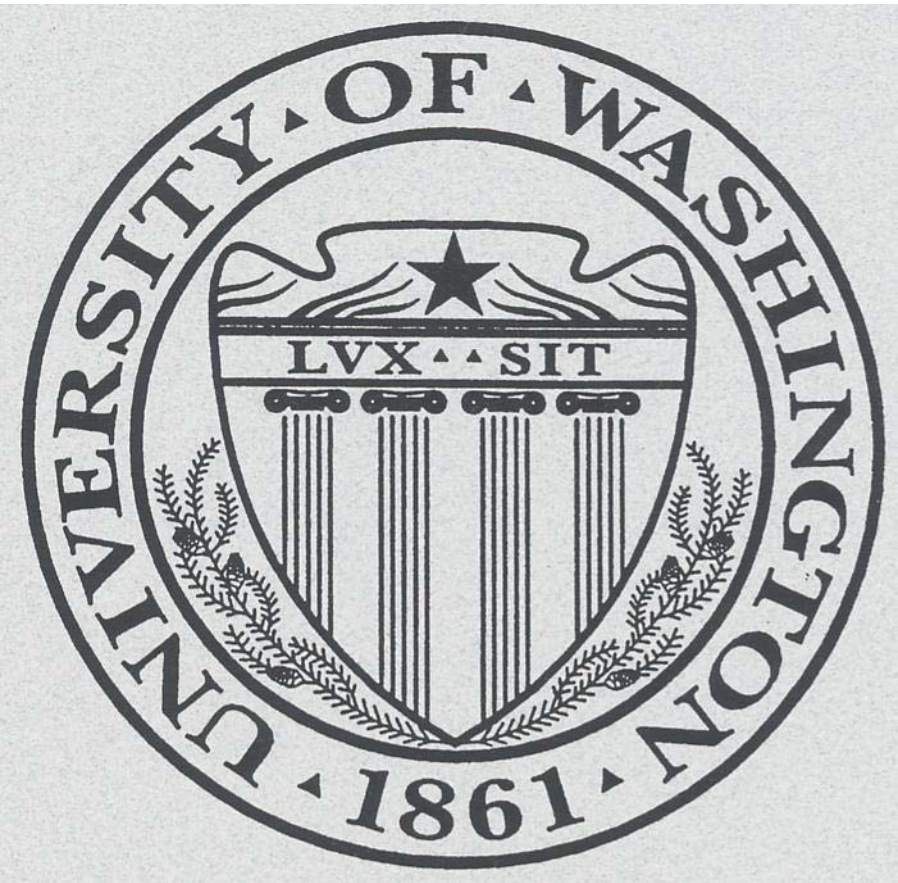
UNIVERSITY OF WASHINGTON
Department of Aeronautics and Astronautics

Modeling and Control of a Flexishaft System



March 19, 2003

Christopher Lum
Travis Reisner
Amanda Stephens
Brian Hass

AA/EE-448 Controls Sensors and Actuators
LAB EXPERIMENT III:
Modeling and Control of a Flexishaft System
March 19, 2003
Group C-1

- ***Brian Hass*** **:** List of symbols, experimental procedure.

  Contribution: 15%

  _____
  Brian Hass

- ***Christopher Lum*** **:** Assemble and perform experiments to determine the model parameters. Perform least squares analysis to fit DSA output. Develop mathematical system model and Simulink models. Develop MatLab code for data reduction. Perform control-law design and root locus for notch filter. Perform simulations to validate accuracy of system model. Prepare and assemble presentation. Prepare and assemble report.

  Contribution: 35%

  _____
  Christopher Lum

- ***Amanda Stephens*** **:** Experimental objectives, experimental apparatus.

  Contribution: 15%

  _____
  Amanda Stephens

- ***Travis Reisner*** **:** Assemble and perform experiments to determine the model parameters. Develop MatLab code for data reduction. Perform control-law design root locus for PID controller. Assess system performance. Prepare and assemble presentation.

  Contribution: 35%

  _____
  Travis Reisner

Estimated total time spent: 100+ hours

Table of Contents

I. Experiment Objectives 4

II. Experiment Apparatus 5

III. List of Symbols 7

IV. System Modeling 9

V. Control System Design 17

VI. Experimental Procedure 31

VII. Modeling Validation 34

VIII. Control Design Performance 66

IX. Conclusions 80

X. Appendix 81

I. Experiment Objectives

The objective of this laboratory was to design a control system that would control the positioning of an inertial load with a DC motor. The inertial load is provided by using a torque wrench and applying a torque directly to the drive motor during the initial design process (this simulates having an additional 'load' motor connected to the drive motor by a flexible shaft). However, during the extended team projects the flexishaft was used to connect the load and drive motors thereby supplying the inertial load. Several different shafts can be used to vary the flexibility between the drive motor and the load motor. The initial part of the lab utilized the "reduced configuration" which has the flexishaft removed and looked only at the drive motor's response to an input reference angle. Traditional PID control structure was used. From there the shaft was put in place and the control system was designed to position the load utilizing the drive motor, without exciting the flexible modes when a reference step of one radian was input (extended project problem E.2).

II. Experiment Apparatus

The experiment involved the use of the following electronic equipment:

- Mixed-Signal Oscilloscope Agilent 54621 D from Agilent Technologies
- Bipolar Operational Power Supply Kepco Model BOP 36-12M
- Bipolar Operational Power Supply Kepco Model BOP 20-10M
- HP 35665A Dynamic Signal Analyzer
- PC computer with a four-channel quadrature encoder input board CIO-QUAD04, and a multifunction measurement and control board CIO-DAS 1602/16 from ComputerBoards, Inc.
- Mathworks software: Simulink, Realtime Workshop and xPC target

No other electronic parts were used.

The extended laboratory experiment included the use of a removable aluminum flexishaft. During this part of the lab, the flexishaft was connected to both the load motor and the drive motor using rigid tri-screw chucks on each end. The drive motor was connected to the main drive shaft using a timing belt around two pulleys with a gear ratio of 5 to 1. All of these components are shown below in Figure 1.

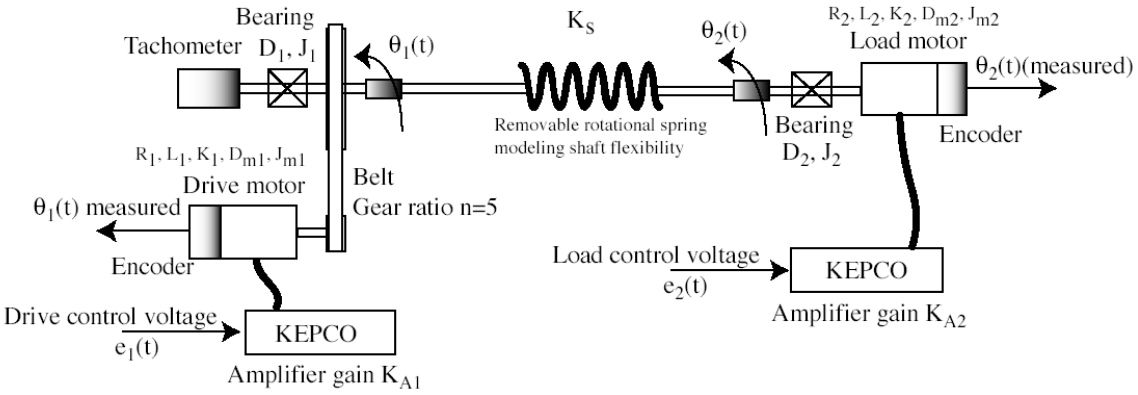


Figure 1: Flexishaft experimental apparatus schematic.

Each motor was an independent actuating Litton brush-type DC motor. Additionally, two Kepco power supplies/amplifiers were used to drive these motors.

For the initial part of the lab the flexishaft was not attached and thus only one motor and amplifier were used (the drive motor) and disturbance torques were provided by a torque wrench.

The sensors for the experiment were two optical shaft encoders (one encoder on each of the motor shafts) and a tachometer (which was disconnected and not used). The encoders provided a resolution of 1000 pulses per revolution.

III. List of Symbols

Table 1: List of symbols and values

Symbol	Definition and value
G(s)	Transfer function between θ_2 and e_1
H(s)	Transfer function between ω_2 and e_1
P(s)	Transfer function between θ_1 and T_L
K_P	Proportional gain
K_I	Integral gain
K_D	Derivative gain
T_L	Torque applied to motor
e_1	Voltage before Kepco amplifier to drive motor
e_2	Voltage before Kepco amplifier to load motor

Table 2: List of symbols and values published from gfshaft.m file

Symbol	Value	Parameter Definition
K_{A1}	3.6 V/V	Drive power amplifier gain, KEPCO BOP36-12M
R_{A1}	60 μΩ	Drive amplifier output resistance
K_1	62.0x10-3 V/(rad/sec)	Drive motor constant
R_1	1.8 Ω	Drive motor armature resistance
L_1	2.1x10-3 H	Drive motor armature inductance
J_{m1}	$24.38x10^{-6}$ kg m^2	Drive motor inertia
J_1	1.25x10-3 kg m^2	Inertial load on shaft A (Drive)
D_{m1}	60x10-6 N-m/(rad/sec)	Drive motor damping constant
D_1	42.35x10-6 N-m/(rad/sec)	Viscous friction on shaft A (Drive)
K_{A2}	2.0 V/V	Load power amplifier gain KEPCO BOP20-10M
R_{A2}	40 μΩ	Load amplifier output resistance
K_2	0.2 V/(rad/sec)	Load motor constant for preload 2V (original value $K_2$62.0x10$_{-3}$)
R_2	1.8 Ω	Load motor armature resistance
L_2	2.1x10-3 H	Load motor armature inductance
J_2	1.0x10-3 kg m^2	Inertial load on shaft B (Load)
J_{m2}	24.38x10-6 kg m^2	Load motor inertia
D_{m2}	60.0x10-6 N-m/(rad/sec)	Load motor damping constant
D_2	42.35x10-6 N-m/(rad/sec)	Viscous friction on shaft B (Load)
K_{s1}	9 N-m/rad	Stiffness of removable shaft (Hex aluminium statically determined value 10.73 N-m/rad)

K_{s2}	61.573 N-m/rad	8mm steel shaft (not used)
K_{s3}	1.75 N-m/rad	3 mm steel shaft (not used)
N	5.0	Gear ratio
k_T	7x10-3 V/rpm	Tachometer constant
p_1	1000	Drive encoder number of lines per revolution
p_2	1000	Load encoder number of lines per revolution
e_{max}	10 V	Maximum input voltage amplitude for KEPCO amplifiers

IV. System Modeling

The system during this experiment consisted of two DC motors that could be independently controlled and connected together through a flexible shaft. The mathematical model of a single motor by itself can be derived using techniques described in Lab Report II: Modeling and Control of a DC Motor[1]. As shown previously, a single motor by itself is a third order system. The state space representation of a single motor (either drive or load) is a third order system. The state space representation for the drive motor using in Sections A-D is shown below,

$$\frac{d}{dt}\begin{pmatrix} i_1 \\ \theta_1 \\ \omega_1 \end{pmatrix} = \begin{pmatrix} -\frac{R_{A1}+R_1}{L_1} & 0 & -\frac{nK_1}{L_1} \\ 0 & 0 & 1 \\ \frac{nK_1}{J_1+n^2J_{m1}} & 0 & -\frac{\left(D_1+n^2D_{m1}\right)}{J_1+n^2J_{m1}} \end{pmatrix}\begin{pmatrix} i_1 \\ \theta_1 \\ \omega_1 \end{pmatrix} + \begin{pmatrix} \frac{K_{A1}}{L_1} & 0 \\ 0 & 0 \\ 0 & \frac{1}{J_1+n^2J_{m1}} \end{pmatrix}\begin{pmatrix} e_1 \\ T_L \end{pmatrix}$$

(Eq.1)[2]

Since the output of the motor is θ_1, the output is given by

$$y = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}\begin{pmatrix} i_1 \\ \theta_1 \\ \omega_1 \end{pmatrix} \qquad \textbf{(Eq.2)}$$

For parts A-D of the lab, these equations are sufficient in order to develop a mathematical model of the system and simulate the response. The Simulink model used to simulate the response of the drive motor alone is shown in the section V: Control System Design.

[1] Lab Report 3: Modeling and Control of a DC Motor. Lum, Christopher. Reisner, Travis. Stephens, Amanda. Hass, Brian.

[2] Lab Handout: Lab Experiment III: Modeling and Control of a Flexishaft System pg. 6. Provided by Professor Uy-Loi Ly. University of Washington. Seattle, WA.

