-velocity transfer function for the Qube motor from Lab 4 must be analyzed.

$$G_2(s) = \frac{14273}{(s+9.67)(s+50)} = \frac{\mathbf{14273}}{\boldsymbol{s^2 + 59.67s + 483.5}}$$

Because there is some flexibility with what can be selected for each restraint, we decided to target a 5% overshoot and a T_s of 0.1 seconds. This leads us to the following calculations:

$$\zeta = \frac{-\ln\left(\frac{\%OS}{100}\right)}{\sqrt{\pi^2 + ln\left(\frac{\%OS}{100}\right)}} \rightarrow \frac{-\ln\left(\frac{5}{100}\right)}{\sqrt{\pi^2 + ln\left(\frac{5}{100}\right)}} \rightarrow \zeta = 0.69$$

$$T_s = \frac{4}{\omega_n \cdot \zeta} \rightarrow \omega_n = \frac{4}{T_s \cdot \zeta} \rightarrow \omega_n = \frac{4}{0.1 \cdot 0.69} \rightarrow \omega_n = 57.97\, rad/s$$

Using these two parameters, we can find our desired dominate pole:

$$\sigma_d = -\zeta \cdot \omega_n \rightarrow \sigma_d = -0.69 \cdot 57.97 \rightarrow \sigma_d = -40$$

$$\omega_d = \omega_n \cdot \sqrt{1-\zeta^2} \rightarrow \omega_d = 57.97 \cdot \sqrt{1-0.69^2} \rightarrow \omega_d = 42$$

$$Doninate\ Pole = \sigma_d \pm \omega_d \rightarrow -40 \pm 42$$

As for the steady-state error of the system, we can take a look at the steady-state error formula table seen below in Figure 1 to decide what controller to implement. According to the chart, if our system becomes a type 1, when it is supplied with a step input, we will obtain 0 error.

		Type 0		Type 1		Type 2	
Input	**Steady-state error formula**	**Static error constant**	**Error**	**Static error constant**	**Error**	**Static error constant**	**Error**
Step, $u(t)$	$\frac{1}{1+K_p}$	K_p = Constant	$\frac{1}{1+K_p}$	$K_p = \infty$	0	$K_p = \infty$	0
Ramp, $tu(t)$	$\frac{1}{K_v}$	$K_v = 0$	∞	K_v = Constant	$\frac{1}{K_v}$	$K_v = \infty$	0
Parabola, $\frac{1}{2}t^2u(t)$	$\frac{1}{K_a}$	$K_a = 0$	∞	$K_a = 0$	∞	K_a = Constant	$\frac{1}{K_a}$

Figure 1: The steady-state error formula table.

Our current system is a type 0, so in order to increase the order of the system, our controller must have a pole at 0. Also considering the fact that we already have two poles that need two respective zeros to cancel them out, we decided to implement a **PID controller**. It has the form:

$$G_{PID\ Controller} = \frac{K(s+a)(s+b)}{s}$$

Combining this controller transfer function to the 2nd-order voltage-to-velocity transfer function seen above, we see:

$$G_{open}\frac{14273 \cdot K(s+a)(s+b)}{s(s+9.67)(s+50)}$$

Step 2: Solving for the Controller Variables

To solve for the controller variables, we need to draw a root locus diagram that contains all poles, zeros, and dominate poles. Lines are then drawn from the poles and zeros to one of the dominate poles. In this case, we chose the -40 + j42 dominate pole. Note: the drawing is not to scale, and the integrator has been omitted from this drawing for simplicity.

