MECH 467

Prelab #2

By Bobsy Narayan

Date: November 5, 2024

Contents

Table of Figures

Table of Equations

System Analyzed:

In this lab, we will be creating control algorithms for the ball-screw driven table identified in Project 1. Our analyzed system model can be seen in the following figure.

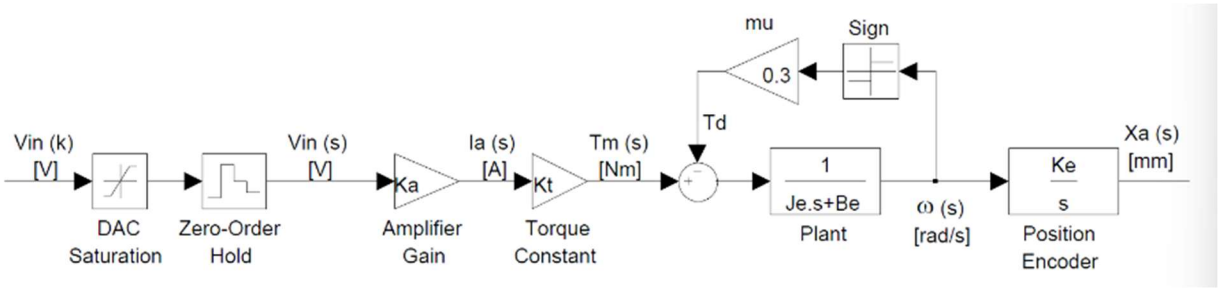

Figure 1: Open Loop Block Diagram of Ball Screw Feed Drive System

Due to the small impact and non-linear effects of torque friction, we can ignore it for our analysis, making this a linear singular path transfer function.

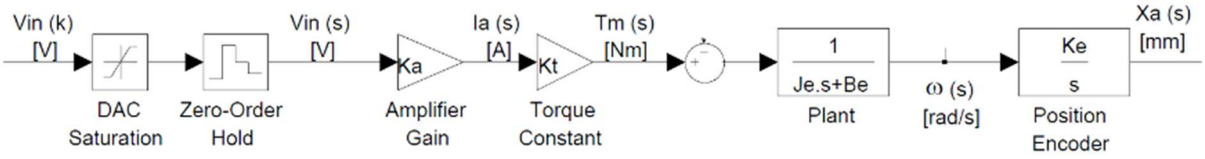

Figure 2: Simplified System Open Loop Block Diagram

In our *Lab 2 instructions document,* we are given values for system constants, as shown in the following table.

Table 1: Defined System Constants

System Constant	Value
Amplifier Gain (Ka)	0.887 A/V
Torque Constant (Kt)	0.72 Nm/A
Position Encoder Constant (Ke)	$20/2\pi$ mm/rad
Inertia Value (Je)	$7x10^{-4}\ kgm^2$
Vicsous Friction Constant (Be)	0.00612 Nm/rad/s
Sampling Time (Ts)	0.0002s

Prelab 1 – Discrete Transfer Function Derivation

1) Manually obtain Zero-Order hold equivalent of open loop transfer function for system analyzed in Lab 1. Compare to Matlab's C2D command/simulated results.

Open loop transfer function can be determined using Black's Formula, as seen in the following equation.

Equation 1: Open Loop Transfer Function for Experimental System

$$G(s) = \frac{X_a(s)}{V_{in}(s)} = K_a * K_t * \frac{1}{(J_e s + B_e)} * \frac{K_e}{s}$$

Using this transfer function, we can determine the Zero-order hold function using function decomposition. We can solve for our constants A & B by finding solutions where system poles are 0.

Equation 2: Partial Fraction Decomposition Setup & Constant Solutions

$$G(s) = \frac{K_a K_t K_e}{s(J_e s + B_e)} = \frac{A}{s} + \frac{B}{(J_e s + B_e)}$$

$$A = \lim_{s=0} s \frac{K_a K_t K_e}{s(J_e s + B_e)}$$

$$A = \frac{K_a K_t K_e}{B_e}$$

$$B = \lim_{s=-B_e/J_e} (J_e s + B_e) \frac{K_a K_t K_e}{s(J_e s + B_e)}$$

$$B = -\frac{J_e K_a K_t K_e}{B_e}$$

$$G(s) = \frac{K_a K_t K_e}{s(J_e s + B_e)} = \frac{K_a K_t K_e}{s B_e} - \frac{J_e K_a K_t K_e}{B_e(J_e s + B_e)}$$

$$G(s) = \frac{K_a K_t K_e}{s(J_e s + B_e)} = \frac{K_a K_t K_e}{s B_e} - \frac{K_a K_t K_e}{B_e(s + B_e/J_e)}$$

Now that we have successfully decomposed our transfer function, we can use this G(s) to find our Zoh function, as seen below.

Equation 3: Solution for G(z) Zero-Order hold function if T=0.0002

$$G(z) = (1 - z^{-1}) * Z\{\frac{G(s)}{s}\}$$

$$G(z) = (1 - z^{-1}) * Z\left\{\frac{K_a K_t K_e}{s^2 B_e} - \frac{K_a K_t K_e}{s B_e (s + B_e/J_e)}\right\}$$

$$G(z) = (1 - z^{-1}) * Z\left\{\frac{K_a K_t K_e}{B_e}\frac{1}{s^2} - \frac{K_a K_t K_e}{B_e}\frac{1}{s(s + B_e/J_e)}\right\}$$

$$G(z) = (1 - z^{-1}) * Z\left\{\frac{K_a K_t K_e}{B_e}\frac{1}{s^2} - \frac{K_a K_t K_e}{B_e}\frac{J_e}{B_e}\frac{B_e/J_e}{s(s + B_e/J_e)}\right\}$$

$$G(z) = (1 - z^{-1}) * \left\{\frac{K_a K_t K_e}{B_e}\frac{T z^{-1}}{(1 - z^{-1})^2} - \frac{J_e K_a K_t K_e}{B_e^2}\frac{z^{-1}(1 - e^{-\frac{Be}{Je}T})}{(1 - z^{-1})(1 - e^{-\frac{Be}{Je}T} z^{-1})}\right\}$$

When we plug this into MATLAB, our responses match our C2D values exactly, proving our manual solution.

Figure 3: Final G(z) for both Manual & C2D Solutions

```
MatlabG_z =

  5.805e-05 z + 5.801e-05
  -----------------------
  z^2 - 1.998 z + 0.9983
```

Prelab 2 – State Space Model

Obtain the discrete time state space model of the machine shown in Figure 1. Simulate the step response of the system using the discrete state space model of the machine given in Figure 1. Compare the results obtained from discrete transfer function and state space models.

Firstly, let's find the continuous state space model from our system. Let's first defined relevant defined system matrices, state variables, and state equations.

- X – State Vector
- Y – Output Vector
- U – Input Vector
- A – State Matrix -
- B – Input Matrix
- C – Output Matrix
- D – Feedthrough Matrix

Equation 4: State Space General Equations

$$\{\dot{X}\} = [A]\{X\} + [B]\{U\}$$

$$\{Y\} = [C]\{X\} + [D]\{U\}$$

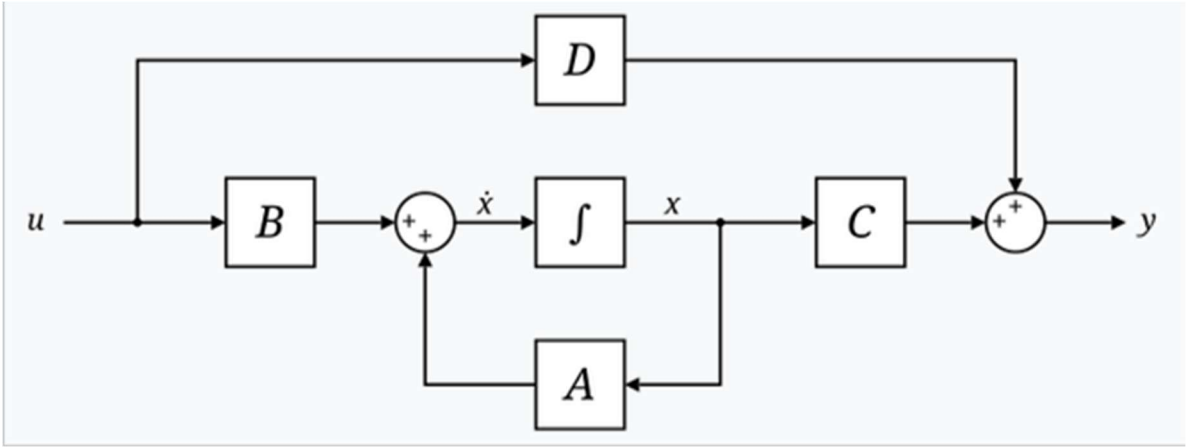

Figure 4: Block Diagram of Linear State-Space Equations (From Wikipedia)

We can define all linear equations and state variables from our main system function. To start, we can choose our input and output vectors.

Equation 5: State-Space Input, Output, and State Vectors

$$\{X\} = \begin{bmatrix} w(s) \\ X_a(s) \end{bmatrix}, \{U\} = \begin{bmatrix} V_{in}(s) \\ T_d(s) \end{bmatrix}; Y = \begin{bmatrix} X_a(s) \\ 0 \end{bmatrix};$$

Using these state variables, we can define our space-state equations.

Equation 6: State-Space Matrix Solutions

$$\begin{bmatrix} \dot{w(s)} \\ \dot{X_a(s)} \end{bmatrix} = [A] \begin{bmatrix} w(s) \\ X_a(s) \end{bmatrix} + [B] \begin{bmatrix} V_{in}(s) \\ T_d(s) \end{bmatrix}$$

$$[X_a(s)] = [C] \begin{bmatrix} w(s) \\ X_a(s) \end{bmatrix} + [D] \begin{bmatrix} V_{in}(s) \\ T_d(s) \end{bmatrix}$$

$$\frac{w(s)}{T(s)} = \frac{1}{J_e s + B_e}$$

$$J_e \dot{w(t)} + B_e w(t) = K_a K_t V_{in}(t) - T_d(t)$$

$$\dot{w(t)} = \frac{K_a K_t}{J_e} V_{in}(t) - \frac{1}{J_e} T_d(t) - \frac{B_e}{J_e} w(t)$$

$$\dot{w(s)} = \frac{K_a K_t}{J_e} V_{in}(s) - \frac{1}{J_e} T_d(s) - \frac{B_e}{J_e} w(s)$$

$$\begin{bmatrix} \dot{w(s)} \\ \dot{X_a(s)} \end{bmatrix} = \begin{bmatrix} -\frac{B_e}{J_e} & 0 \\ K_e & 0 \end{bmatrix} \begin{bmatrix} w(s) \\ X_a(s) \end{bmatrix} + \begin{bmatrix} \frac{K_a K_t}{J_e} & -\frac{1}{J_e} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_{in}(s) \\ T_d(s) \end{bmatrix}$$

$$\begin{bmatrix} X_a(s) \\ 0 \end{bmatrix} = [0 \quad 1] \begin{bmatrix} w(s) \\ X_a(s) \end{bmatrix} + [0 \quad 0] \begin{bmatrix} V_{in}(s) \\ T_d(s) \end{bmatrix}$$

Now, we can simulate the results of a step-response input using our above state-space model, and our discrete transfer function found in Step 1. These results can be seen below.