When the two motors are connected through the flexible shaft as in part E, the system becomes a sixth order system governed by Eq.3 shown below,

$$\frac{d}{dt}\begin{pmatrix} i_1 \\ i_2 \\ \theta_1 \\ \omega_1 \\ \theta_2 \\ \omega_2 \end{pmatrix} = A\begin{pmatrix} i_1 \\ i_2 \\ \theta_1 \\ \omega_1 \\ \theta_2 \\ \omega_2 \end{pmatrix} + B\begin{pmatrix} e_1 \\ e_2 \\ T_L \end{pmatrix} \qquad \textbf{(Eq.3)}$$

where

$$A = \begin{pmatrix} -\frac{R_{A1}+R_1}{L_1} & 0 & 0 & -\frac{nK_1}{L_1} & 0 & 0 \\ 0 & -\frac{R_{A2}+R_2}{L_2} & 0 & 0 & 0 & -\frac{K_2}{L_2} \\ 0 & 0 & 0 & 1 & 0 & 0 \\ \frac{nK_1}{J_1+n^2J_{m1}} & 0 & -\frac{K_S}{J_1+n^2J_{m1}} & -\frac{\left(D_1+n^2D_{m1}\right)}{J_1+n^2J_{m1}} & \frac{K_S}{J_1+n^2J_{m1}} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & \frac{K_2}{J_2+J_{m2}} & \frac{K_S}{J_2+J_{m2}} & 0 & -\frac{K_S}{J_2+J_{m2}} & -\frac{\left(D_2+D_{m2}\right)}{J_2+J_{m2}} \end{pmatrix}$$

$$B = \begin{pmatrix} \frac{K_{A1}}{L_1} & 0 & 0 \\ 0 & \frac{K_{A2}}{L_2} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{1}{J_1+n^2J_{m1}} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Since the tachometer is not available to directly measure angular velocity, the only outputs of the system are θ_1 and θ_2 and the output is given by

$$\underline{y} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} i_1 \\ i_2 \\ \theta_1 \\ \omega_1 \\ \theta_2 \\ \omega_2 \end{pmatrix} \qquad \textbf{(Eq.4)}$$

The actual values of all of the constants have been previously provided in Section III: List of Symbols.

Since sections A-D only involved designing a rough PID controller for the drive motor, only Eq.1 and Eq.2 are needed to derive relationships between inputs and outputs and model the system. As stated before, all of the techniques needed to derive the parameters in Eq.1 and Eq.2 are detailed in a previous report.

For part E.2, it is desired to accurately command a θ_2 using only the drive motor and feedback of θ_2. This requires an accurate transfer function or state space representation between e_1 and θ_2. This transfer function can be obtained from Eq.3 and Eq.4 by picking appropriate columns of the B matrix and appropriate rows of the C matrix. The transfer function G(s) can be found using

$$G(s) = C(sI - A)^{-1} B \qquad \textbf{(Eq.5)}$$

Using the published values in the gfshaft.m file, the following transfer function is obtained (written in bode plot normalized form).

$$G(s) = \frac{\theta_2}{e_1} = \frac{2.15 \times 10^{12} \left(\frac{s}{857.18} + 1 \right)}{27.69 s \left(\frac{s}{27.69} + 1 \right) 827.60 \left(\frac{s}{827.60} + 1 \right) 835.18 \left(\frac{s}{835.18} + 1 \right) 14011.61 \left(\frac{s^2}{14011.61} + \frac{s}{565.3} + 1 \right)}$$

(Eq.6)

The pole zero map of Eq.6 is shown below in Figure 2

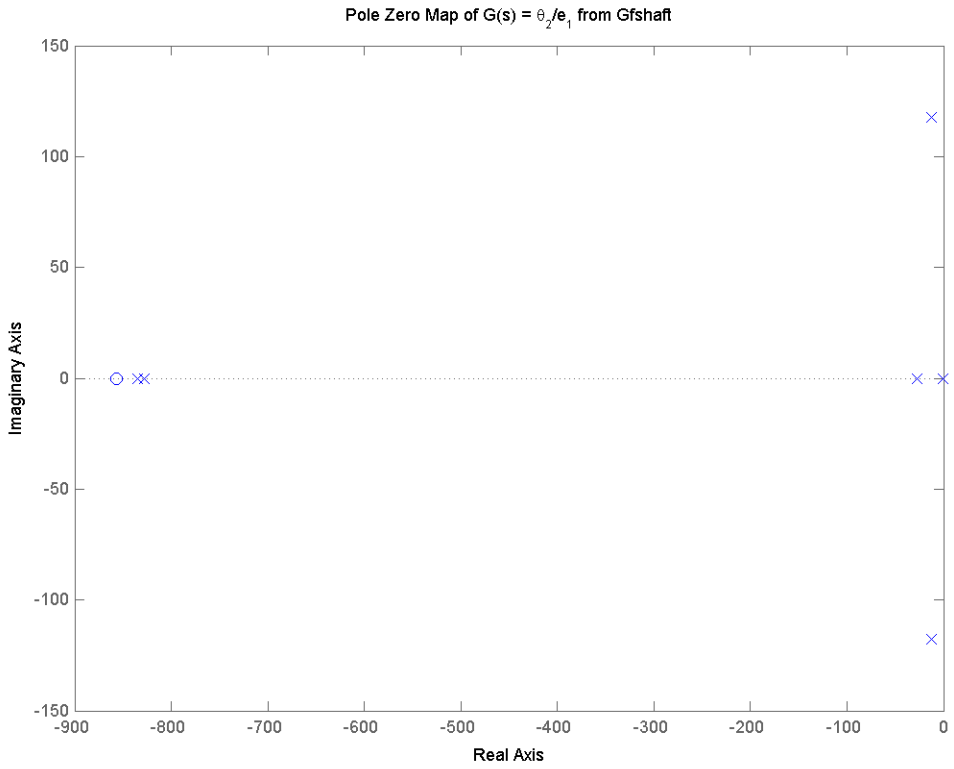


Figure 2: Pole/zero map of G(s) obtained from gfshaft.m file

The interesting thing to notice in Figure 2 is the location of all of the poles. There are essentially 6 poles and one zero. This is expected by looking at Eq.6. However, two of the poles and the zero have extremely high break frequencies (roughly 800 rad/s). This will become important during least squares analysis later.

As stated before, in order to design an accurate notch filter using root locus techniques, it is crucial that the location of all of the roots are accurately located on the real/imaginary plane. In order to determine the location of these poles, a DSA analysis is performed between the input e_1 and ω_1 which is denoted as G(s). Eq.6 can easily be modified to represent the transfer function between e_1 and ω_1 by simply removing the pole at the origin which represents an integrator. The output of the DSA can be plotted against the modified Eq.6 in order to ascertain the accuracy of the published values. This is shown below in Figure 3.

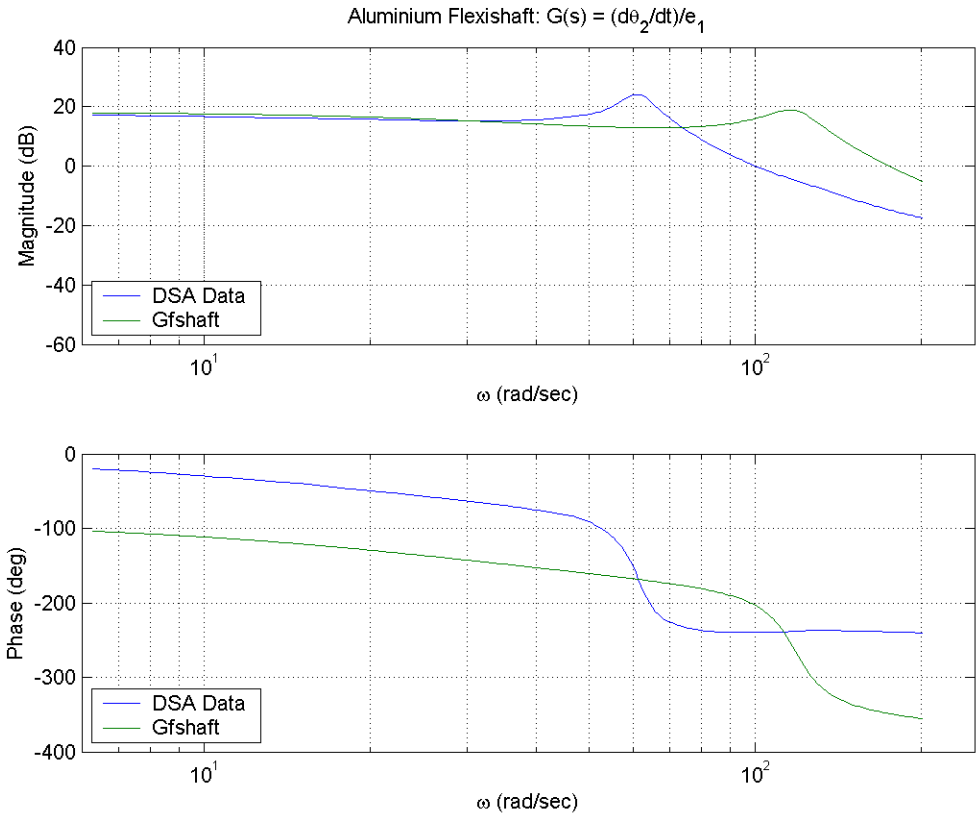


Figure 3: Bode plot of G(s) from DSA and modified Eq.6

As can be seen from Figure 3, the Eq.6 and the DSA output do not match very well. In particular, the natural frequency and damping ratio of the complex poles do not match. This means that the published values in the gfshaft.m file may not be entirely accurate. A transfer function must be derived to match the DSA output in order to accurately identify the location of the actual complex poles. By looking at the DSA analysis, the highest frequency that is available to fit is 200 rad/s. Therefore, any terms in Eq.6 with break frequencies higher than 200 rad/s will not play a role in fitting the DSA data and can be neglected along with the pole at the origin. This yields a simplified expression for H(s) as shown below.

$$H(s) = \frac{\omega_2}{e_1} = \frac{3.11\times10^6}{27.69\left(\frac{s}{27.69}+1\right)14011.61\left(\frac{s^2}{14011.61}+\frac{s}{565.3}+1\right)} \qquad \textbf{(Eq.7)}$$

Writing Eq.7 in a general form yields

$$H(s) = \frac{K}{a\left(\frac{s}{a}+1\right)\omega_n{}^2\left(\frac{s^2}{\omega_n{}^2}+\frac{2\zeta s}{\omega_n}+1\right)} \qquad \textbf{\textit{(Eq.8)}}$$

As can be seen, H(s) is comprised of four components

1. A constant gain K
2. A pole at the origin with $\omega_{break} = a$
3. A pair of complex poles (ω_n and ξ)

These four variables can be systematically varied until the bode plot of H(s) matches up with the bode plot using least squares error analysis. The results of the analysis is shown below.

a = 21.11
ξ = 0.0778
K = 6.0556 x 10^5
ω_n = 62.0 rad/s

These values can now be substituted back into Eq.8. This yields a third order approximation of the system. The high frequency poles and zeros can be substituted back into the third order system in order to make the system sixth order once more. The bode plots of the DSA output, the third order fit, and the sixth order fit are shown below in Figure 4.

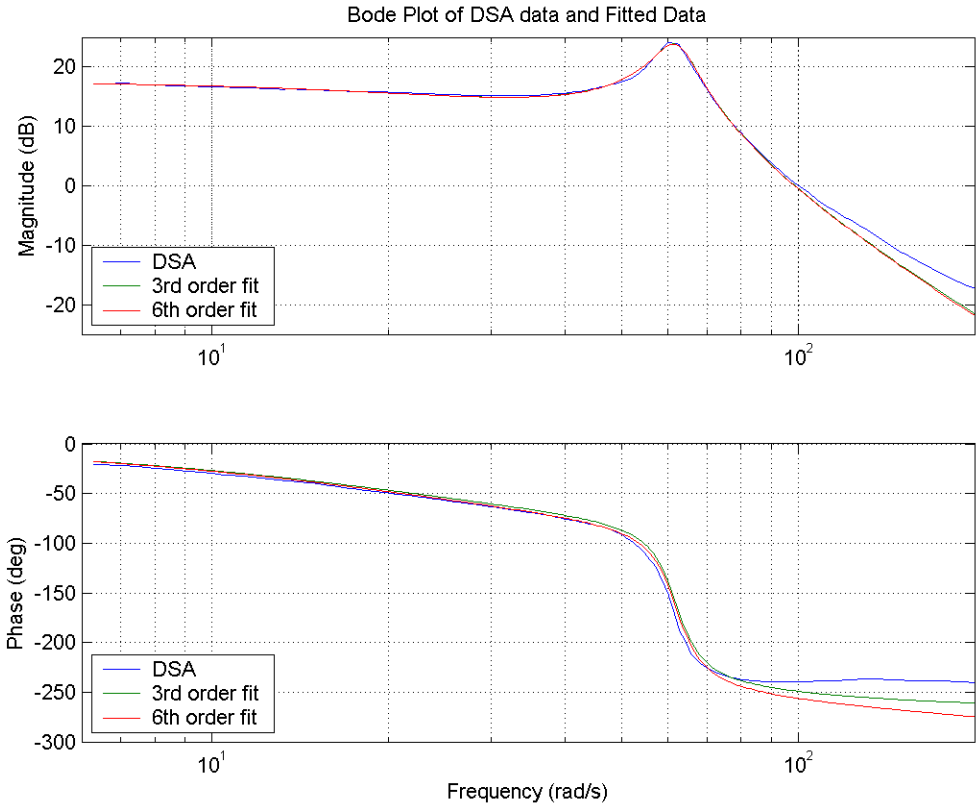


Figure 4: Bode plot of DSA and fitted data using least squares analysis

As can be seen, the fitted parameters match the output of the DSA very closely. Also notice that the addition of the high frequency poles did not change the plot significantly. Therefore it is possible to approximate the 6th order system with a simpler 3rd order system.

The pole/zero map of the fitted transfer function and the original transfer function zoomed in near the origin is shown below Figure 5.