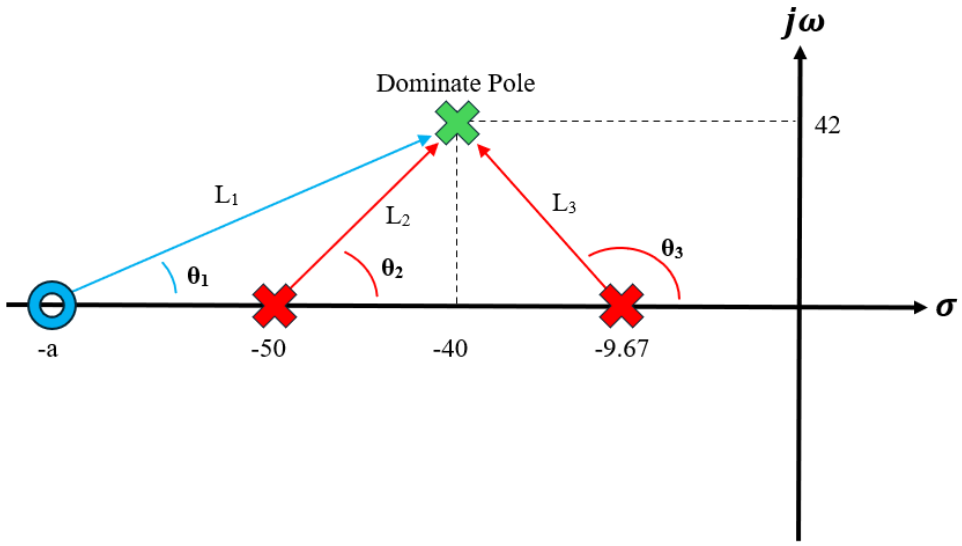

Figure 2: The root locus drawing of the transfer function.

Now we can begin solving for the controller variables. First, we will solve for all of the angles:

$$\Sigma\theta_z - \Sigma\theta_p = -180$$

$$\theta_1 - (\theta_2 + \theta_3) = -180$$

Where, $\theta_2 = \tan^{-1}\left(\frac{42}{10}\right) \rightarrow \theta_2 = 76.60°$ and $\theta_3 = 180 - \tan^{-1}\left(\frac{42}{30.33}\right) \rightarrow \theta_3 = 125.83°$

$$\text{So, } \theta_1 - (76.6 + 125.83) = -180 \rightarrow \theta_1 = 22.43°$$

The location of the non-zero zero can now be found using the previously found angles:

$$\theta_1 = \tan^{-1}\left(\frac{42}{a-40}\right) = 22.43 \rightarrow \tan(22.43) = \left(\frac{42}{a-40}\right) \rightarrow a = 141.75$$

Using some simple trigonometry, we can find the lengths of each of the lines:

$$L_1 = \sqrt{95.06^2 + 42^2} \rightarrow L_1 = 103.92$$

$$L_2 = \sqrt{10^2 + 42^2} \rightarrow L_2 = 43.17$$

$$L_3 = \sqrt{30.33^2 + 42^2} \rightarrow L_3 = 51.81$$

Using the magnitude property of root locus, the overall gain of the controller can be found:

$$K_c = \frac{\Pi\, pole\ lengths}{\Pi \text{ zero lengths}}$$

$$K_c = \frac{(43.17)(51.81)}{103.92} \rightarrow K_c = 21.5$$

Because we know the gain of the controller is equal to 14273 multiplied by the gain of the PD controller, K, the overall controller gain can be calculated:

$$14273K = 21.2 \rightarrow K = 0.0015$$

The PD controller equation becomes:

$$G_{PD} = 0.0015(s + 145.06)$$

All that's left is to add the PI controller. Initially we had the PI controller's zero located at 0.01 (close to the origin). After some testing, we noticed that our desired parameters weren't exactly what we were expecting. To fix this issue, we simply moved the zero closer to the -9.67 pole. The location of -7.85 was found to give the optimal results (explained in the Simulation section of the report). This means our final controller equation becomes:

$$G_{PID\ Controller} = \frac{0.0015(s + 145.06)(s + 7.85)}{s}$$

This controller equation will be placed in cascade with the voltage-to-velocity transfer function.

Step 3: Finding the P, I, and D Values

To implement this controller in MATLAB/Simulink, we'll need to find the separate values for the P, I, and D parts of the controller. To do this we just multiplied the numerator out and then cancelled the s term in the denominator.

$$\frac{0.0015(s + 145.06)(s + 7.85)}{s} \rightarrow \frac{0.0015s^2 + 0.299s + 1.7}{s} \rightarrow 0.0015s + 0.299 + \frac{1.7}{s}$$

This gives us the values of:

$$K_P = 0.299$$

$$K_I = 1.7$$

$$K_D = 0.0015$$

This concludes the calculations for the design solution.

Simulation

In order to validate the design, simulation was used. This was all done inside of MATLAB and Simulink. On top of simulation, MATLAB is also used to double check the calculations. The first step was to define our given.

```
%Set a variable for OS%
OS = 5;

%OL poles from our tf
p1 = -9.67;
p2 = -50;

%Set the Settling Time
Ts = 0.1;
```

Figure 3: Given Variables.