One important note is the use of our friction torque. For our state-space model, we consider both Voltage & friction torque inputs for clarity. However, in our MATLAB comparison, we will ignore the friction input and focus solely on the Voltage input, as stated in our *System Analyzed* section.

Code 1: MATLAB State-Space Solution

```
% State-space matrices (continuous time)
A = [-Be/Je, 0; Ke, 0];
B = [Ka*Kt/Je, 1/Je; 0, 0];
C = [0, 1];
D = [0, 0];

% Continuous-time state-space system
sys_cont = ss(A, B, C, D);

% First Input – Discrete State-Space System
StateSpaceG_z_Xa = ss(StateSpaceG_z.A, StateSpaceG_z.B(:,1), StateSpaceG_z.C,
StateSpaceG_z.D(:,1), Ts);
```

Our Step-Response simulations, alongside our function bode plots for our state-space model and our MATLAB C2D model can be seen below.

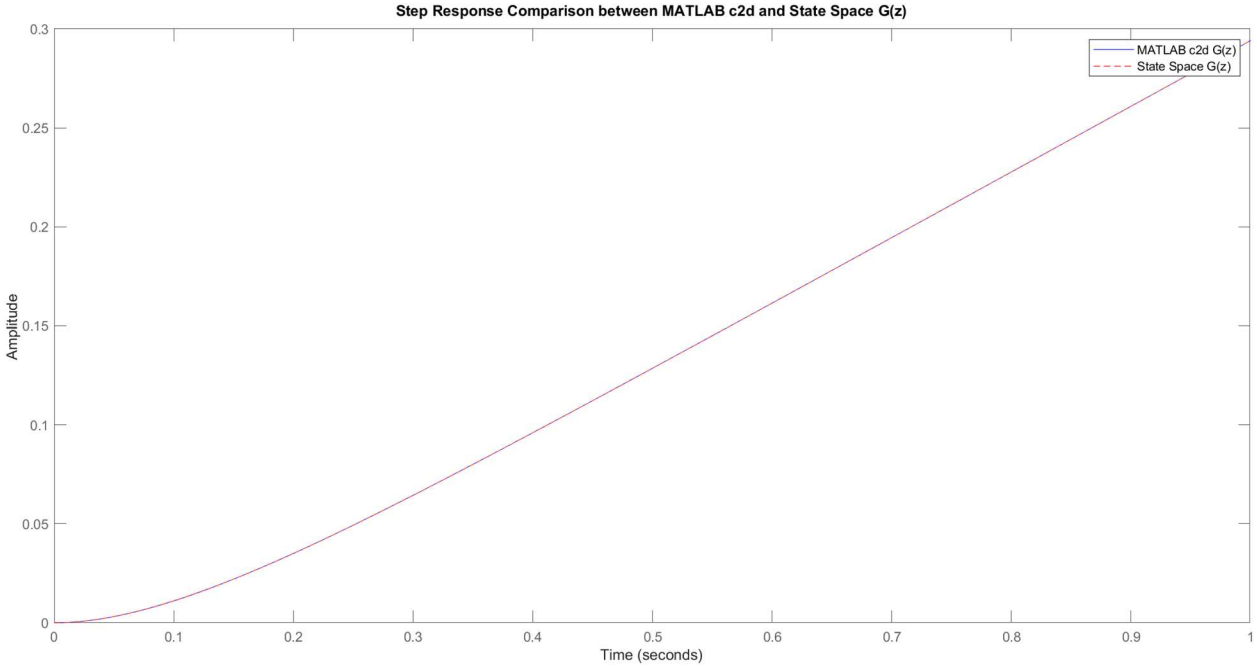

Figure 5: Step Response Comparison for State-Space Model & C2D Function

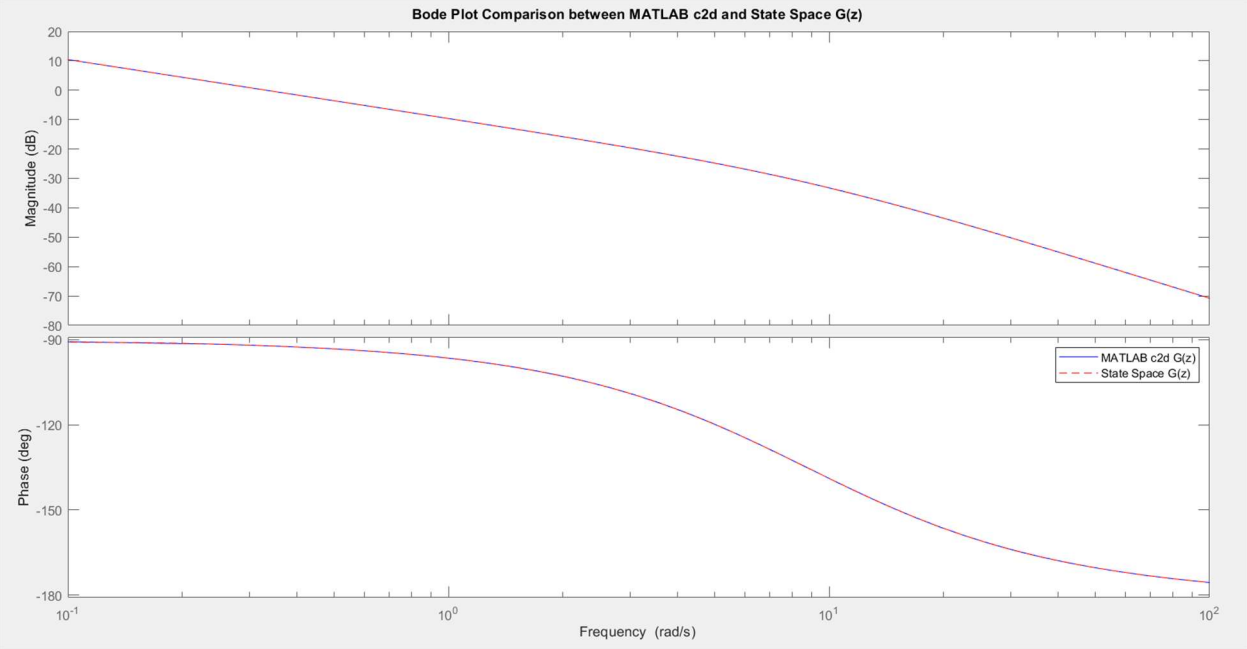

Figure 6: Bode Plot Comparison for State-Space Model & C2D Function

As we can see, our responses align perfectly, confirming this state-space model.

Prelab 3 - Stability Analysis

Assuming Closed-Loop P Controller, plot Root Locus of the drive Gol in s & z plane and observe how closed-loop poles changes as Kp increases from 0 to infinity. Derive the basic expressions manually.

Root locus is the graphical representation of the possible locations for closed loop poles for varying values of a certain system parameter. For our open loop system, our transfer function is G(s). For our closed-loop system with unity feedback, we can derive our basic continuous transfer function as follows.

Equation 7: Closed-Loop Transfer Function

$$G_{closedloop}(s) = \frac{K_p G_{ol}(s)}{1 + K_p G_{ol}(s)}$$

In simplistic terms, our closed-system poles, which determine the behaviour of our system, will change as Kp changes. Root locus will show us how our system & our closed-loop poles will change as Kp increases from 0 to inf. Our Root-locus solutions for our system in the s-plane & z-plane can be seen below.