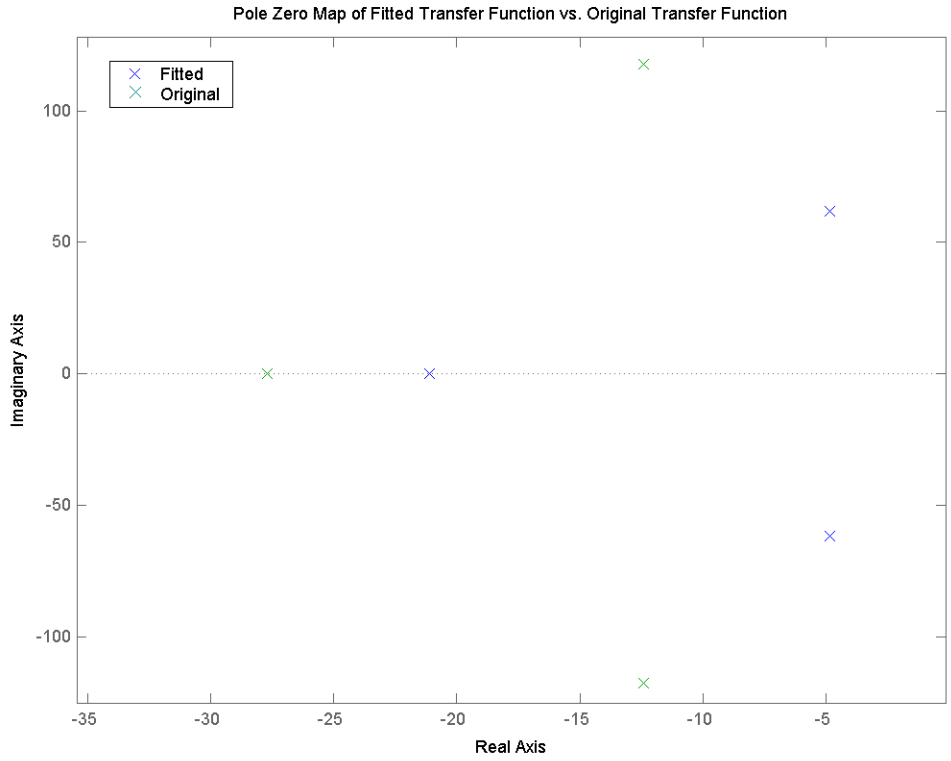


Figure 5: Fitted roots and original roots near the origin

As can be seen, the fitted transfer function has poles that are closer to the imaginary axis. The original transfer function from gfshaft was roughly accurate, but did not place the poles in their actual locations. The fitted transfer function of $G(s) = \theta_2/e_1$ is now given by

$$G(s) = \frac{0.008705s + 7.462}{1.783\times10^{-11}s^6 + 3.019\times10^{-8}s^5 + 1.331\times10^{-5}s^4 + 0.0005004s^3 + 0.05228s^2 + s}$$

(Eq.9)

Now that the locations of the poles and zeros of the plant are known, a compensator (PI controller and notch filter) can now be designed using root locus techniques. This is discussed later in Section IX: Control System Design.

V. Control System Design

The Simulink model used in lab to interface with the xpcTarget computer is shown below in Figure 2.

To recap, Part A required the development of an open-loop controller, Part B was a purely proportional position controller, Part C, a proportional-differential position controller, and Part D a proportional-integral-differential controller.

a. Open-Loop Control

The design goal of the open-loop controller was to provide position change of approximately one radian to the drive motor. This corresponds to a five-radian rotation where the position is measured. The scale difference is due to the encoder being mounted at the output disk. The output disc has a 5:1 gear ratio to the drive motor as previously discussed.

The desired rotation control was developed empirically by varying the duration and intensity of the control pulse. A pulse intensity of 2.3V with a duration 0.177sec was found to yield a position change of 240°, which is equivalent to 4.19 radians. This result was deemed satisfactory. Figure 7 shows the results from experiment and simulation with three runs used for repeatability.

One notices that the results shown below only appear to show one run. However, data from three runs are shown and overlap due to their consistency. This repeatability was ensured by manually positioning the drive motor in a consistent direction to a common starting point. This ensured that any non-symmetry in the drive motor was negligible and that the motor's hysteresis was consistent.

One notices that that the simulated results do not closely match the magnitude of the experimental results. This is largely due to the poor modeling of the drive motor's stiction effects. In simulation a dead-zone block of –0.2V to 0.2V was used to simulate this effect. This block attempts to model the stiction effect by simply cutting off any signal within it's dead-zone.

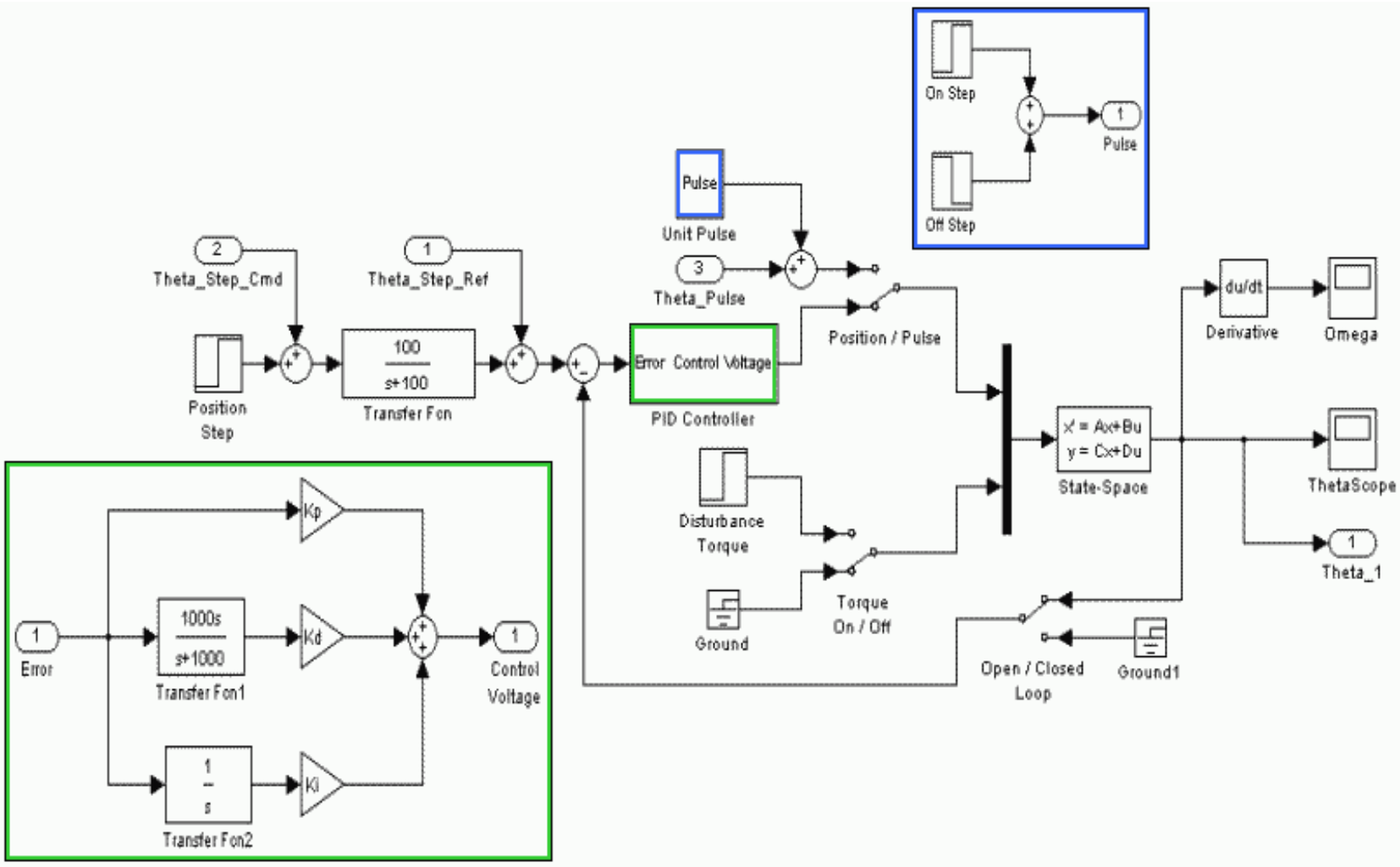


Figure 6 - Simulink model used for Open Loop, Proportional, Proportional-Derivative, and Proportional-Integral-Derivative Control

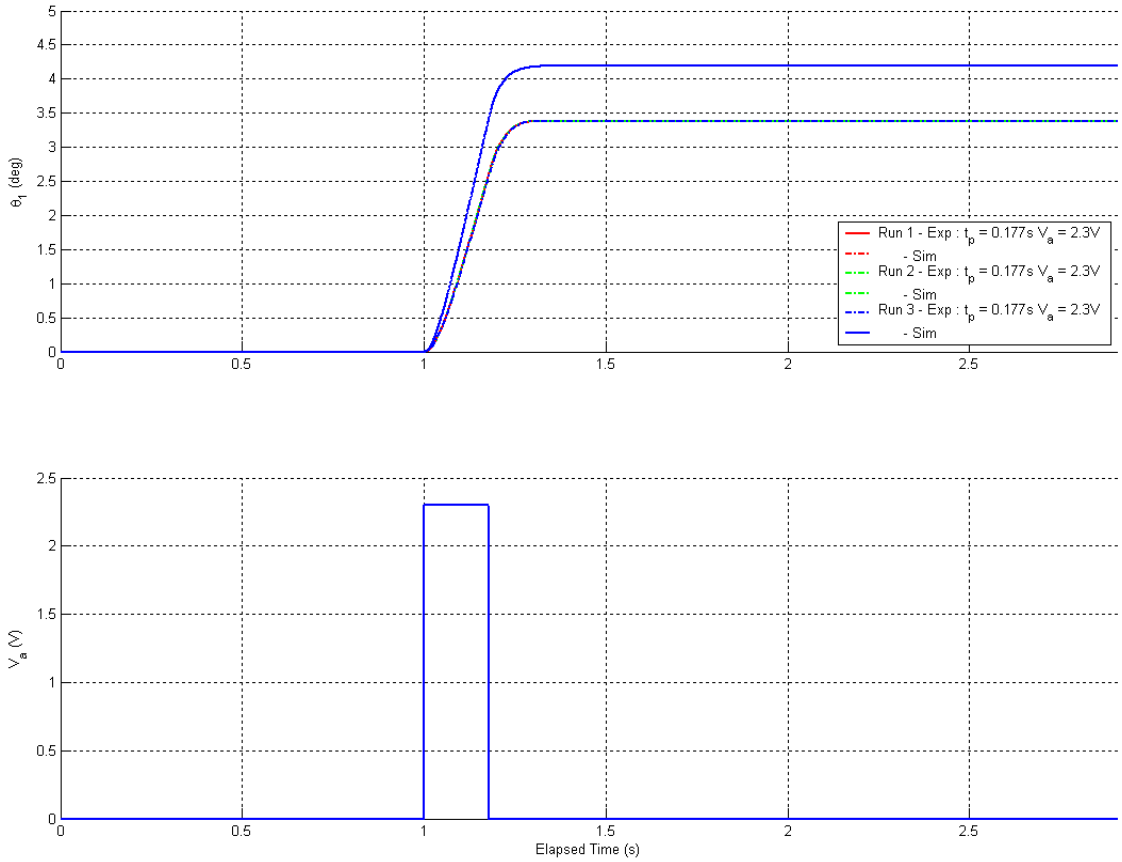


Figure 7 - Open loop pulse control response of θ_1 and V_a.

Development of this controller was done on-line through experimentation with the physical model. Table 3 lists the system parameters used in the development of this model

b. Position Control System: Proportional (P) Controller

In development K_P was varied until the desired results were obtained. A rough estimate for K_P was developed using the Simulink model shown in Figure 6. The results of varying K_P led to a final design value of 3.1 being chosen.

Table 3 – Proportional position control runs.

Run	Proportional Gain – K_P
1	3.5
2	3.2
3	3.0
4	3.1

Next, the proportional controller was redesigned to provide a steady-state error of 0.1 radians in response to a constant disturbance torque, T_l, of 1 N-m. Table 4 lists the proportional gains used in developing this controller with Figure 9 showing the error in the system's response.

Table 4 – Proportional position control runs under constant disturbance torque.

Run	Proportional Gain – K_P
6	15
7	16

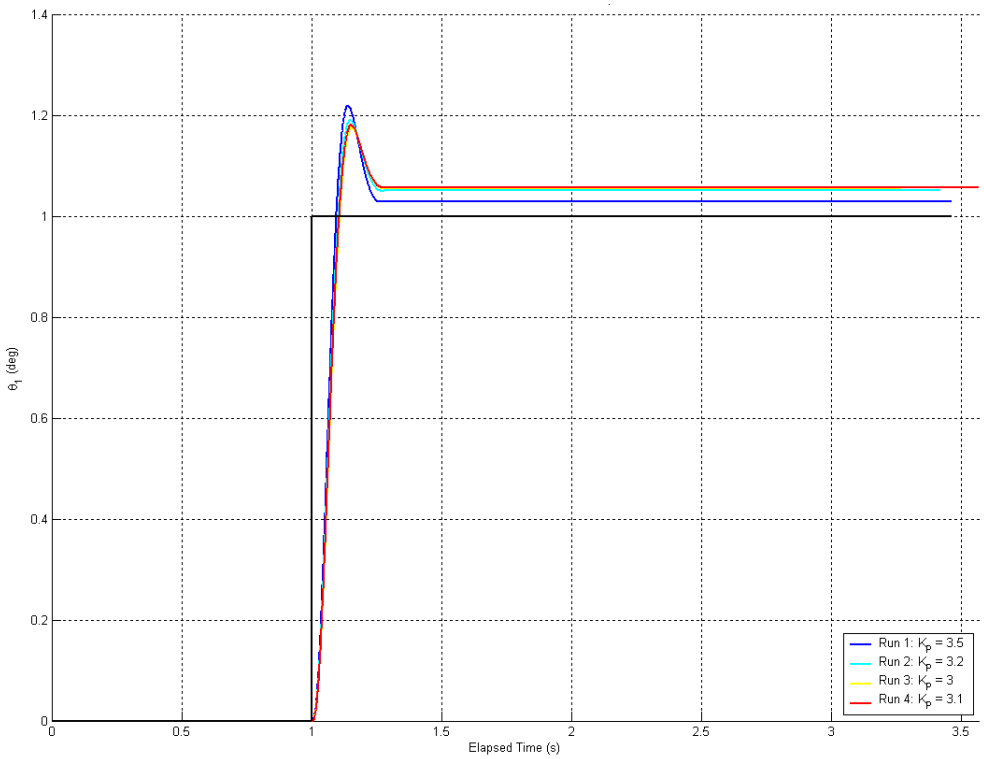


Figure 8 – Proportional controller step response with variations in K_P.