The next step is to determine the dominant pole. This can be done by finding the damping ratio and natural frequency.

```
%Set the Settling Time
Ts = 0.1;

%Calculate damping ratio for OS%
DR = round(-log(OS/100)/sqrt(pi^2 + (log(OS/100)^2)), 3);

%Calculate the natural frequency from damping ratio and chosen settling time
wn = 4/(Ts*DR);

%Calculate the desired real part of the dominant pole
sd = -DR * wn;

%Calculate the desire imaginary part of the dominant pole
wd = wn*sqrt(1-DR^2);
```

Figure 4: Dominant Pole calculation.

Using the dominant pole, we can determine the position of where to add the zero for the derivative.

```
%Find the angle between the dominant pole and the first OL pole from the real axis
t1 = mod(atand(wd/(sd-p1)),180);

%Find the angle between the dominant pole and the second OL pole from the real axis
t2 = mod(atand(wd/(sd-p2)),180);

%Calculate the position of the added zero from the derivative of the PID controller
a = wd/(mod(tand((t1+t2)-180),180))-sd;
```

Figure 5: Zero Position for Derivative.

Next is to choose the position of the zero for the integral controller, then combining the original transfer function with the derivative and integral controller.

```
%Choose the added zero from the integral of the PID controller
b = 7.85;

%Set 's' to automatically for transfer functions
s = tf('s');

%Our original transfer function
G = 14273/((s+9.67)*(s+50));

%Combine the original transfer function with the derivative
Ga = G*(s+a);

%Add the integral into the transfer function
G1 = Ga*(s+b)/s;
```

Figure 6: Zero for Integral and Controller combination.

Next is to calculate the gain for the proportional part of the controller and combine it with the rest of the system.

```
%Calculate the gain of the controller
L1 = sqrt((a+sd)^2 + wd^2);
L2 = sqrt((p2-sd)^2 + wd^2);
L3 = sqrt((p1-sd)^2 + wd^2);
k = (L2*L3)/L1;

%Calculate the total gain
k1 = k/14273;

%Add the gain to the system
G2 = G1*k1;
```

Figure 7: Gain calculation and Combination.

Next is to finally plot the system. This can be done using the *rlocus* command and the step command. *Sgrid* is used to plot the damping ratio, and the feedback function is multiplied by 15 to get 15 rads/second.

```
%Plot the rlocus and damping ratio
rlocus(G2);
sgrid (.69,0);

%Plot the step response
figure(2); step(15*feedback(G2,1))
```

Figure 8: Plotting the Root Locus and Step Response.

In the previous step, the b value, which dictates the integration, was set to 7.85. This value was found after some trial and error, first starting at 0.01, which can be seen below. This does not meet the requirements as it is under 15 and does not settle to 15 within the 0.6 settling time requirement.

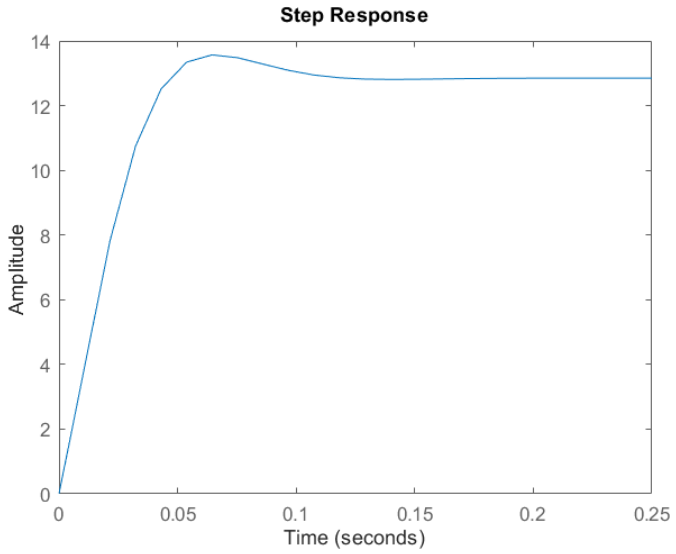

Figure 9: Step Response with 0.01 b value.