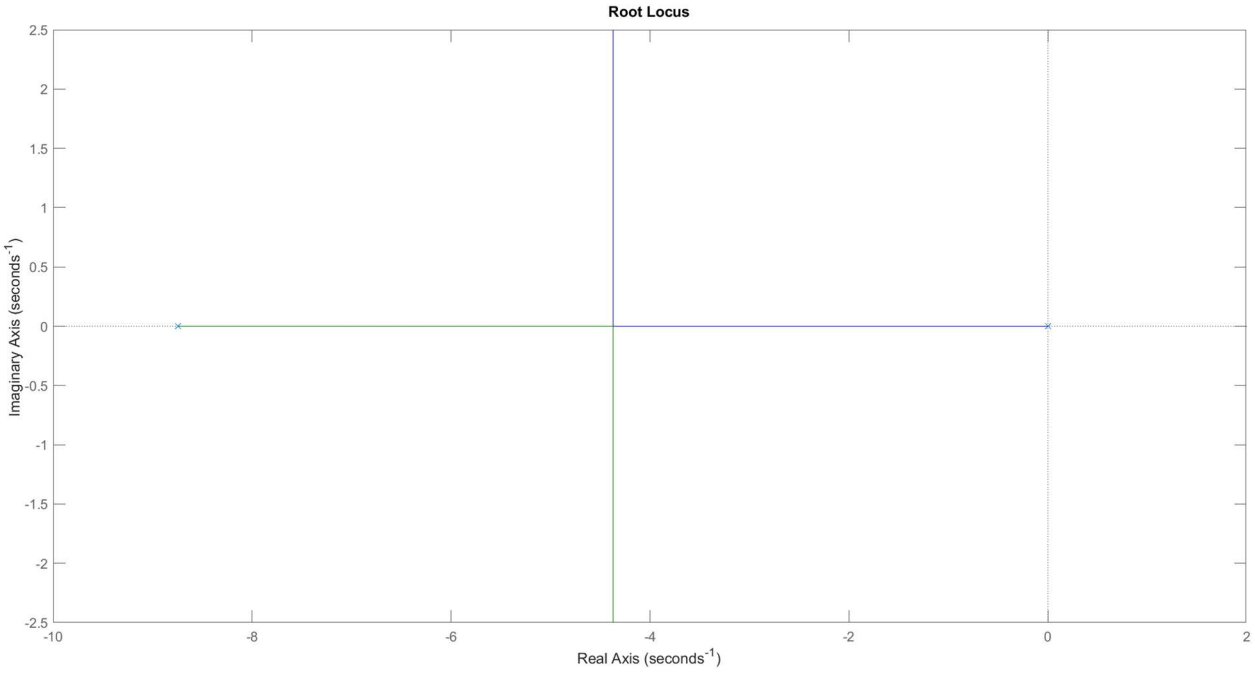

Figure 7: S-Plane Root Locus Diagram

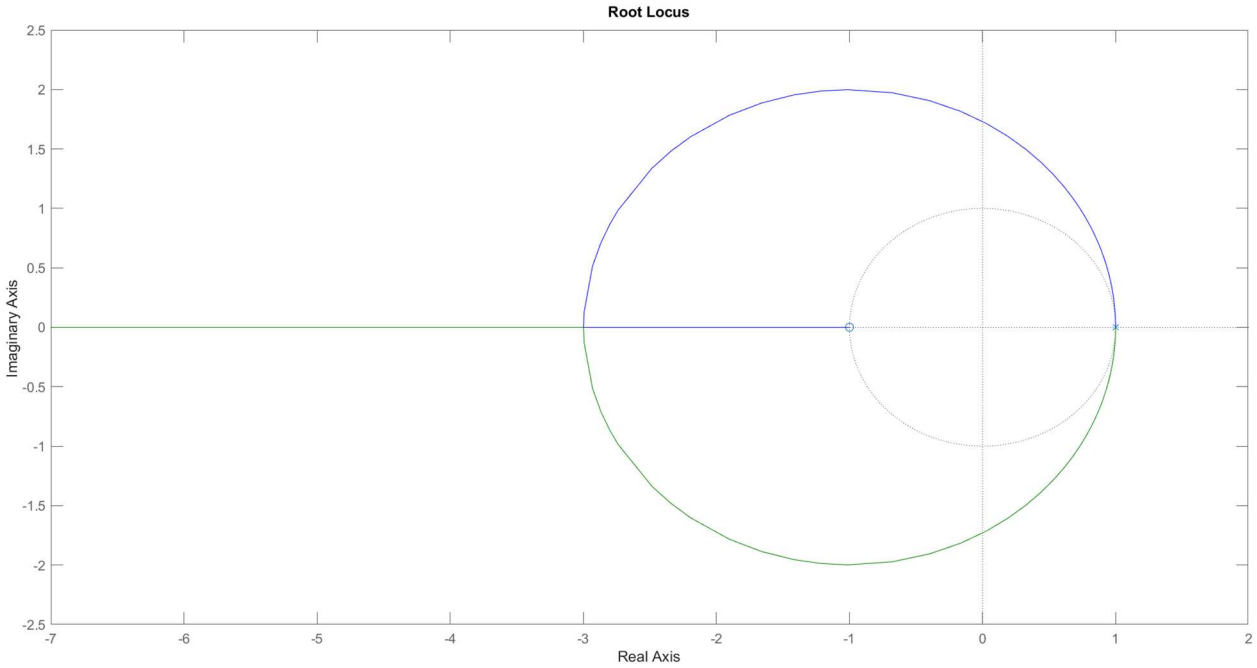

Figure 8: Z-Plane Root Locus Diagram

Find the phase and gain margins of Gol(s) and Gol(z) in the s-plane & z-plane. Comment on the stability in the closed loop systems.

Below is the Bode Plots for G_OpenLoop in the s-plane & z-plane.

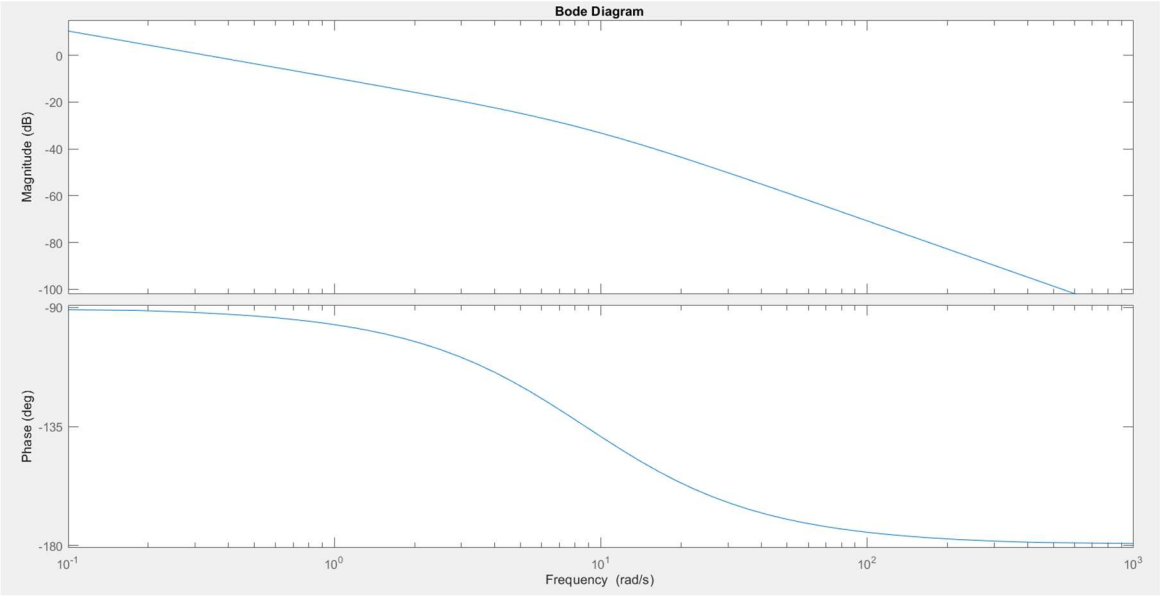

Figure 9: S-Plane Open-Loop Bode Plot

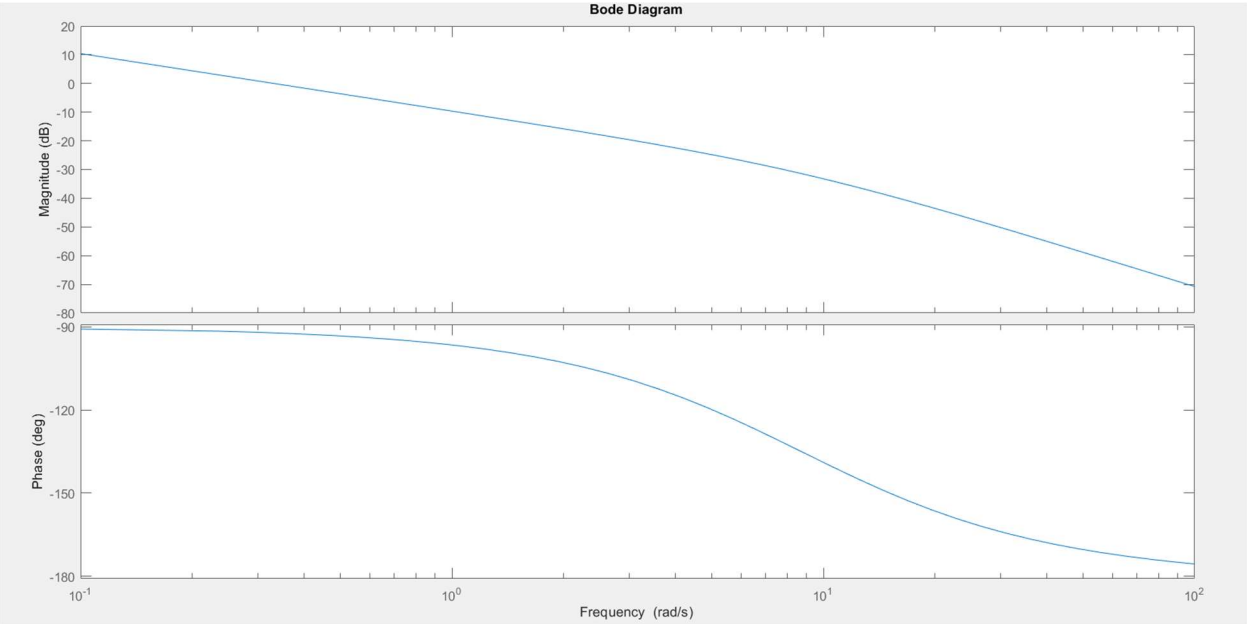

Figure 10: Z-Plane Open-Loop Bode Plot

As we can see, our bode plots for the continuous and discrete systems are the same. Since they are the same transfer function in different time domains. To determine closed-loop stability, we will observe both our bode plots and our root locus diagrams as done below.

In this S-plane, our poles begin in the negative real axis and 0 imaginary axis with Kp=0. As Kp increases, our poles begin to diverge and come closer together, until they meet at the origin of the real and imaginary axis. As Kp continues to increase past this point, our poles will diverge again, with decay rate=0 and our oscillation increasing with a steady state-response. In this situation, our system is always stable. Stable Systems can be defined as any poles where real value is negative.

In the z-plane, our poles begin with a positive real stable pole at 1. As our Kp increases, our poles will travel around the circle, representing higher oscillatory systems. At some point, our poles will begin to diverge, with some poles exiting the stable circle area, telling us that our system will become unstable at some high Kp value. Stable Systems can be defined as any poles where inside the radius of our defined system.

In short, our system will always be stable in the continuous frequency domain. As Kp increases, our decay and oscillatory responses of our system will change. In the discrete range, our decay and oscillatory responses will also change. However, at some high Kp value, our system will stop being stable and become unstable, due to the speed of our ZOH sampling rate compared to the frequency of the system itself.

Finally, the gain and phase margins of the system can be seen in the following table.

Table 2: Gain & Phase Margins of Gol in S-plane & z-plane

System	Phase Margin (deg)	Gain Margin (dB)
Gol(s)	10	49.5
Gol(z)	-20	-10.6

Discussion: Is stability in continuous and discrete domains always equivalent? Why? Using MATLAB, find the gain margin of Gol(z) for three different sampling time of 0.02, 0.002, and 0.0002. Which one is more stable? What do you conclude?

The Bode Plots for G_OpenLoop(z) for three different Sampling Times are shown below.