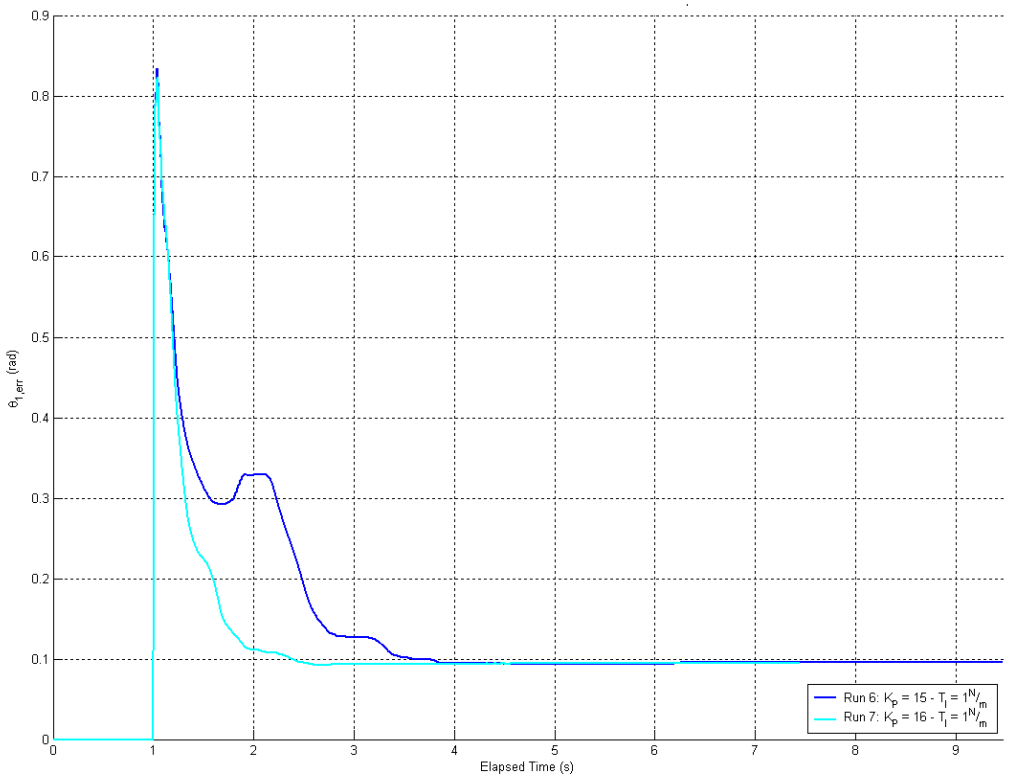


Figure 9 - Error in proportional position controller with varying K_P under disturbance torque.

c. Proportional-Differential Position Control

Using the same system as was previously used, the controller was modified to a proportional-differential (PD) controller. Additionally, the design requirements were tightened to allow for less than 5% overshoot and the ability to produce a steady-state position error of less than 0.1 radians under a constant disturbance torque of 1 N-m.

In this development of this controller a constant pseudo-derivative frequency of 100 rad/s was chosen and kept fixed. The controller was then tuned by varying K_P and K_D until the desired results were obtained. These effects of these variations are plotted Figure 10 in with the corresponding parameters for each run listed in Table 5.

Table 5 – Proportional-Differential position control runs under constant disturbance torque.

Run	Proportional Gain K_P	Differential Gain K_D	Applied Torque T_L
1	7	0.1	0 N-m
2	7	0.1	0 N-m
3	12	0.2	0 N-m
4	12	0.2	1 N-m

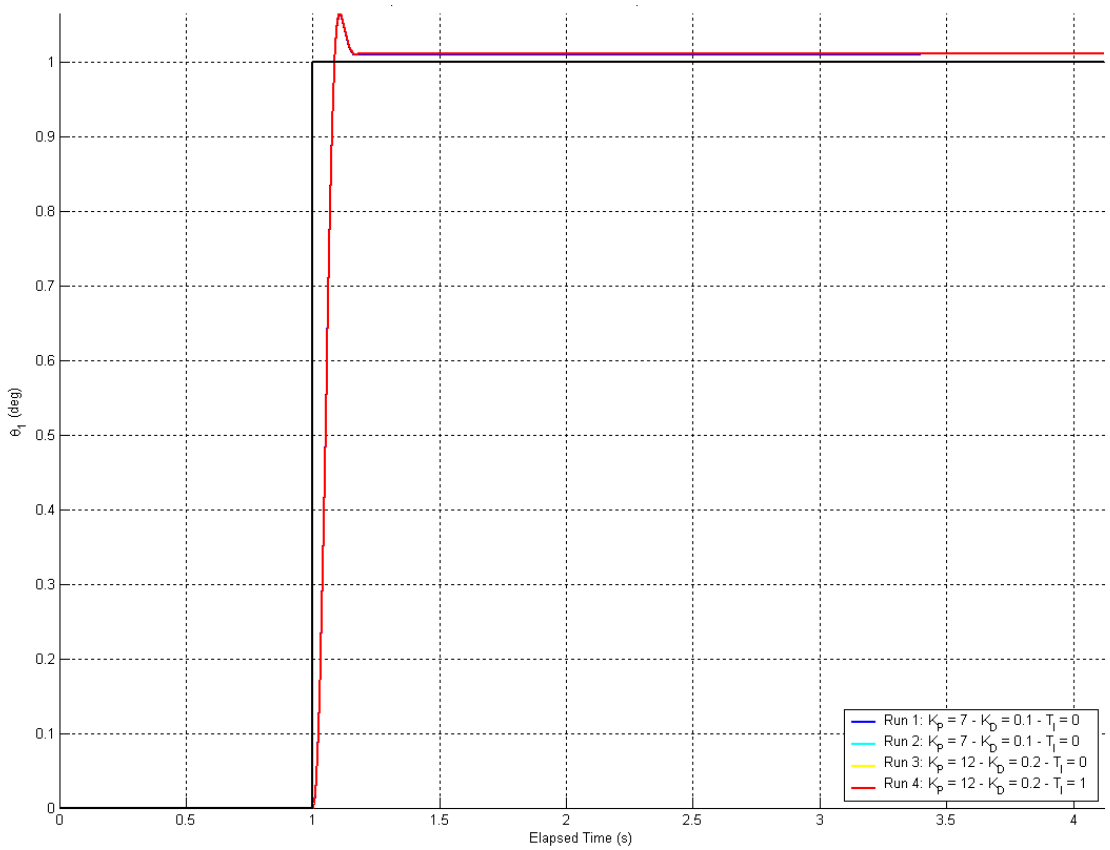


Figure 10 - Response of proportional-differential controller with variations in K_P K_D, and T_I.

d. Proportional-Integral-Differential Position Control

In this phase of the experiment, the controller was again modified, this time to include an integral function to the controller. This change makes the controller a PID controller. Requirements for the controller's performance were left untouched at less than 5% overshoot, and less than 0.1 radians of steady-state error when given a disturbance torque of 1 N-m.

In designing this controller, the proportional and differential gains used previously were kept constant. Fortunately, the initial integral gain tested yielded quite suitable results. These results were deemed desirable enough that further variation was not explored. This decision was also based on experience with PID controllers gained in previous experimentation. Table 6 lists the parameters used in the controller for each of the PID control runs and Figure 11 shows the resultant system response.

Table 6 – PID position control runs.

Run	K_P	K_D	K_I	T_L
1	12	0.2	3	1 N-m
2	12	0.2	3	0 N-m

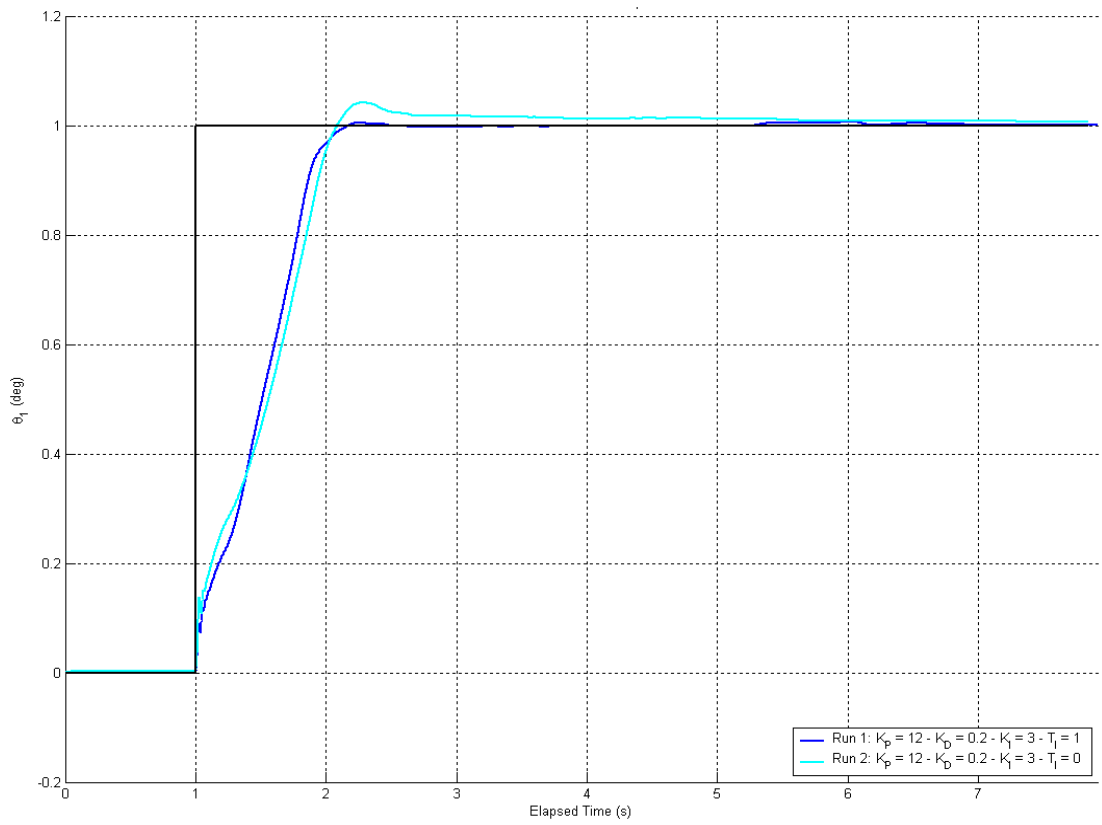


Figure 11 - PID position controller response to step input with and without disturbance torque.

e. Notch Filter Design

For part E, the goal was to excite the flexible modes of the shaft with and without a notch filter and see the effects of adding the filter in an effort to increase damping. Initially, a simple feedback scheme was used in an effort to excite the flexible mode. The feedback loop is shown below.

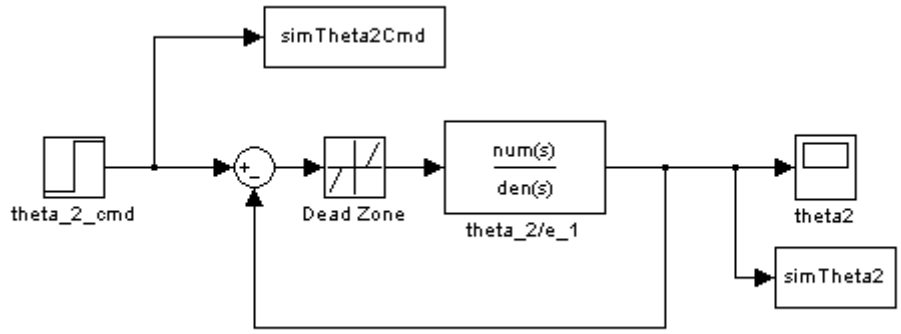


Figure 12: Simple feedback loop with deadzone of +/- 0.2 volts

The simulated and actual results for this system are shown below in Figure 13.