Eventually landing on 7.85 which provides the following step response and root locus.

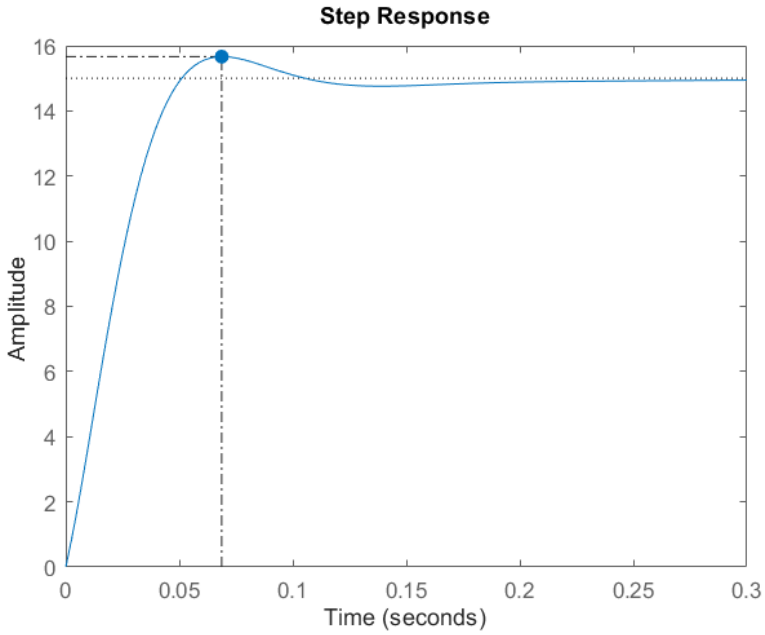

Figure 10: Step Response with 7.85 b value.

The step response is under 10% overshoot which can be seen as 16.5 is roughly 10% more than 15. The system settles before 0.6 seconds and the steady state is 15 rad/sec. This means that this value of b meets all the requirements of the design project.

The root locus of the system looks good, with the damping ratio line crossing through it.

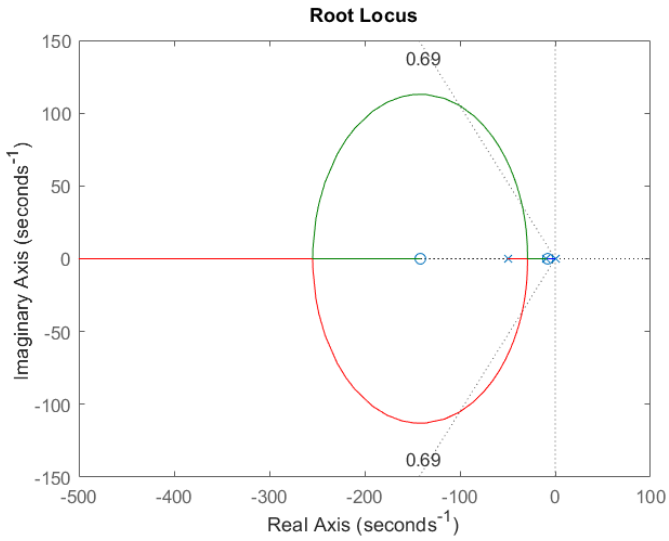

Figure 11: Root Locus of the Final System.

After testing everything in MATLAB, everything must be simulated in Simulink to make sure that it will work properly with the motor. To do this, the lab 4 schematic is used and modified to have a side branch in order to simulate without affecting the motor. The final schematic would look like this:

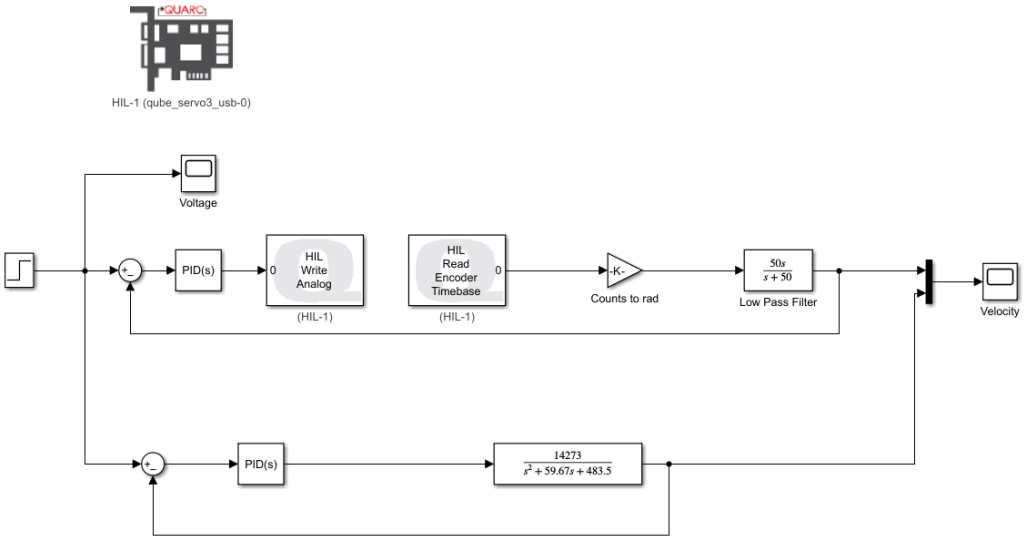

Figure 12: The final Simulink model.

The PID transfer function is converted to a controller equation and entered into the PID controller block. The filter coefficient can be adjusted however 100 worked well in this application.

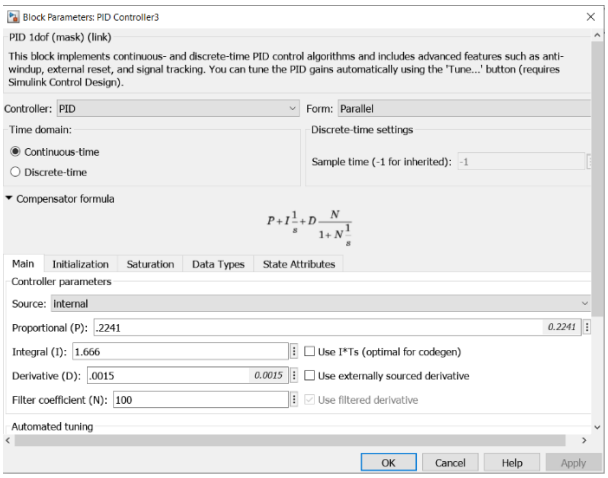

Figure 13: PID controller settings.

Next is to disconnect the motor and test the simulation and get the graph from Simulink. This can be seen below.

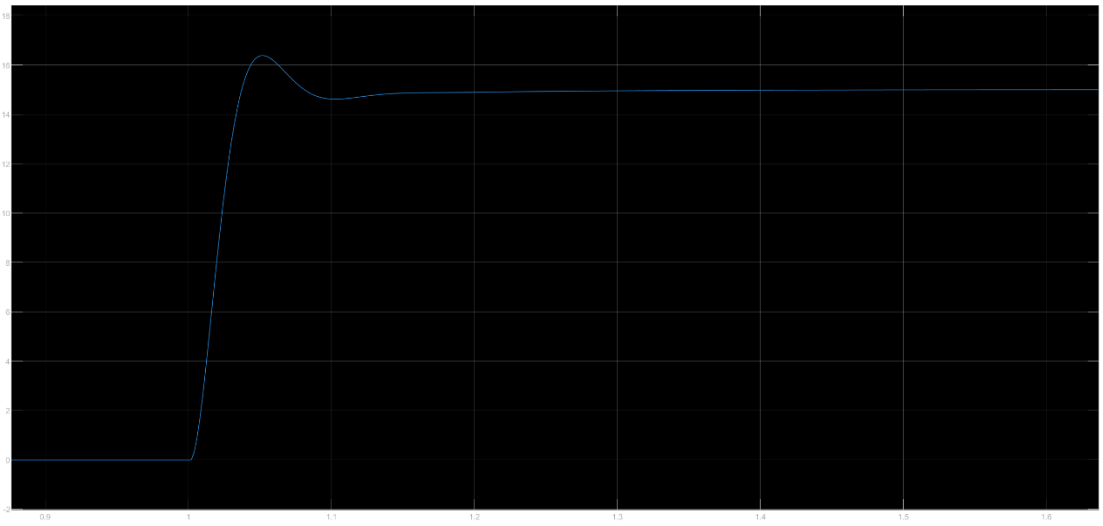

Figure 14: The Simulink results.