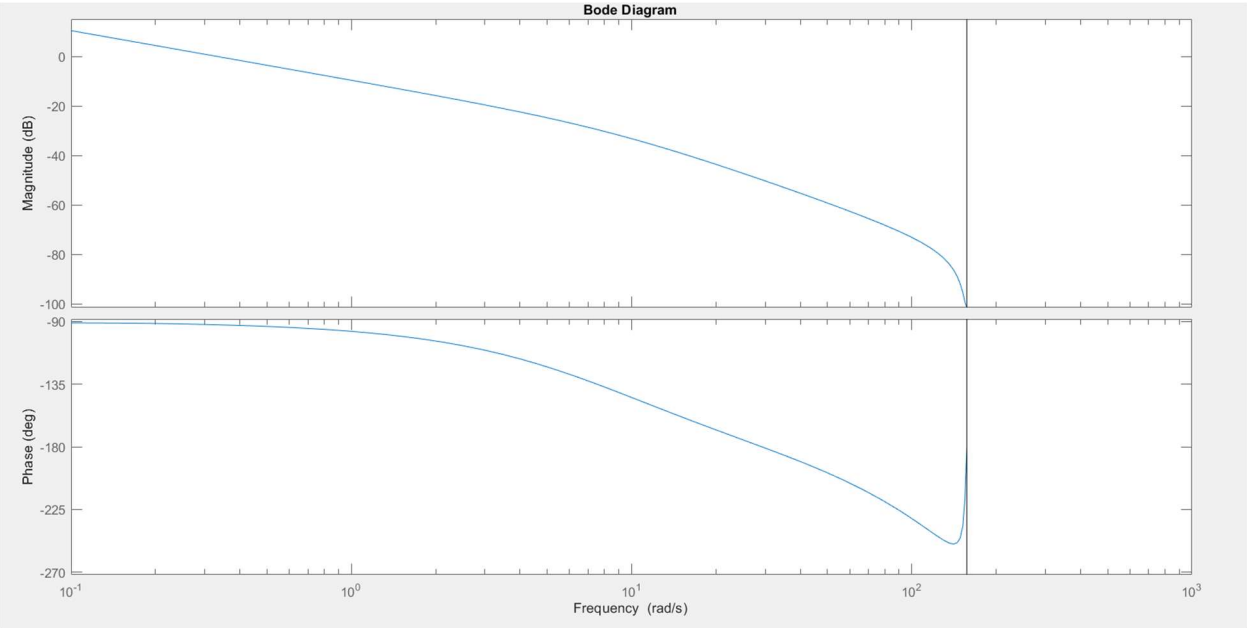

Figure 11: Z-Plane Open Loop Bode Diagram for Sampling Time = 0.02s

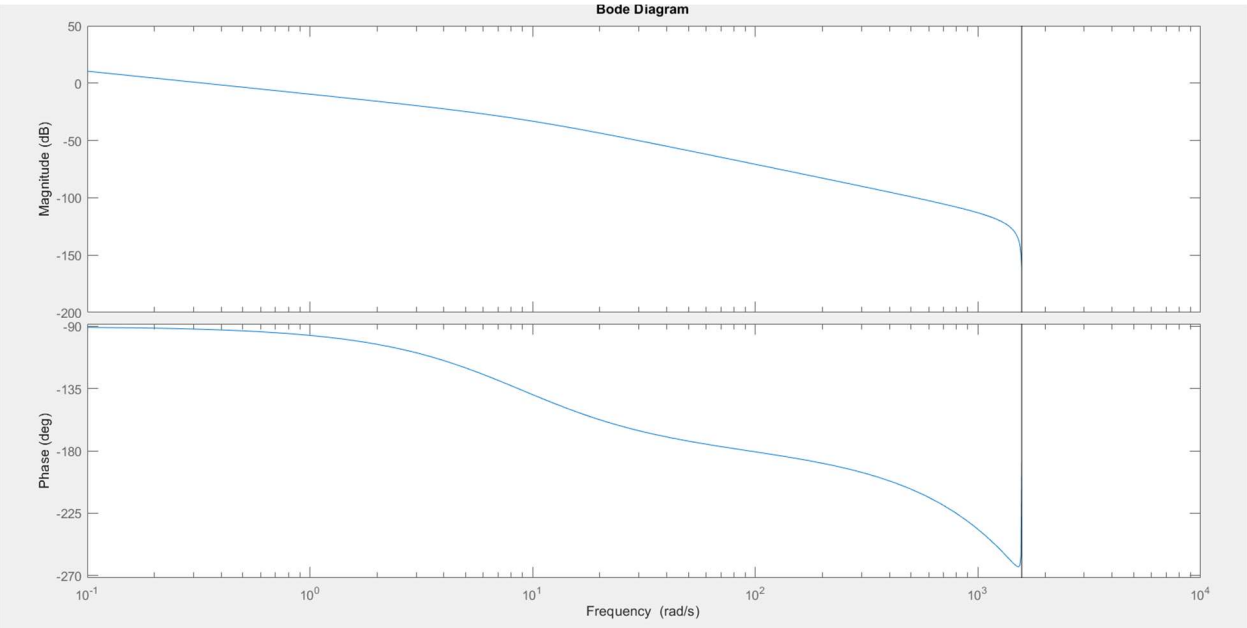

Figure 12: Z-Plane Open Loop Bode Diagram for Sampling Time = 0.002s

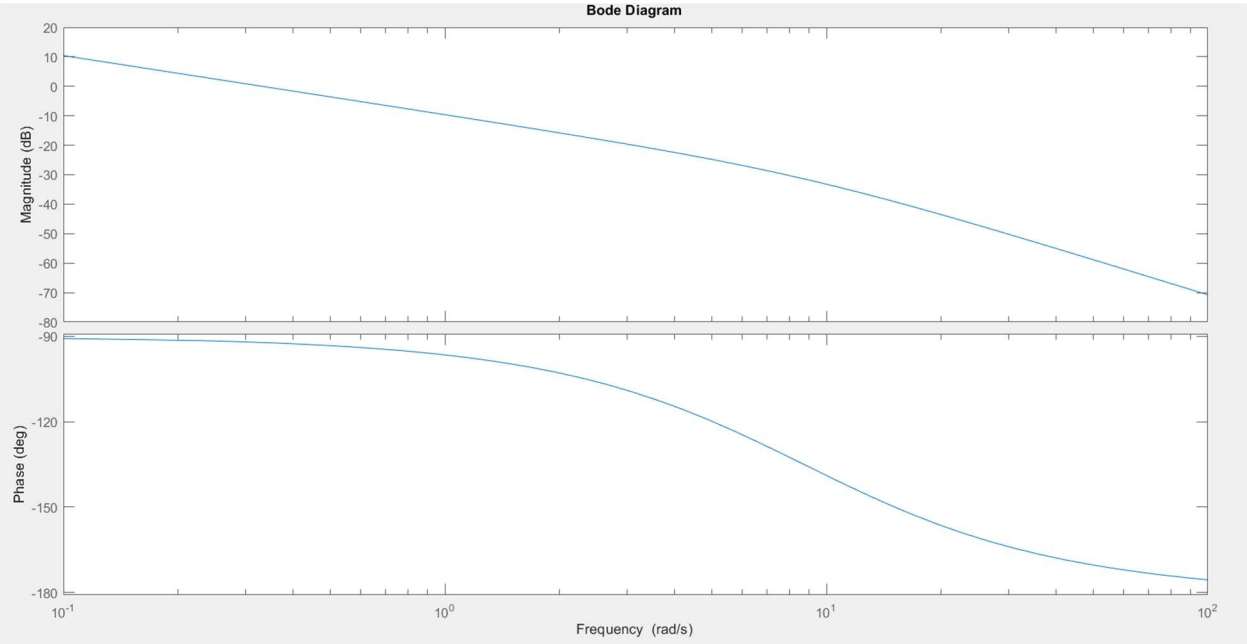

Figure 13: Z-Plane Open Loop Bode Diagram for Sampling Time = 0.0002s

The gain margins for the three values are listed in the table below. It was found by finding the difference between 0db and the gain at the point where the frequency crosses -180deg.

Table 3: Gain Margins at Various Timesteps

Time Step (s)	Gain Margin
0.02	-10.6
0.002	9.68
0.0002	27.8

As stated earlier, stability in the continuous and discrete domains are *NOT* the same. For our system, our continuous system is always stable, but our discrete system is not at some high Kp values. As Kp increases, our system will oscillate faster and our sampling time will not be able to catch up, leading to aliasing, causing our discrete system to "see" high-frequency values, leading to unstable behaviours. We need our sampling rate to be at least twice the highest frequency in the system, but as Kp increases, our sampling time will be low for this criterion, which leads to instability from our analog to digital conversion.

For the three plots above, we can determine that our third figure is the most stable. Our final system has a longer operating range and gain margin, showing a more stable system. As our sampling time increases, the gain at our crossover frequency increases, leading to a higher gain margin as we compare our crossover frequency gain to 0DB. A higher gain margin demonstrates a more stable system. This confirms our previous solution, where a higher sampling frequency leads to better ZOH analog to digital conversions and more stable systems.

Prelab 4 – P Controller Design

Using G_OpenLoop, find proportional gain Kp such that unity gain crossover frequency in z-domain is 60rad/s.

To understand how to find Kp properly, let's go into the relationship between Kp, Gcl, and Gol a bit further. As stated earlier, our relationship between our closed-loop system, Kp, and our open-loop system is as follows.

Equation 8: Closed-Loop Transfer Function

$$G_{closedloop}(s) = \frac{K_p G_{ol}(s)}{1 + K_p G_{ol}(s)}$$

In part 3, we discussed our Kp will affect the poles of our system and change system stability. However, Kp will also affect our system bandwidth. As lower frequencies, our gain function is approximately 1, representing our operating bandwidth range. As Kp increases, our bandwidth increases. However, as found in part 3, our poles also changes as Kp increases. If we increase our Kp too much, we will cause instability at higher frequencies, causing system issues.

In short, we want to increase Kp to increase our system bandwidth, but not too high so that we cause system instability.

As seen in equation 8, our Kp is directly apart of our Loop Return Ratio, L. As L increases, our bandwidth increases. The frequency at which our Loop Return Ratio crosses a gain of 1 is called our unity gain crossover frequency. At this frequency, our gain will begin to decrease. At this frequency, our system will have a specific phase. If our phase at unity gain crossover is too close to -180, instability will occur.

In short, we can use our relationship between Kp, unity gain crossover frequency, and output gain to determine Kp. Afterwards, we will check the phase margin at the given Kp to ensure it isn't close to -180deg. We would check phase margin by finding the phase where the gain is 0DB; our unity gain cross over frequency. At this point, we would determine the phase and compare it to -180deg to determine phase margin.

Using the relationships defined above, we can calculate Kp. Our transfer function for our system is defined below, as found earlier in the prelab. We can enter our s=0+jw value into our transfer function to determine our Kp_threshold value.

Equation 9: Kp Threshold Solution with s=jw=60j

$$G(s) = \frac{K_a K_t K_e}{s(J_e s + B_e)}$$

$$G(60j) = \frac{K_a K_t K_e}{60j(J_e * 60j + B_e)}$$

$$k_p = \frac{1}{G(60j)} = 1.252$$

To find our phase margin, let's plot our Loop Return Ratio, which is our plant & our Kp controller combined, and find where our gain is 0.