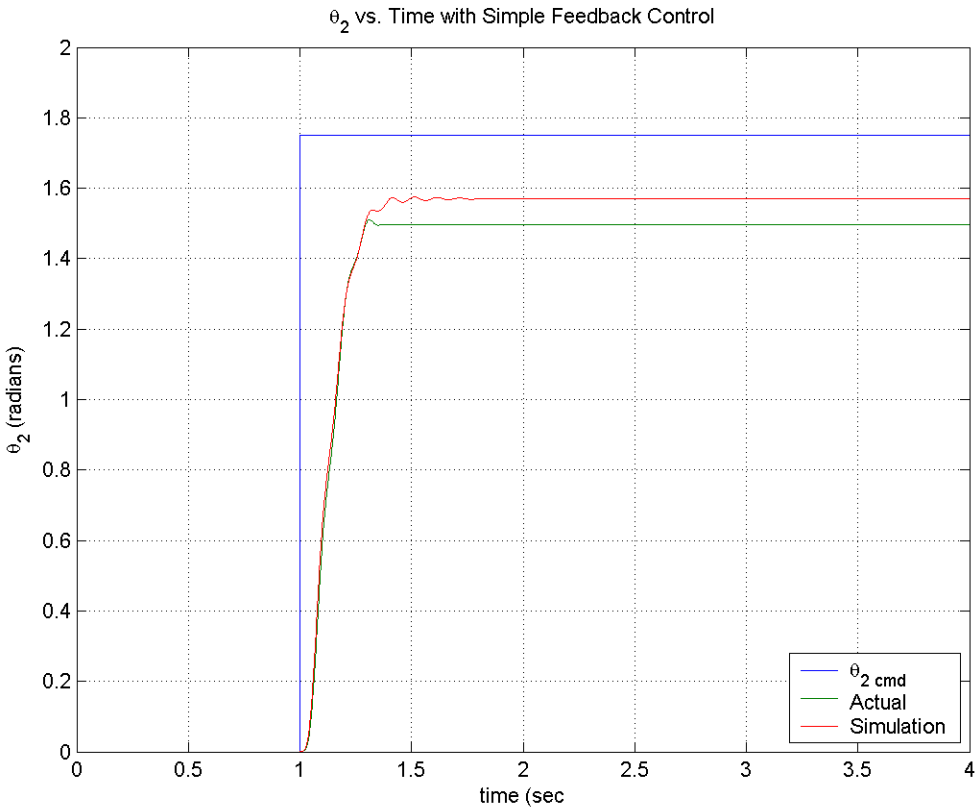


Figure 13: Simulated and actual response with simple feedback

As can be seen in Figure 13, the response of system is very sluggish with a steady state error. Both simulation and actual data show that the flexible modes are not excited since there are very small oscillations. A notch filter could be designed to damp out the very small oscillations but its effect would not be very noticeable. In an effort to increase the bandwidth of the system and increase the amplitude and number of oscillations, a PI controller is designed as a compensator.

The PI controller used has $K_P = 6$ and $K_I = 2$. The block diagram of the system is shown below in Figure 14.

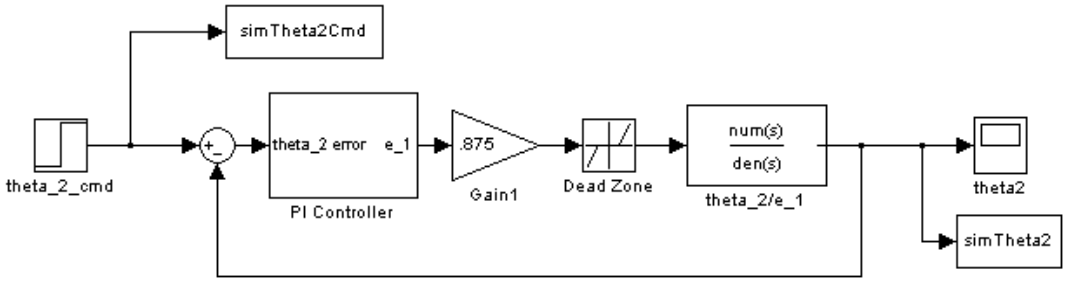


Figure 14: Block diagram with PI controller and gain

A constant gain of 0.875 is included after the PI controller for reasons that are detailed in section VII: Modeling Validation. The response of the simulation and actual lab data are shown below in

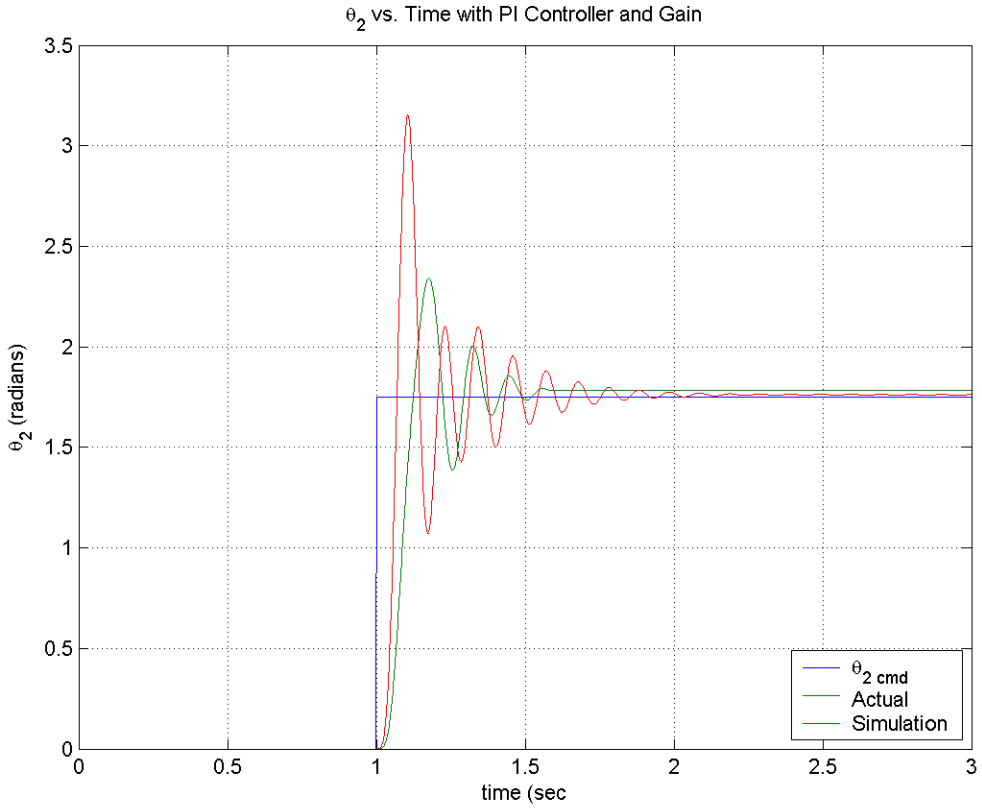


Figure 15: Response vs. time with PI controller and gain

As can be seen by comparing Figure 15 with Figure 13, it is seen that the addition of the PI controller increases the system bandwidth and also excites the

flexible modes. This is evident from the large amplitude and number of oscillations. These oscillations are actually desirable for Section E. This will make the presence of the notch filter more evident when it is incorporated. The root locus for the block diagram shown above in Figure 14 is shown below in Figure 16.

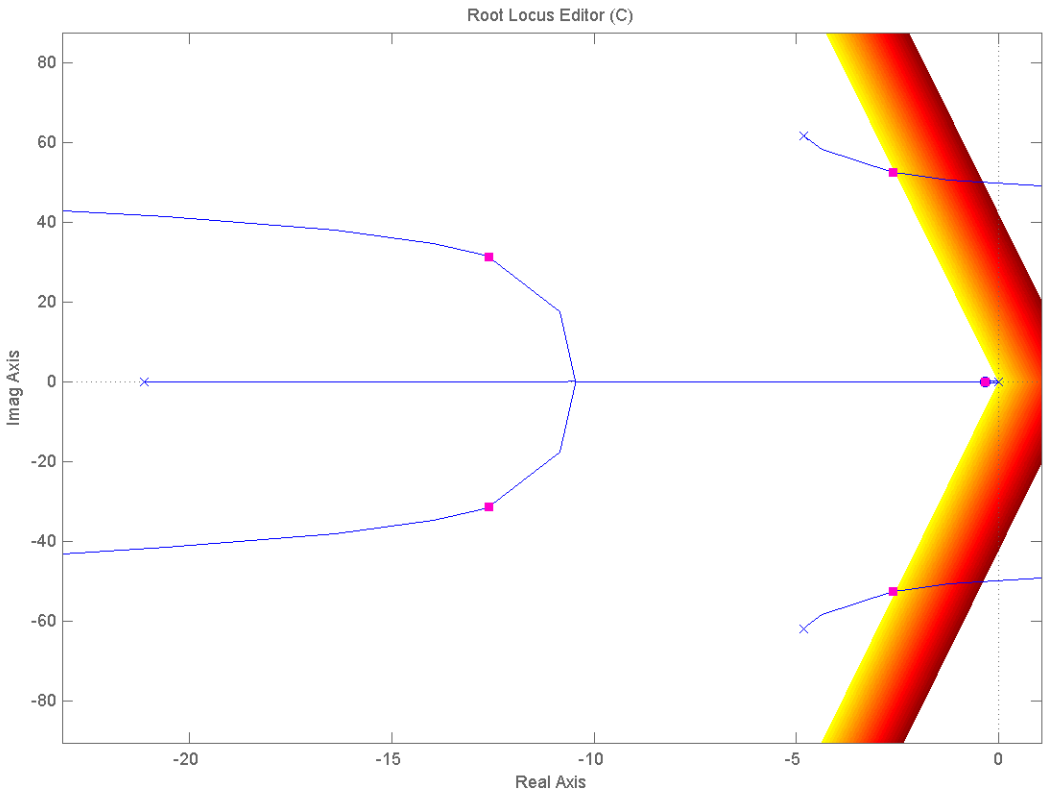


Figure 16: Root locus of system with PI controller and gain

As can be seen above, the flexible modes are the pair of complex poles which move towards the imaginary axis with increasing gain. These are the limiting poles which prevent the damping from being increased. The maximum damping coefficient line is shown as the yellow/orange line. The maximum damping coefficient possible with this control architecture is

$$\zeta = 0.0495$$

In order to improve the damping and performance of the system, a notch filter can be added to the system. This is a pair of complex poles and zeros which serve to locally move the complex poles farther away from the imaginary axis while leaving the rest of the root locus relatively unchanged. The location of the complex poles and zeros are placed near the complex poles which represent the

flexible modes. This is why it was so important to obtain an accurate transfer function between θ_1 and e_1 so that the complex poles could be accurately located and the notch filter could be effectively designed. The notch filter was designed using the SISOtool. A pair of complex poles and zeros can be placed near the flexible mode complex poles. Their location is then varied until a desirable root locus (improved damping) is achieved. The best simulated notch filter is given by

s = -14.2857 ± 69.98j (zeros)
s = -20.4444 ± 63.4545j (poles)

This notch filter can be introduced in parallel with the PI controller. The final block diagram architecture is shown below.

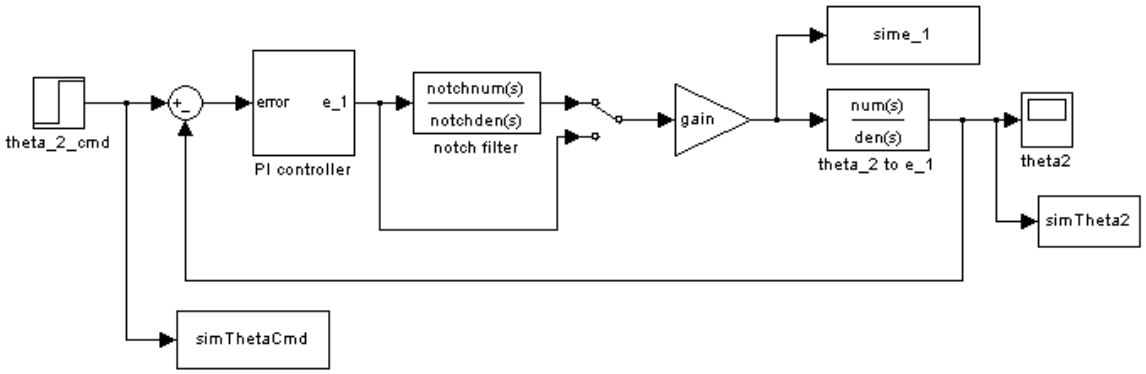


Figure 17: Block diagram of system with PI controller and notch filter

The roots locus for the system with the PI controller and the notch filter is shown below.