The Simulink results are a bit different from the MATLAB results. This can be from many things, but it seems to stem from the different solver type. Using variable step seems to get it close to our MATLAB but fixed step seems to change the results. Also, the PID controller block can act differently than the transfer function. The results are still within the design specifications, having an overshoot of under 10%, zero steady state error at 15 rad/sec, and a settling time of less than 0.6 seconds.

Experimental Results

To get the final results we connect the motor back up the system and run the motor. The motor spins for a short period of time before slowly coming to a stop. The results are plotted, as seen below. Note: The yellow line is the motor results while the blue line is the simulated, ideal plot.

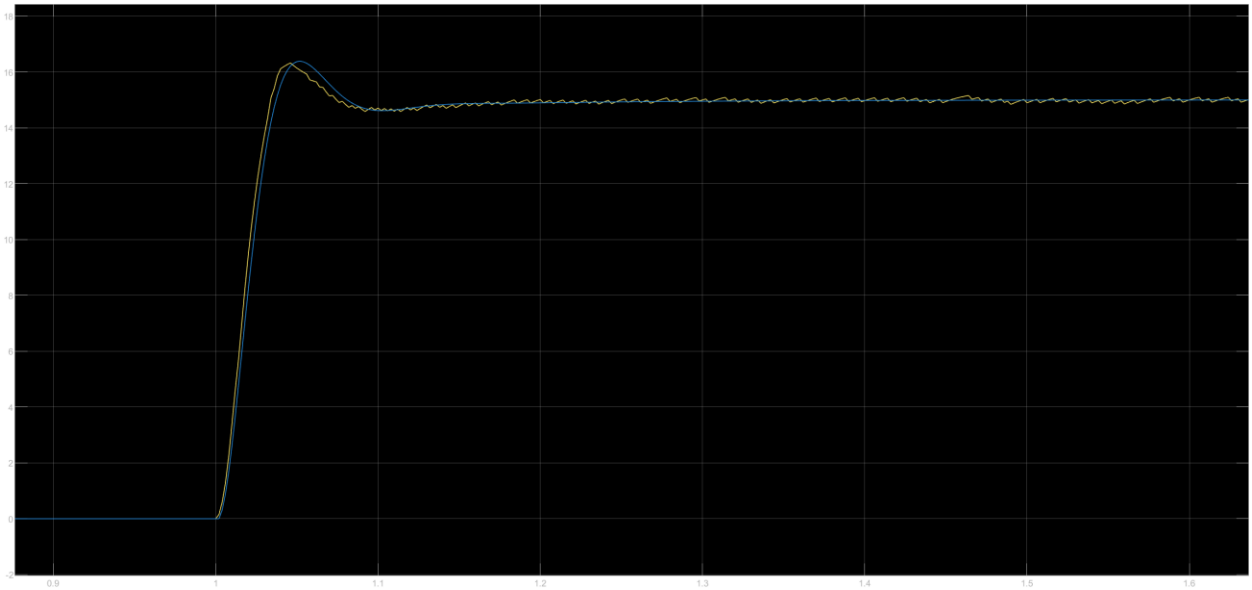

Figure 15: Motor vs Simulation Graph

The motor seems to resemble the simulation fairly accurately except it is responding a tad faster. The filter also isn't smoothing the final result as good, this could just mean there is more noise than expected. Overall, we think the results the motor gave us are within the margin of error from the simulated plot.

Conclusion

The results of the design project are a success! The motor reached 15 rad/sec with zero steady state error. It did this while being under 10% overshoot and having a settling time of less than 0.6 seconds. There were some differences between the simulation and the actual motor response, however everything still met the design requirements.

The concepts and calculations discussed in this report may seem simple in theory, but they took a lot of trial and error to perfect. We attempted to obtain the perfect values for the controller over six different times before ultimately getting these results. None of the work was in vain, though. We learned valuable lessons about designing controllers for an already existing system as well as many skills that may help us in our future careers.

Overall, this project taught us how to design a root locus to get our expected outcome and take that root locus and turn it into a controller for our desired output.

Appendix A

To be as open and transparent as possible, we created a GitHub folder containing all of our MATLAB, Simulink, and graphical results. You can access the folder using the link below. If you have any questions or concerns, please don't hesitate to reach out to us!

GitHub Folder