Equation 10: Loop Return Ratio Equation

$$LRR(s) = K_p \frac{K_a K_t K_e}{s(J_e s + B_e)}$$

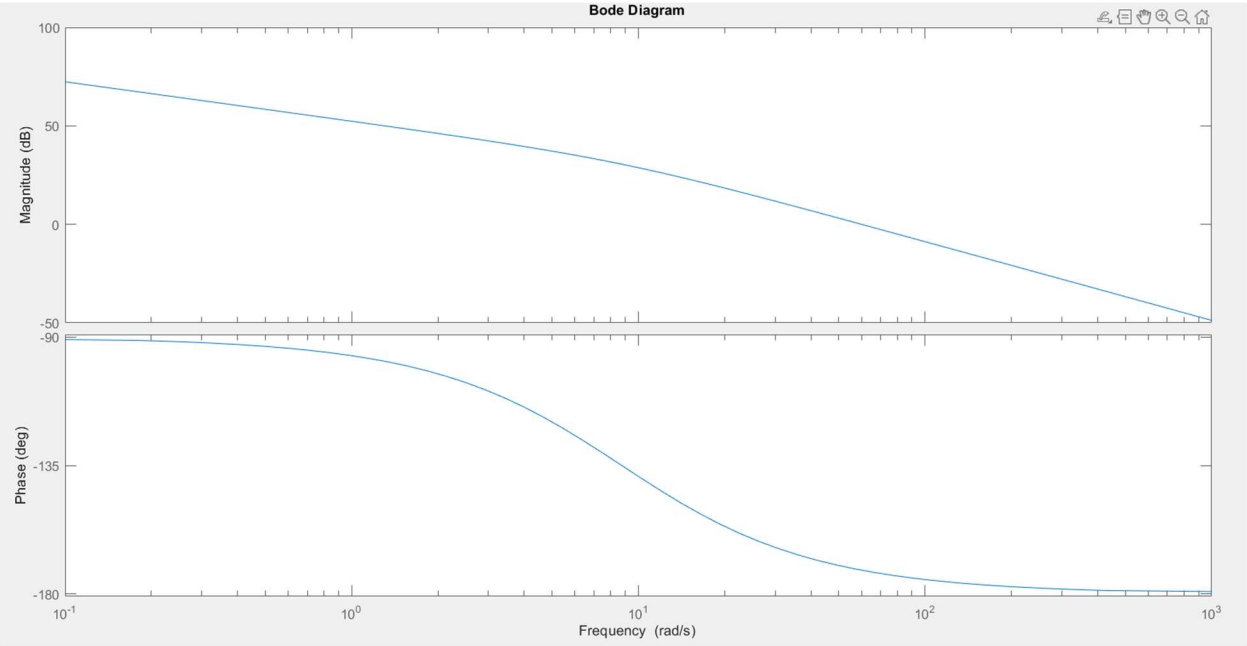

Figure 14: LRR Bode Plot & Phase Margin

Using MATLAB, I extracted our current plant phase, which is -180.8312, which tells us that our system can easily become unstable. In the future, we'd like to add more phase at this location to compensate our small phase margin.

Create a Simulink model for our system with P-Controller including zero order hold. Include Coulumb friction & Saturation Blocks.

A Simulink model was created for the closed-loop system with a p-controller, as shown in the following figure.

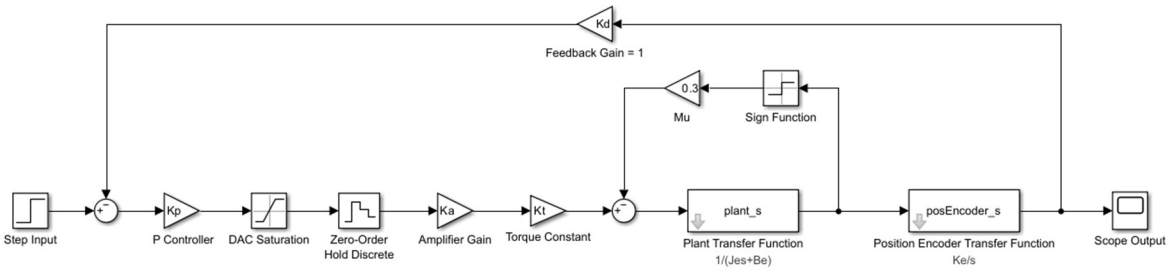

Figure 15: Simulink P-Controller System Model

Obtain step response of digital system with and without friction models. Comment on Coulomb friction effect on overshoot, rise time, and settling time.

Using this model, I will determine the step response of the digital system with and without the friction model. We will use our Kp value determined previously, and constant values defined in the *Lab Manual*. To better comment on Coloumb Friction's effects, we will use multiple other friction values besides 0 and 0.3