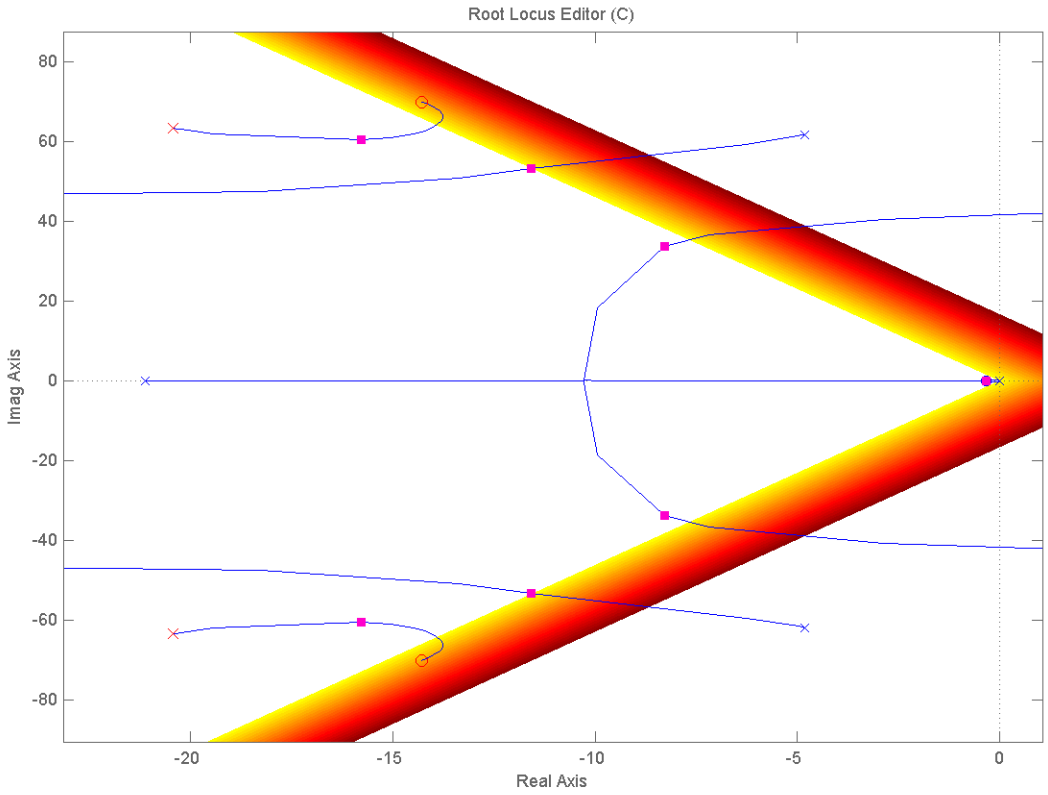


Figure 18: Root locus of system with PI controller, gain, and notch filter

As can be seen from Figure 18, the damping has greatly increased with the addition of the notch filter. The maximum damping ratio possible with this system is given by

$$\xi = \mathbf{0.2120}$$

This is a significant improvement simple PI controller, increasing the damping ratio over 4 times. The constant gain of 0.875 shown in Figure 16 is required to extract the most performance out of the system. As seen above in Figure 18, there are actually 4 poles on the line of maximum damping ratio. Two of these are the complex poles that represent the flexible mode of the shaft. As the gain K is increased, these poles move farther away from the imaginary axis. However, there are also a pair of complex poles that originate from the pole on the real axis and one of the poles at the origin. These poles leave the real axis at roughly $\sigma = -10$. As the gain K is increased, these poles move towards the imaginary axis. Therefore, a gain of K = 0.875 is the best compromise between the two.

VI. Experimental Procedure

This Lab assignment was split up into 4 separate sections. First was Part A: Open-Loop control, then Part B: Position Control with a proportional controller, Part C: Proportional Derivative Controller, and finally, Part D Proportional Derivative Integral Controller.

The purpose of Part A was to simply model the system's open-loop control. This was done by first connecting the load motor encoder to Channel 1 of the encoder board and the motor drive encoder to Channel 2 of the encoder board. The Kepco amplifier gain was then confirmed to be approximately Ka1=3.6V/V. A Simulink model was then built to interface with the Quadrature Encoder board. This interface can be seen in Figure 19 below.

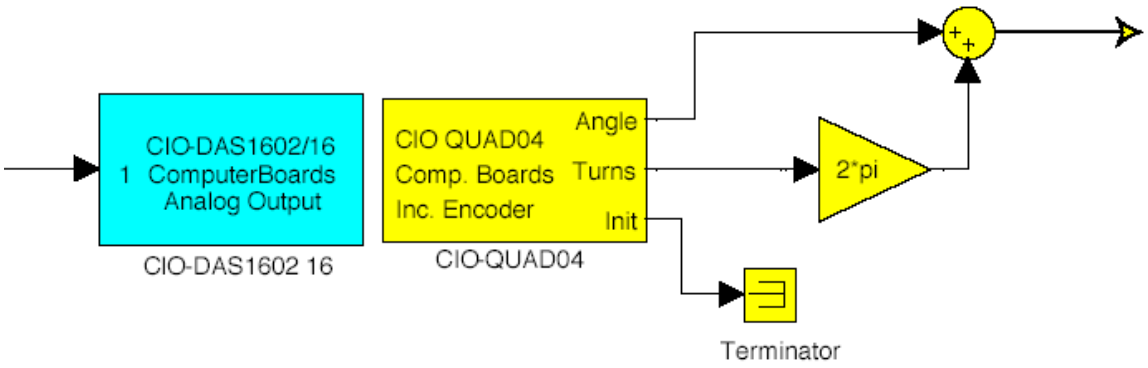


Figure 19 – Simulink blocksets for Analog Output and Encoder Input.

We then connected the encoder sensor of the motor drive to Channel 2 and the Analog output of the Multifunction board for the control voltage, $e_1(t)$, to the Kepco amplifier input. The analog output from the multifunction board was also connected to the proportional amplifier input of the Kepco power supply.

To produce the transient response of the system, a voltage was sent to the motor drive, $e_1(t)$, in the form of a pulse such that the motor rotated approximately 1 radian. This was done using 2 step inputs applied at different start times. The response of the encoder was then saved for future use.

For Part B the task was to design a proportional linear feedback control system to track a position reference command. This was done with by first creating a Simulink model with a proportional feedback control of position error. This model can be seen below in Figure 20

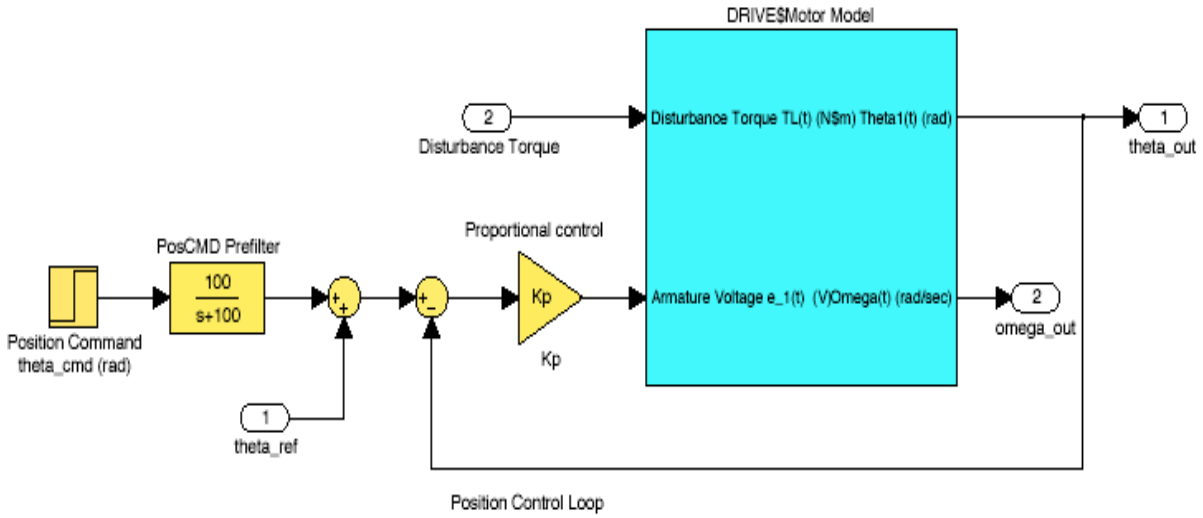


Figure 20 – Simulink blockset for proportional position feedback controller.

To design the proportional gain K_p a trial and error approach was used. The design constraints for the proportional controller were good tracking with less than 20% overshoot for a step input of 1 radian, fast rise and settling time, and good disturbance rejection with a small settling time to a disturbance input. To assure the last design constraint was met, a constant 1 N-m torque was applied to the rotor disk with a torque wrench. From this, the steady-state position error was recorded and evaluated to determine how well the controller performed. We then attempted to find how the steady-state error due to the disturbance torque depended on K_p. To do this we experimentally found the disturbance torque, in N-m, which caused a 0.1 radian position error. We also decided that adding a derivative controller would be best suited to reduce overshoot and provide better disturbance rejection.

For Part C the goals were to design a Proportional/Derivative controller which could provide less than 5% overshoot, and a steady-state position error of 0.1 radians for a constant disturbance torque of 1 N-m. In order to design this controller we utilize a pseudo-derivative instead of a pure derivative. This was due to the fact that a pure derivative with an encoder with a finite resolution would result in unnecessary noise in the feedback control. The Simulink model of this controller can be seen below in Figure 21.

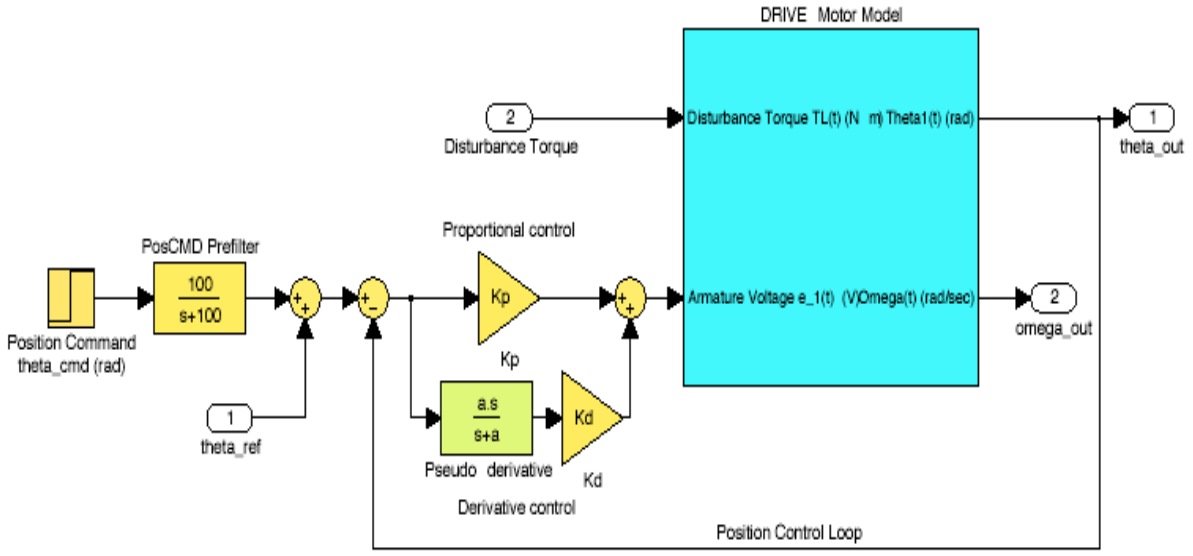


Figure 21 – Proportional and pseudo-Derivative controller for motor position

This controller was then evaluated on is performance in terms of steady-state error to a step command of 1 radian, rise and settling time to a step command of 1 radian, and position error due to a step disturbance torque of 2 Nm.

For Part D we designed a Proportional Integral Derivative controller. The design constraints were the same as in Part C, less than 5% overshoot and 0.1 radian steady-state error for a disturbance torque of 1 N-m. The Simulink model of the controller can be seen below in Figure 22.

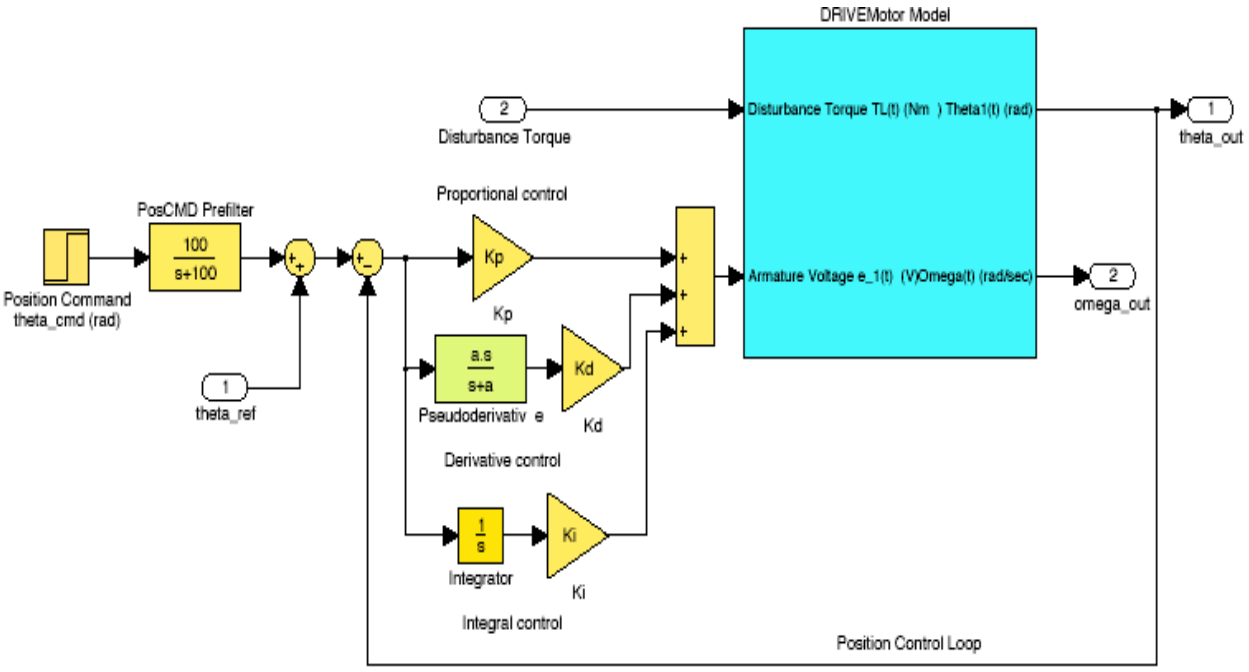


Figure 22 – Proportional, Integral, and Derivative Controller for motor position

VII. Modeling Validation

Part A: Open-Loop Control

As stated previously in the Experimental Procedure section, a pulse was applied to the motor and the output θ_1 was recorded. The Simulink model used is shown below in Figure 23.

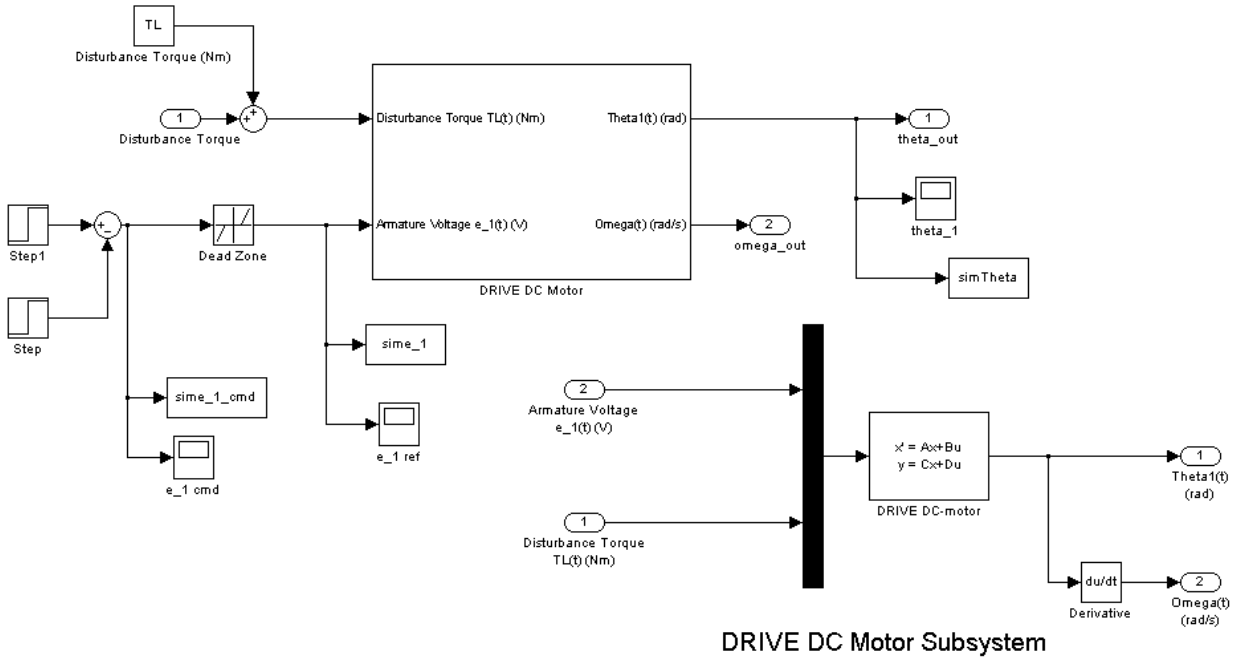


Figure 23: Simulink model used to model open loop control

The DRIVE DC-motor state space model is given by Eq.1 and 2. The simulated results are shown with the data gathered in lab is shown below in Figure 24 when a pulse of amplitude = 2.3 V and of duration t = 0.177 seconds is applied.

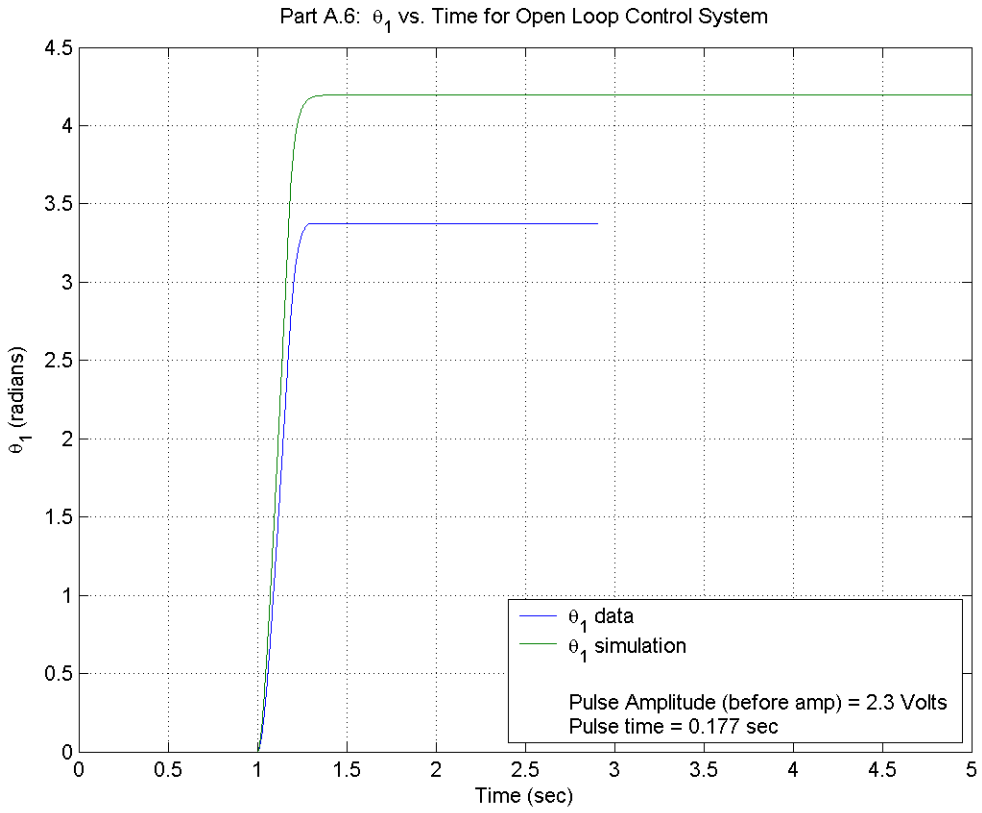


Figure 24: θ_1 vs. Time for open loop control system

As can be seen, the lab results and the simulation are similar, but not exactly equal. It appears that the simulation is not as stiff a model. This could be that the actual motor has more stiction. Increasing the dead zone could possibly make the simulation match closer to the actual data.

Part B: Position Control System: Proportional (P) Controller

As stated in the lab handout, a model that does not contain nonlinearities is constructed and shown below in

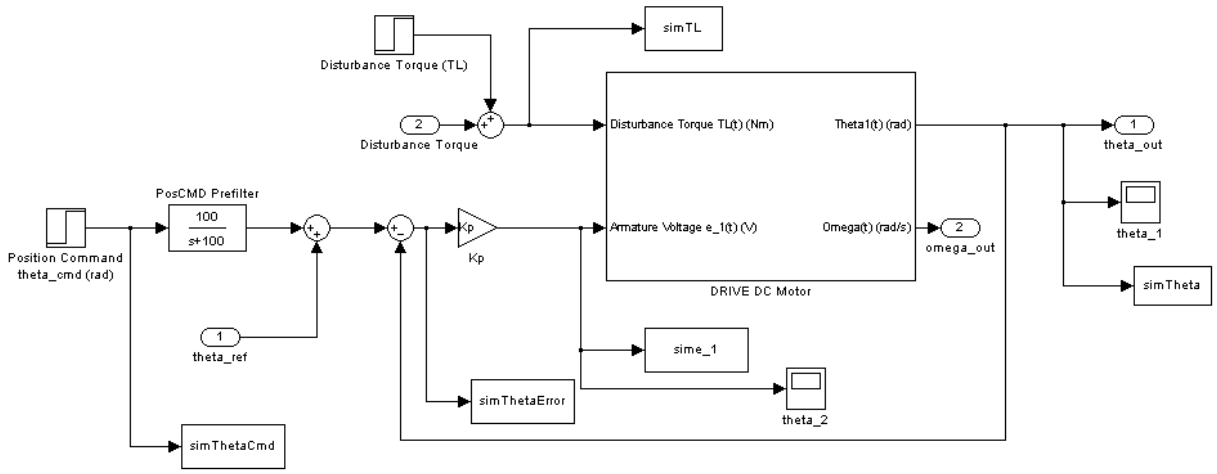


Figure 25: Simulink model with feedback and simple proportional gain

The parameter K_P is varied to find the gain at which 20 percent overshoot is achieved. In simulation, this critical gain if found to be 3.1. The simulation and actual lab results with $K_P = 3.1$ is shown below in Figure 26.

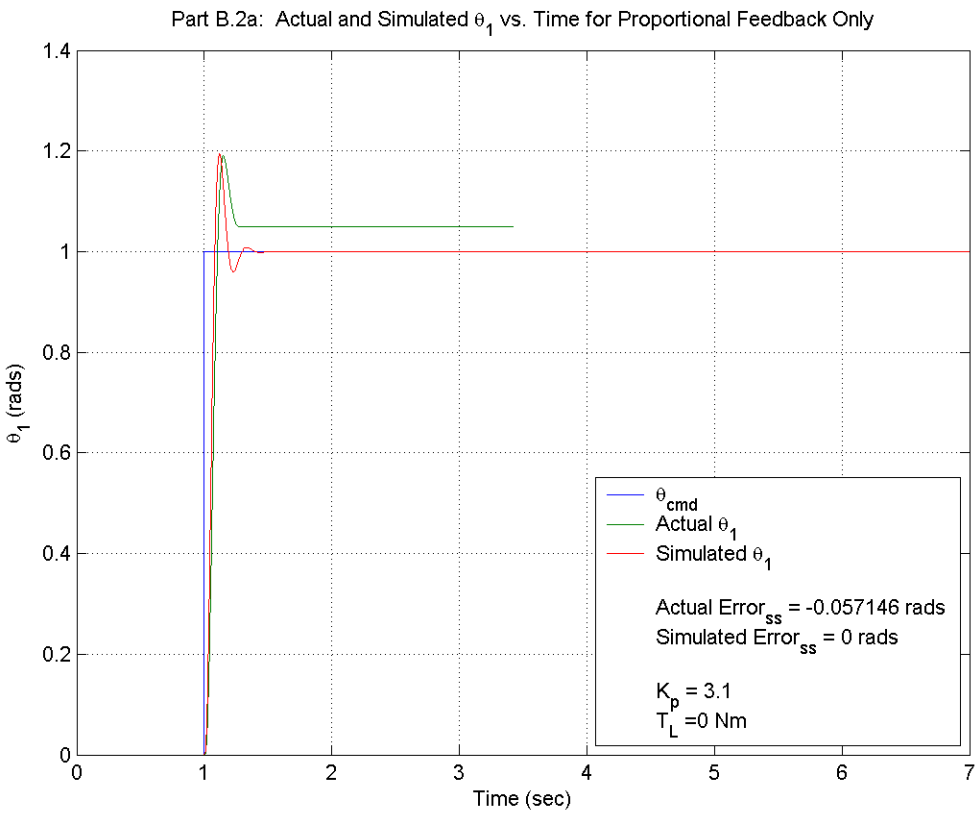


Figure 26: Actual and simulated θ_1 vs. time for proportional feedback only

As can be seen above, the simulation and the actual data acquired in the lab with a K_P = 3.1 both produce almost exactly 20 percent overshoot. Furthermore, the rise time and settling time are acceptable, so the design satisfies criterion B.2.a and B.2.b from the lab sheet. The only difference between simulation and reality is the steady state error. In the lab, the presence of stiction prevented the error from going to zero. However, since the model intentionally did not include any non-linear effects such as stiction, there is no steady state error.

The design's ability to reject a disturbance can be simulated and plotted against lab data as shown below in Figure 27. It was discovered that the motor was not able to reject a constant disturbance of T_L = -2 Nm, so -1 Nm was used instead. The sign convention is negative since the torque was applied in a direction opposing the motor rotation.

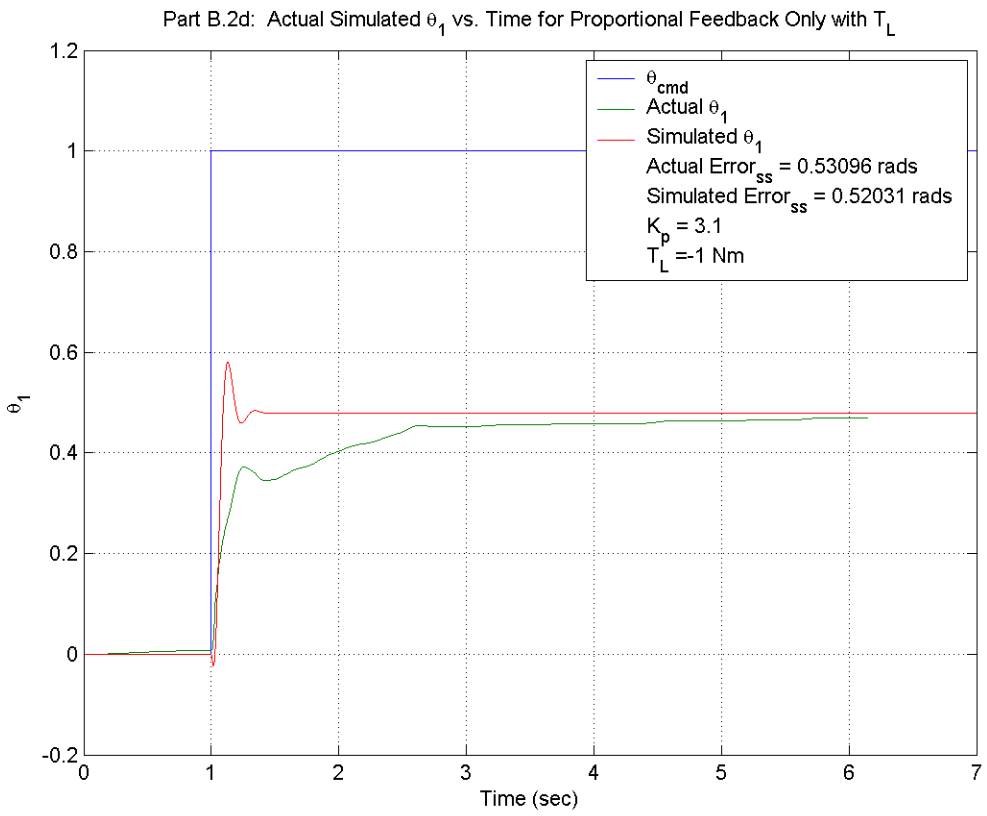


Figure 27: Actual and simulated vs. time for proportional feedback only with constant torque

As can be seen above, at steady state, the simulation agrees very well with the actual lab data. The simulation accurately predicts the steady state error under a constant torque. The simulation differs from the actual data initially simply because it was very difficult to apply a constant torque when the step is first introduced. The motor initially began to move and the torque-applier's reaction was not fast enough to respond to the motor. Therefore, initially, the torque applied in lab is variable. Only at steady state was a constant torque able to be applied.