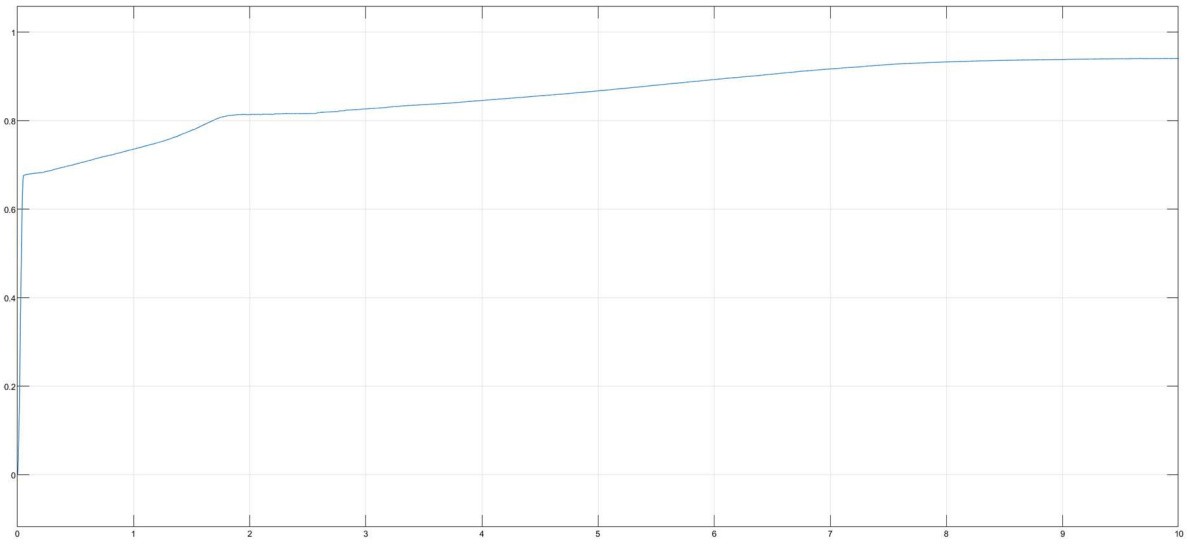

Figure 16: Step Response Output for System with Coulomb Friction = 0.5Nm

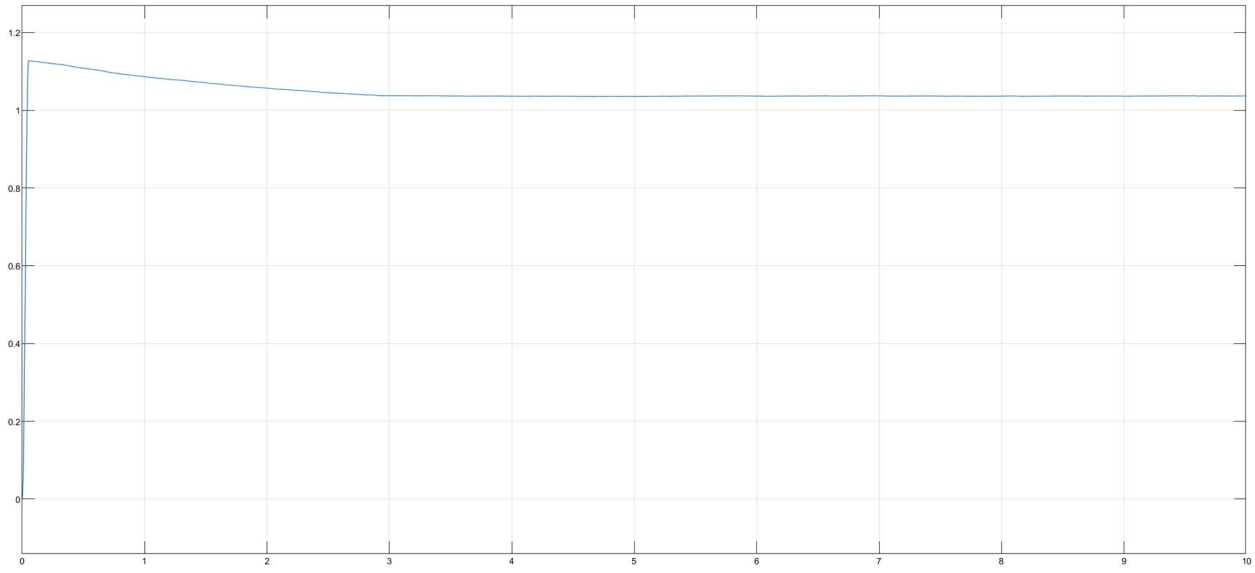

Figure 17: Step Response Output for System with Coulomb Friction = 0.3Nm

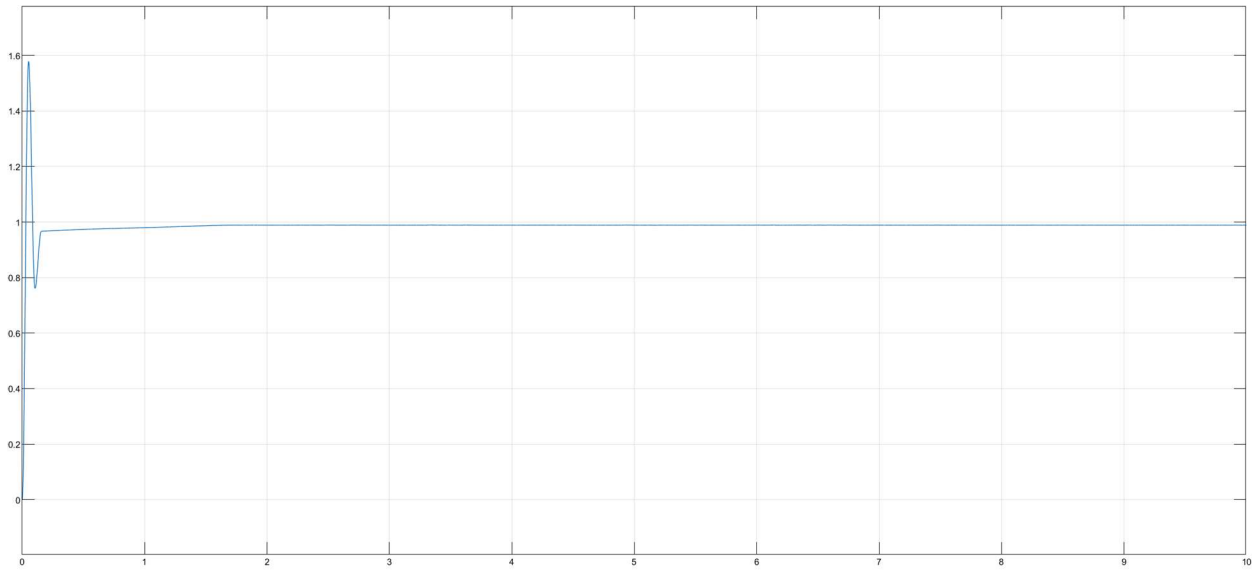

Figure 18: Step Response Output for System with Coulomb Friction = 0.1Nm

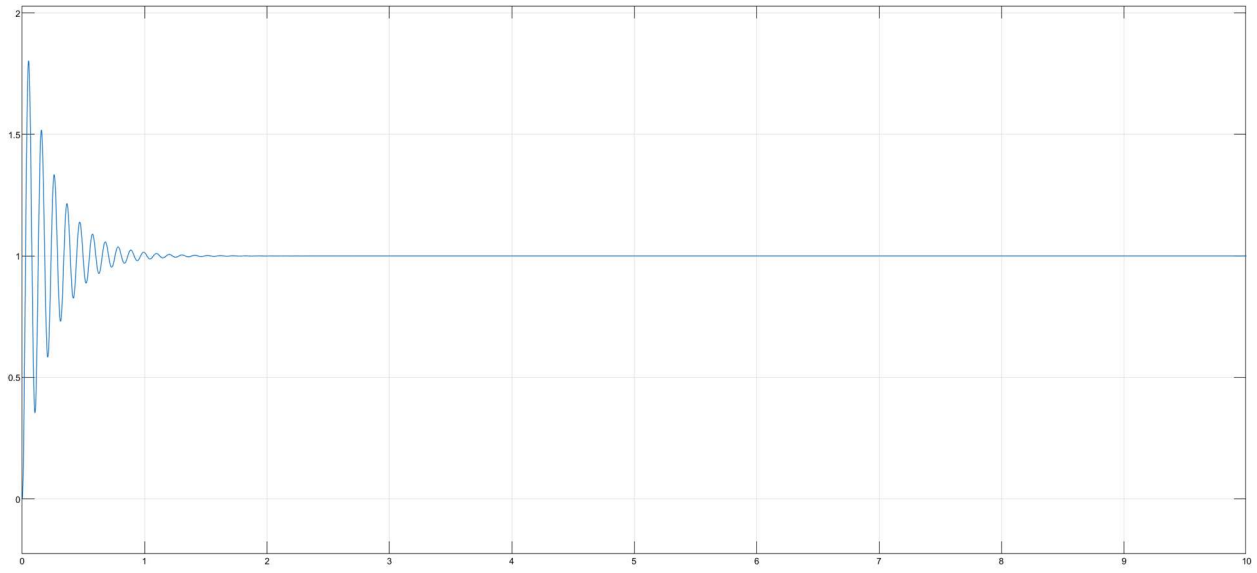

Figure 19: Step Response Output for System with Coulomb Friction = 0Nm

As seen in the following graphs. Coulomb Friction plays a major role in overshoot, rise time, & settling time.

- When Coulomb Friction is high, we have an overdamped system, where settling time is high, rise time is high, and overshoot doesn't occur, as the system slowly builds up to our final value.
- When Coulomb Friction is low, we have an underdamped system, where settling time is faster, rise time is faster, and overshoot does occur, as the system oscillates as its gets to our final position.
- For Coulomb Friction = 0.3, our system overshoots a bit, but decreases to its steady-state value, showing an underdamped system that's closer to a perfectly damped system as there are no oscillations.

How does saturation block affect overshoot, rise time, and settling time.

Similar to the last question, we will provide plots of multiple outputs with different saturation limits. Our coulomb friction constant will remain at 0.3.

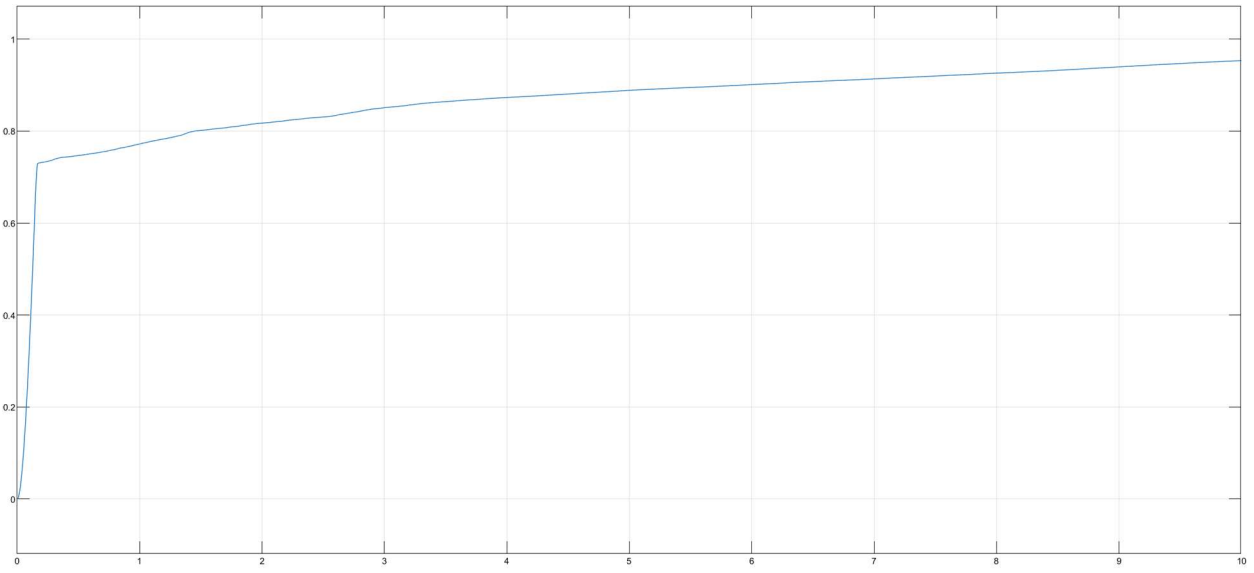

Figure 20: Step Response Output for System with Saturation = +-0.5

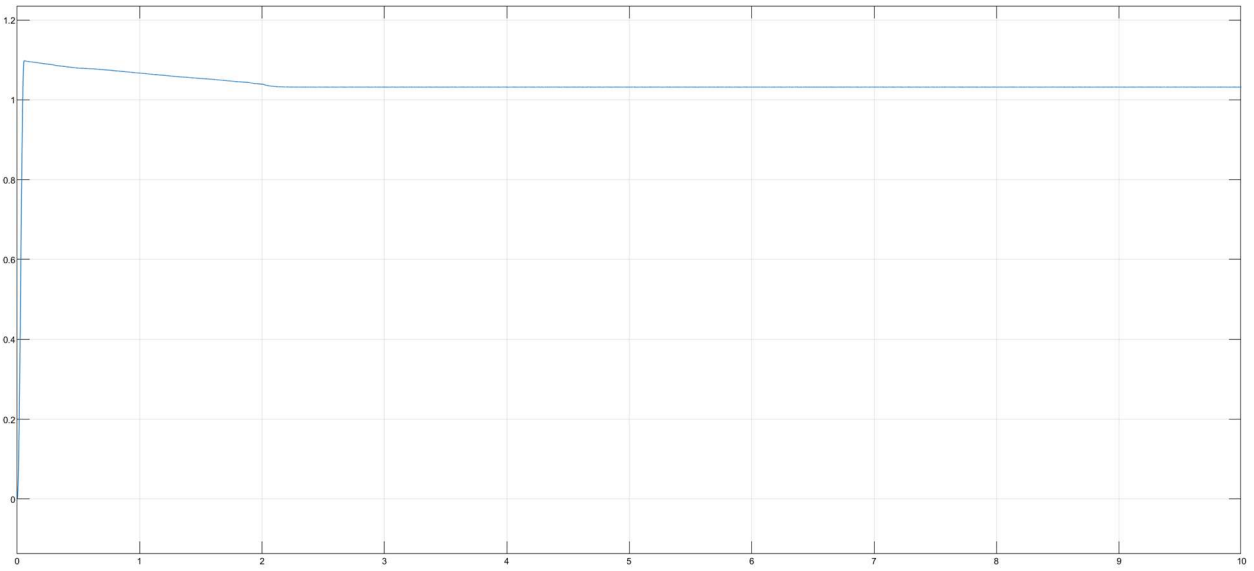

Figure 21: Step Response Output for System with Saturation = +-1

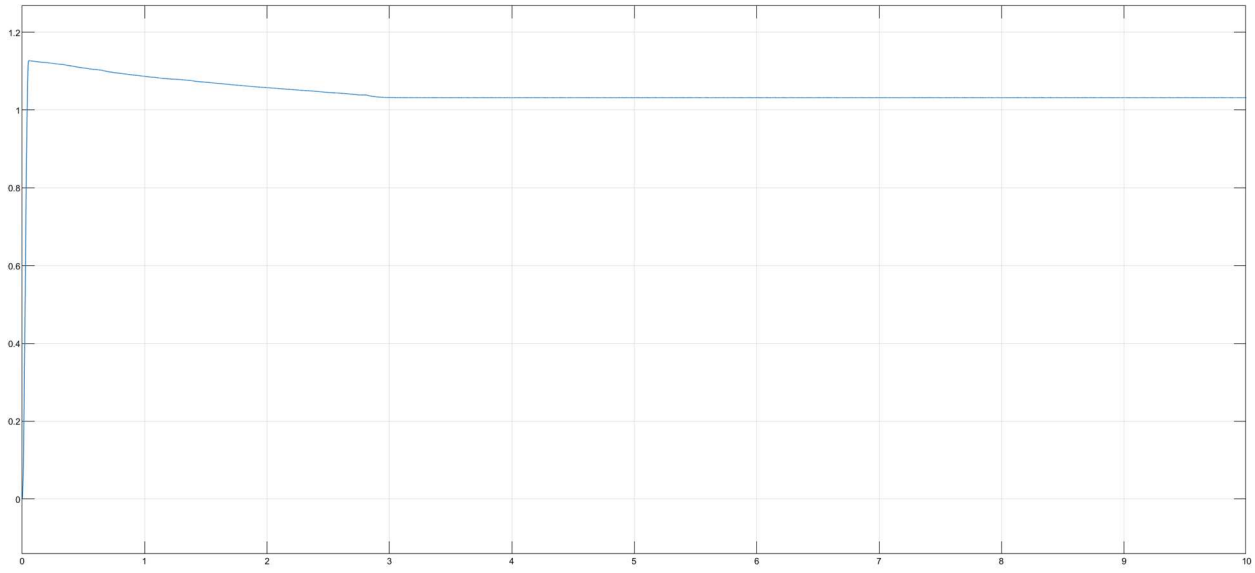

Figure 22: Step Response Output for System with Saturation = +-3

The saturation block tries to model the physical limits of a system. In this system, it limits the input voltages into the system so it could match what the actuator and hardware can deliver. At lower limits, the system might try to hit these higher voltages, but are unable to, leading to slower overall responses, like a damped system.

We can use this information to determine how saturation affects rise time, settling time, and overshoot. At lower limits, our rise time and settling times are slow, with no overshoot occurring, similar to a damped system. At higher limits, our system will react faster, with possible overshoot occurring. However, the system will be limited by the other friction constants into the system. For example, with higher saturation blocks, our output is still the same, as seen in Figure 21 and 22. This is because our system is limited by the Coulomb Friction still.

Prelab 5 – Lead Lag Compensator Controller Design

Determine Lead Compensator Parameters if gain crossover frequency is 377rad/s. Plot Loop Return Ratio System to confirm design criteria is met.

Lead compensators can be designed to add additional phase near the unity-gain cross over frequency. As we saw in Part 4, we had a small phase margin at cross-over, which can lead to instability at faster sampling times. We can use a lead compensator to add additional phase, preventing this.

Our target crossover frequency is 377 rad/s. Our target phase margin is 60deg, as per the *Lab 2 Manual*. We found our current phase in Part 2, which is -172deg. Using these values, we can determine the required peak phase of our lead compensator using the following equation.

Equation 11: Peak Phase Calculation

$$\emptyset_c = \emptyset_m - \emptyset_p - 180°$$

$$\emptyset_c = 60° - (-180.831) - 180°$$

$$\emptyset_c = 60.831°$$

Using our required peak phase, we can calculate our required lead controller parameters.

Equation 12: Lead Compensator Parameter Calculations

$$\alpha = \frac{1 + \sin \emptyset_c}{1 - \sin \emptyset_c} = \frac{1 + \sin 60.831°}{1 - \sin 60.831°} = 14.7711$$

$$\tau = \frac{1}{w_c \sqrt[2]{\alpha}} = \frac{1}{377 \sqrt[2]{14.7711}} = 6.9016 * 10^{-4}$$

Using these values, we can calculate Proportional Gain Kp for this system. We can calculate Kp using the following equation.

Equation 13: Lead Compensator Proportional Gain Calculation

$$|L(jw_c) = 1 = |KC_0(jw_c)G(jw_c)|$$

$$C_0(377j) = \frac{\alpha\tau(377j) + 1}{\tau(377j) + 1}$$

$$G(377j) = \frac{K_a K_t K_e}{60j(J_e * 60j + B_e)}$$

$$k_p = \frac{1}{|C_0(60j)G(60j)|} = 12.7376$$

To confirm that our lead lag compensator is working as expected, we can plot our Loop Return Ratio Bode Plot, as seen in the following figure.

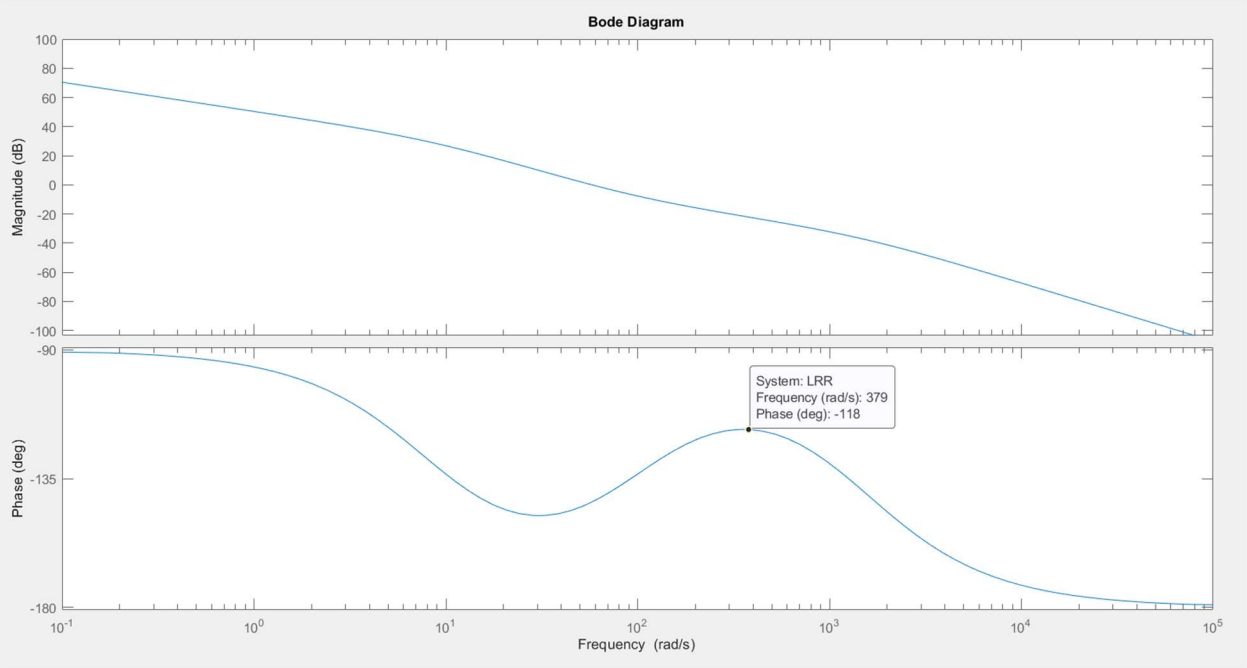

Figure 23: Lead Lag Compensator LRR Bode Plot

As you can see in the above plot, we have injected phase into our system at our crossover frequency, which removes the stability issues and positive feedback issues we had earlier.

Add Integral action (ki+s)/s to lead lag compensator. Ki=wc/10. Simulate Step & ramp input responses with friction and show the effect of integral action on steady-state input. Compare results with and without integral controller.

First, let's find the results of our system using the lead-lag compensator alone. We use the block diagram function shown in the following figure.

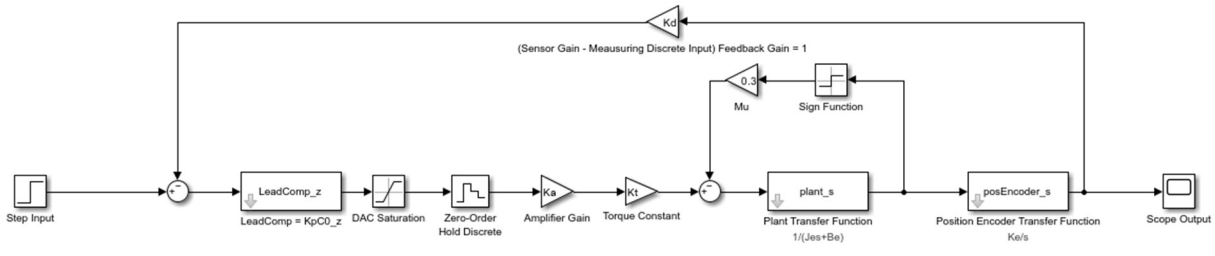

Figure 24: Lead Compensator Simulink Model

Our output results for a step-response and ramp response can be seen below. Our input response signal is yellow, and our output response signal is blue.

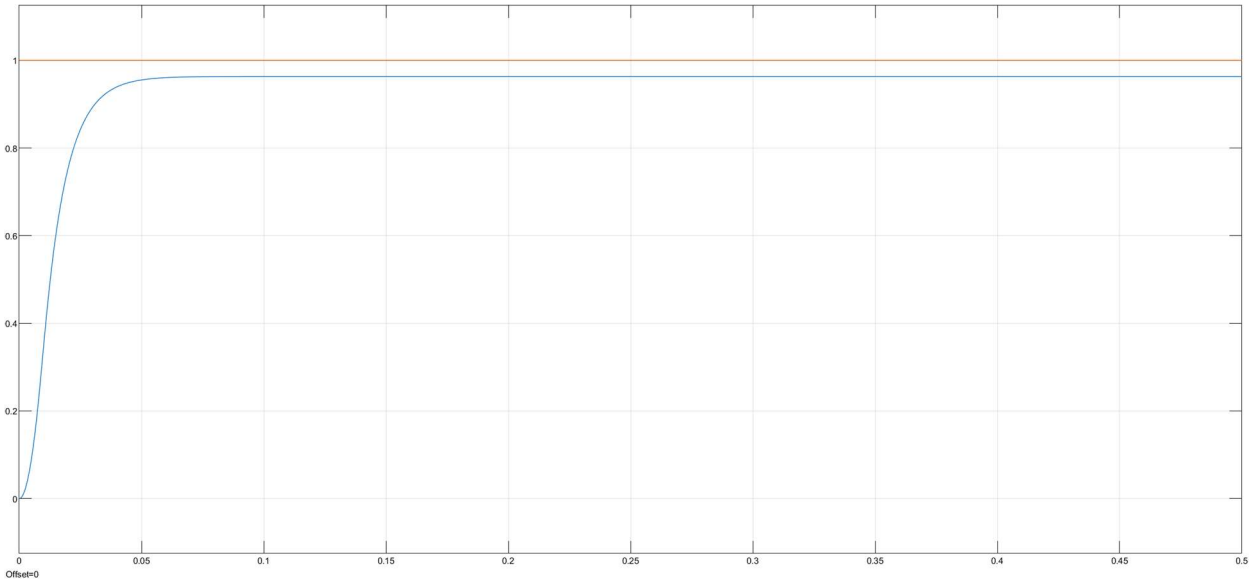

Figure 25: Step Response for Lead Compensator System

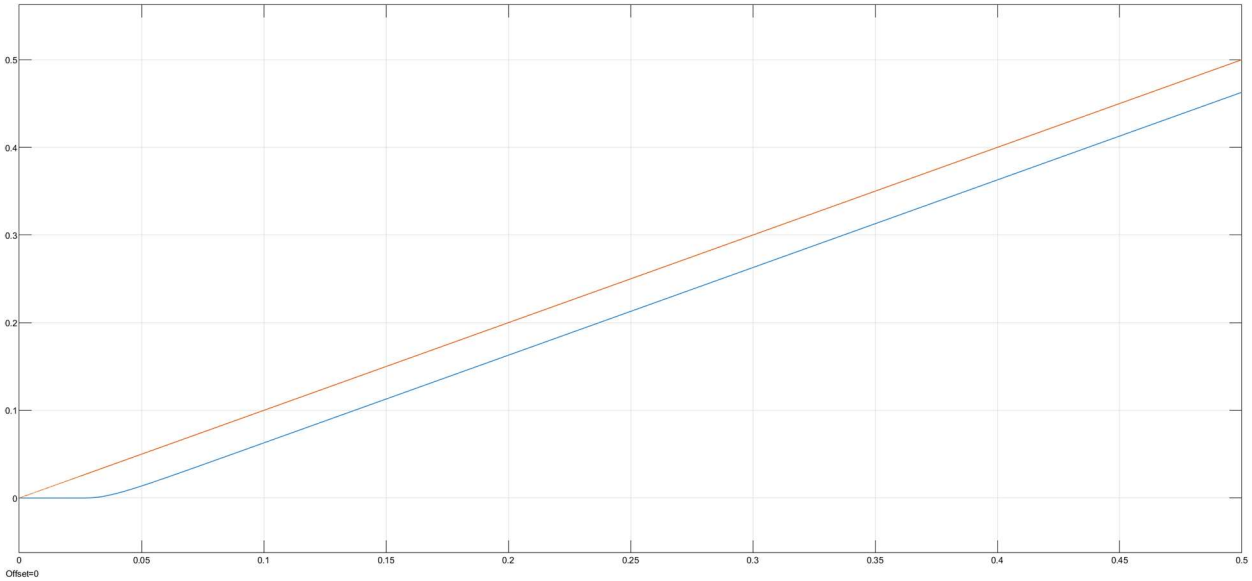

Figure 26: Ramp Response for Lead Compensator System

Our integration block diagram can be seen below. Notice how the lead compensator gain block is multiplied by the integrator function.

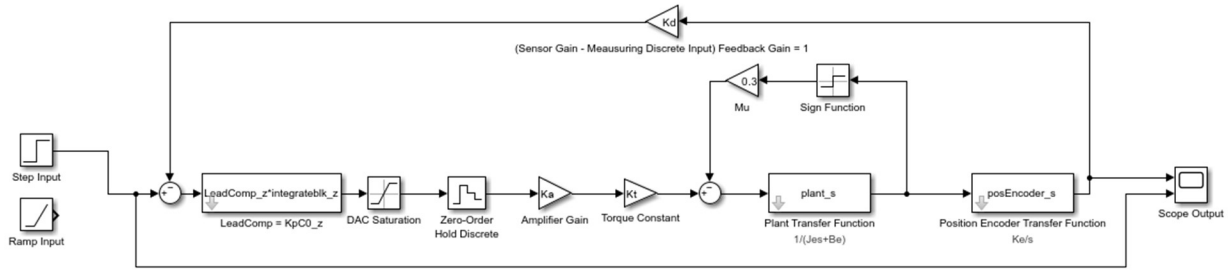

Figure 27: Lead Compensator + Integration Block Simulink Model

Our output results for this integrator system for a step-response and ramp response can be seen below. Our input response signal is yellow, and our output response signal is blue.

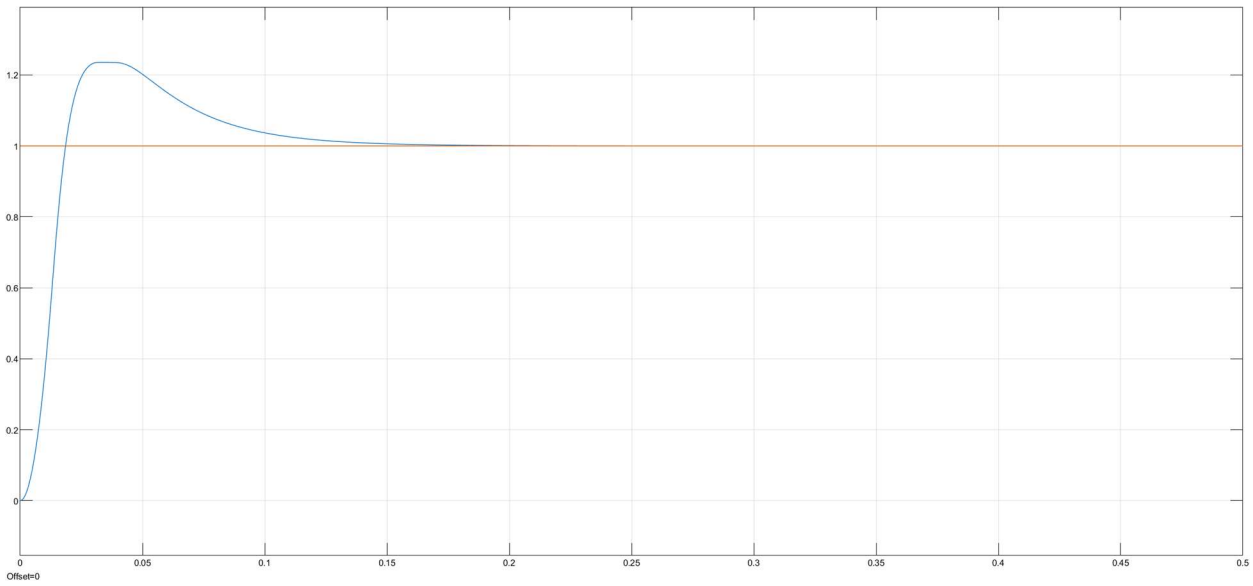

Figure 28: Step Response for Lead Compensator + Integration Block System

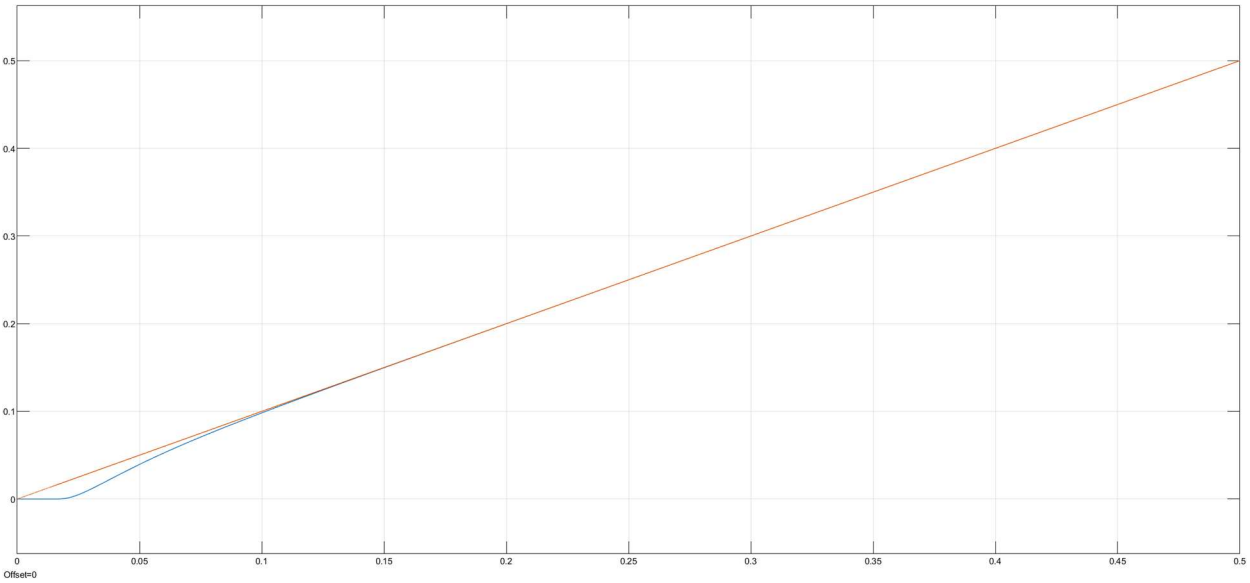

Figure 29: Ramp Response for Lead Compensator + Integration Block System

As we can see, the integration transfer function affects our system dramatically. For the original lead-compensator system without integration. Our output results match our input expected value, but our final steady-state values never fully hit our expected values. I assume that is due to friction. When we implement our integration transfer block, we see that our final output will match our intended input. For the step input, our final output will overshoot and come to steady-state at the intended final value. For the ramp function, our final output will have an increasing slope that decreases over time to match our final expected value.

To finalize this section, the following table shows our parameter values for all three Controller utilized in this prelab.

Table 4: Final Controller Parameters

Controller	Wc (rad/s)	Kp	Alpha	Tau	Ki
K Controller	60	1.252	N/A	N/A	N/A
Lead Compensator	377	12.7376	14.7711	$6.9016 * 10^{-4}$	N/A
Lead Compensator + Integration	377	12.7376	14.7711	$6.9016 * 10^{-4}$	37.7

Prelab 6 – Discussion

*Plot Magnitude & Phase of LRR, Lead Compensator, Lead Compensator+Integration, LRR*LL, LRR*LLI. Comment on how lead compensator and integrator affect magnitude and phase. Qualitively discuss how DC gain & gain crossover frequency affect steady-state error and rise time.*

The Bode Plots of all 5 systems can be seen below.

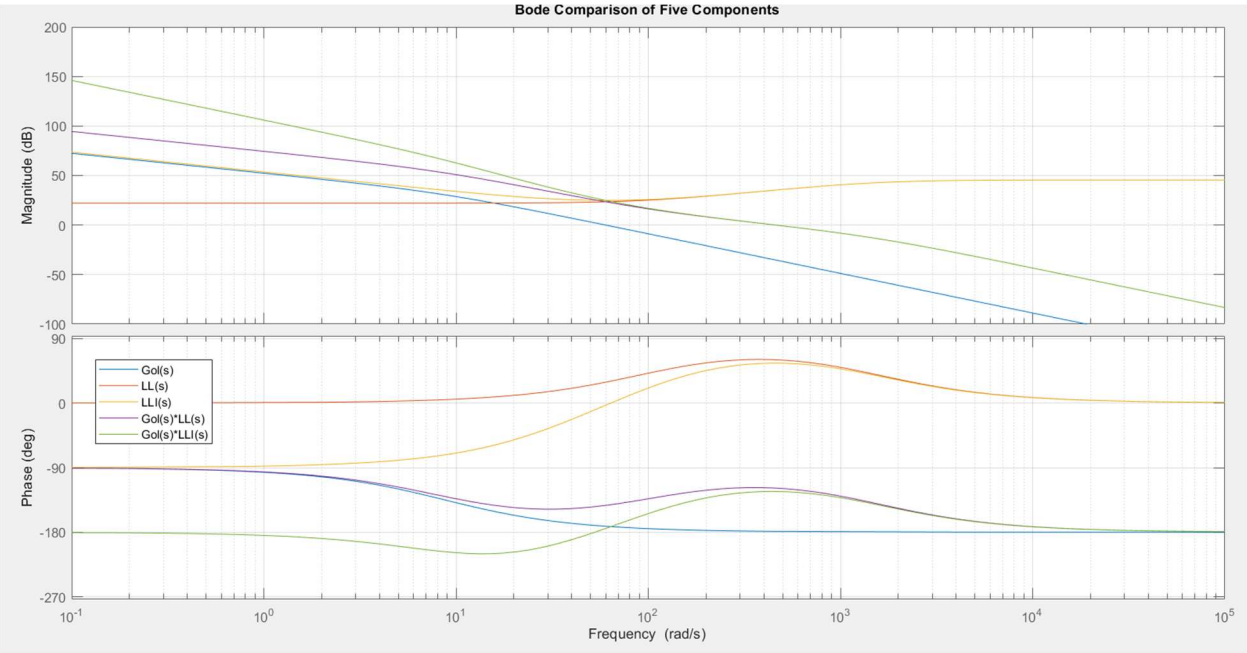

Figure 30: Bode Plots for Multiple Transfer Functions

Since the lead compensator and Integration Block is multiplied by our LRR, our Bode Responses are also multiplied by each other. However, since they are plotted on a log-log axis, it's the same as adding each bode plot together. For example, we can see that Gol*LL is the same as adding the Gol & LL transfer functions blocks together.

DC gain will reduce steady-state errors as the system can more accurately follow input functions without significant deviation. If the gain is too high, the system could respond faster, leading to faster rise times and oscillations. However, DC gain more directly impact steady-state error then rise time.

Gain Crossover frequency allows the system to handle higher frequencies while remaining stable, which will help steady-state error being minimized as the system is stabilized. A higher gain crossover frequency will lead to a system more responsive to changes, reducing rise time. In general, gain crossover frequency more directly impacts rise time than steady-state error.