As shown above, the steady state error is a function of the disturbance torque applied. It is also a function of the gain K_P. The transfer function $P(s) = \theta_1/T_L$ can be derived in order to find this relationship. The state space representation of the motor is given in Eq.1 and Eq.2. Since the prefilter is not in the command path between θ_1 and T_L, its state space representation is not needed. The transfer function P(s) can be calculated using Eq.5. This yields,

$$P(s) = \frac{\theta_1}{T_L} = \frac{L_1 s + R_{A1} + R_1}{\alpha_3 s^3 + \alpha_2 s^2 + \alpha_1 s + a_o} \qquad \textbf{(Eq.10)}^3$$

where

$$\alpha_3 = L_1\left(J_1 + J_{m1} n^2\right)$$

$$\alpha_2 = \left[D_1 L_1 + D_{m1} L_1 n^2 + \left(J_1 + J_{m1} n^2\right) R_1 + \left(J_1 + J_{m1} n^2\right) R_{A1}\right]$$

$$\alpha_1 = \left[K_1 K_{A1} K_P n + \left(K_1 L_1 n^2 + D_1 R_1 + D_{m1} n^2 R_1 + D_1 R_{A1} + D_{m1} n^2 R_{A1}\right)\right]$$

$$\alpha_o = K_1 K_{A1} K_P n$$

The steady state error can now be calculated. Since θ_{1cmd} is equal to zero, then the error is θ_1. The error to a constant torque is given by

$$E(s) = P(s)\frac{T_L}{s} \qquad \textbf{(Eq.11)}$$

The final value theorem can now be applied (assuming all poles are in LHP) to find the relationship between the steady state error, K_P, and T_L can now be found.

$$\lim_{t\to\infty} E(t) = \lim_{s\to 0} sE(s) \qquad \textbf{(Eq.12)}$$

Substituting Eq.11 into Eq.12 yields

$$E_{steadystate} = \frac{\left(R_{A1} + R_1\right)}{K_1 K_{A1} n}\frac{T_L}{K_P} \qquad \textbf{(Eq.13)}$$

As can be seen, the steady state error is directly proportional to the applied torque and inversely proportional to K_P. Holding T_L constant, the gain K_P can be varied in simulation in order to illustrate this phenomenon. The results are shown below in Figure 28.

[3] See mathematica file section_b_5.nb for more information and derivation.

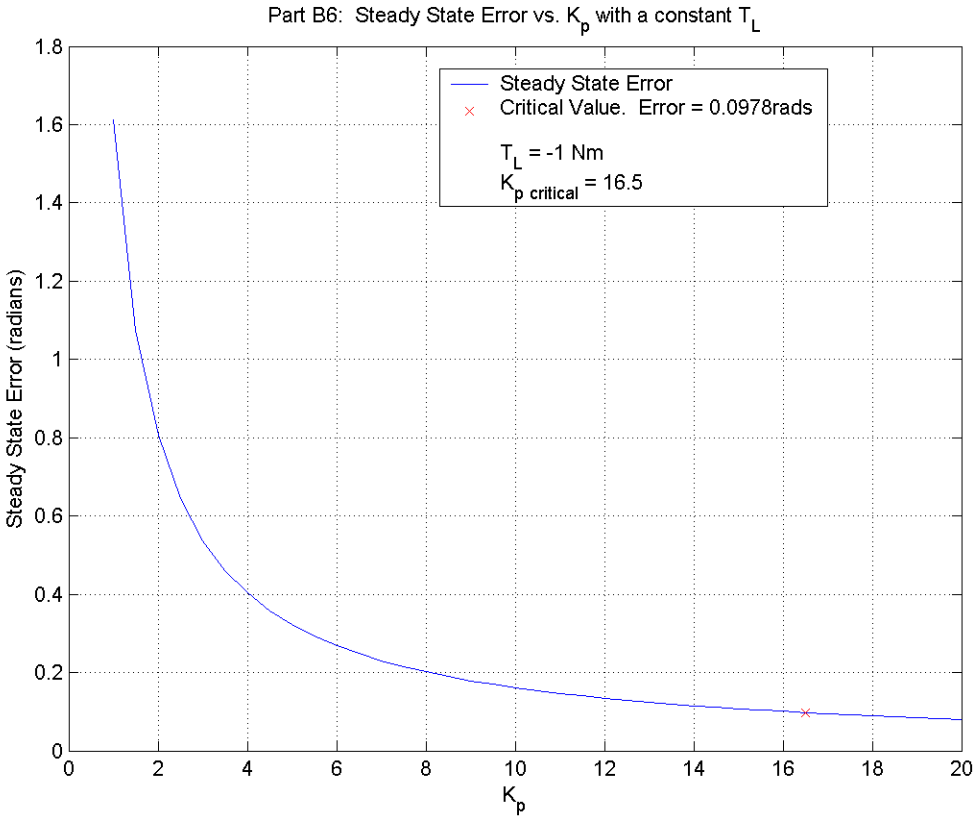


Figure 28: Steady state error vs. K_P with constant T_L

As can be seen above, the steady state error is indeed inversely proportional to K_P.

Assuming a constant torque of -1 Nm, Eq.13 can be solved for the K_P that results in $E_{steady\ state}$ equal to 0.1 radians. The critical K_P is found to be,

K_P = 16.13

The model simulation with $K_P = 16$ is shown below plotted with actual laboratory data.

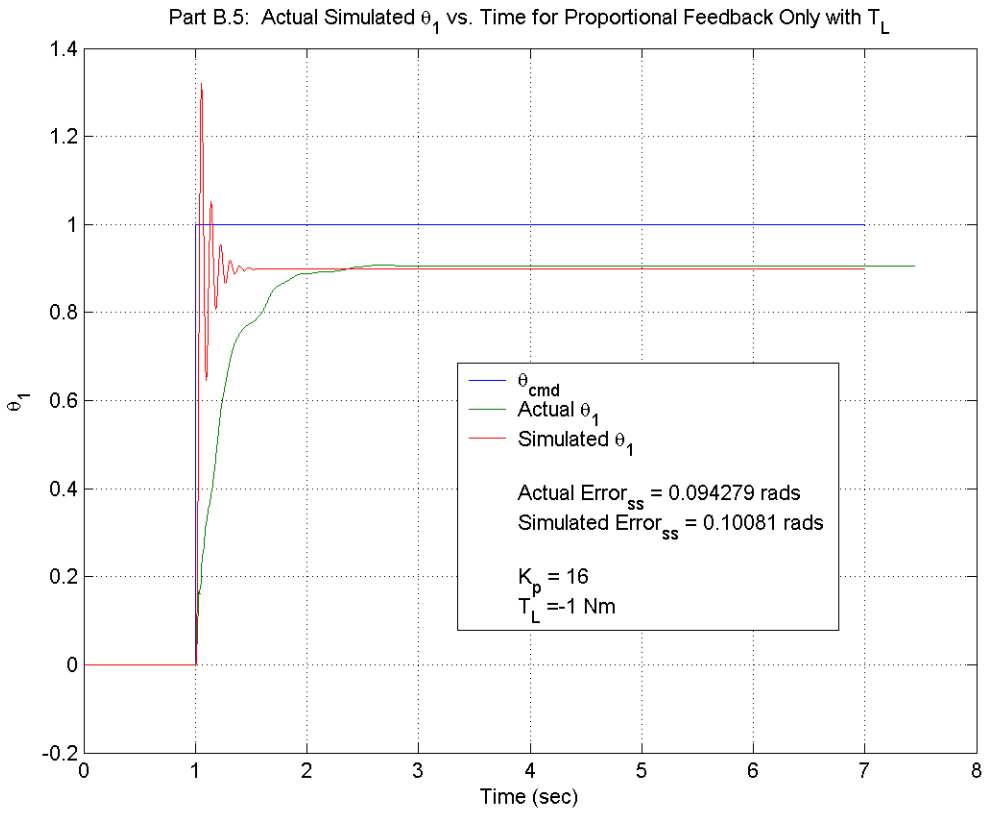


Figure 29: Actual and Simulated θ_1 vs. time with proportional feedback only with constant T_L

As can be seen, the simulation and actual data agree remarkably well. The simulation predicts the steady state error remarkably well. Once again, the discrepancy initially is due to the fact a human operator is not able to apply a constant torque initially.

It is known that the worst disturbance torque action on the system is up to 2 Nm. It was already found that the system cannot reject a disturbance of 2 Nm. In order to reject a 2 Nm constant torque, the K_P must be 32.2. If this gain is used, the overshoot is very high. The simulated response with this gain is shown below in Figure 30.

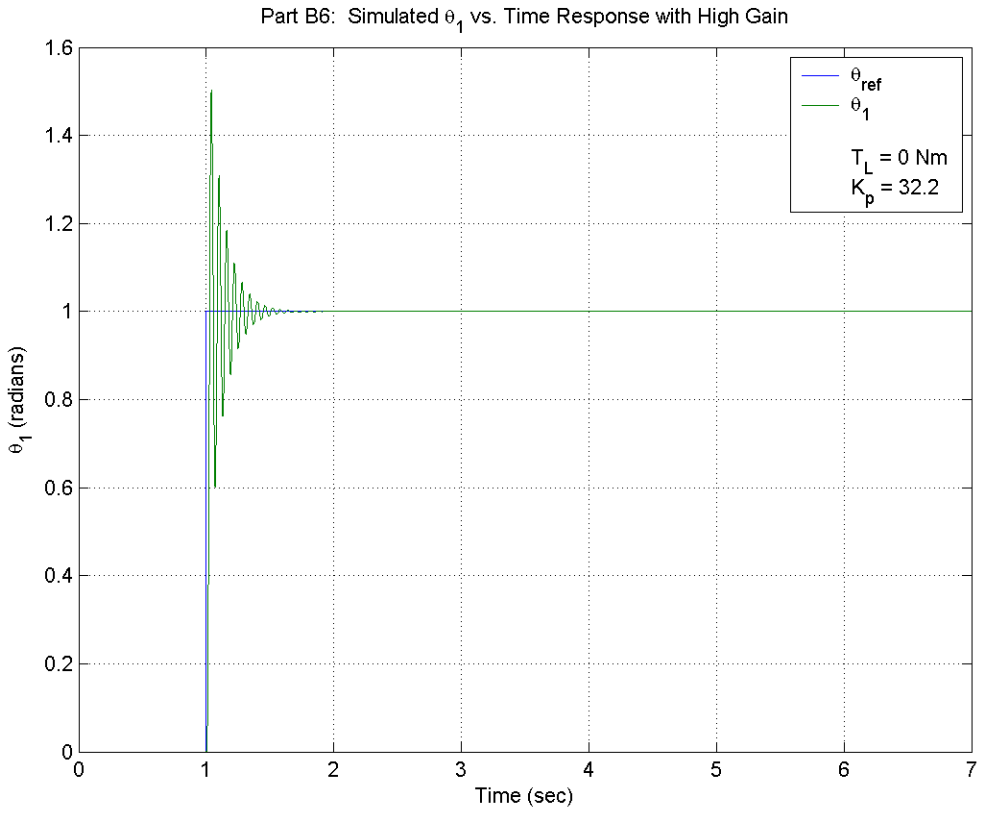


Figure 30: Simulated θ_1 vs. time response to $K_P = 32.2$

As can be seen, this high gain yields extremely high overshoot, almost 50%. However, as shown in Eq.13, the steady state error is inversely proportional to the gain K_P. Therefore, it is desirable to increase the gain as much as possible. However, from the analysis here, it is shown the percent overshoot increases with K_P. Therefore it may be necessary to include a derivative term as well. This would serve to reduce the percent overshoot but at the same time, allow for a high value of K_P and therefore also reduce steady state error.

For the remainder of the lab, it is assumed that the worst torque the system will experience is 1 Nm and all parameters with this value in mind.

Another parameter that must be monitored during the design of the control system is the motor input voltage e_1. In practice, this cannot exceed 10 volts since this is the limitation of the Labview output board. The input voltage e_1 corresponding to $K_P = 3.1$ and $T_L = 0$ (Figure 26, section B.2) is shown below.

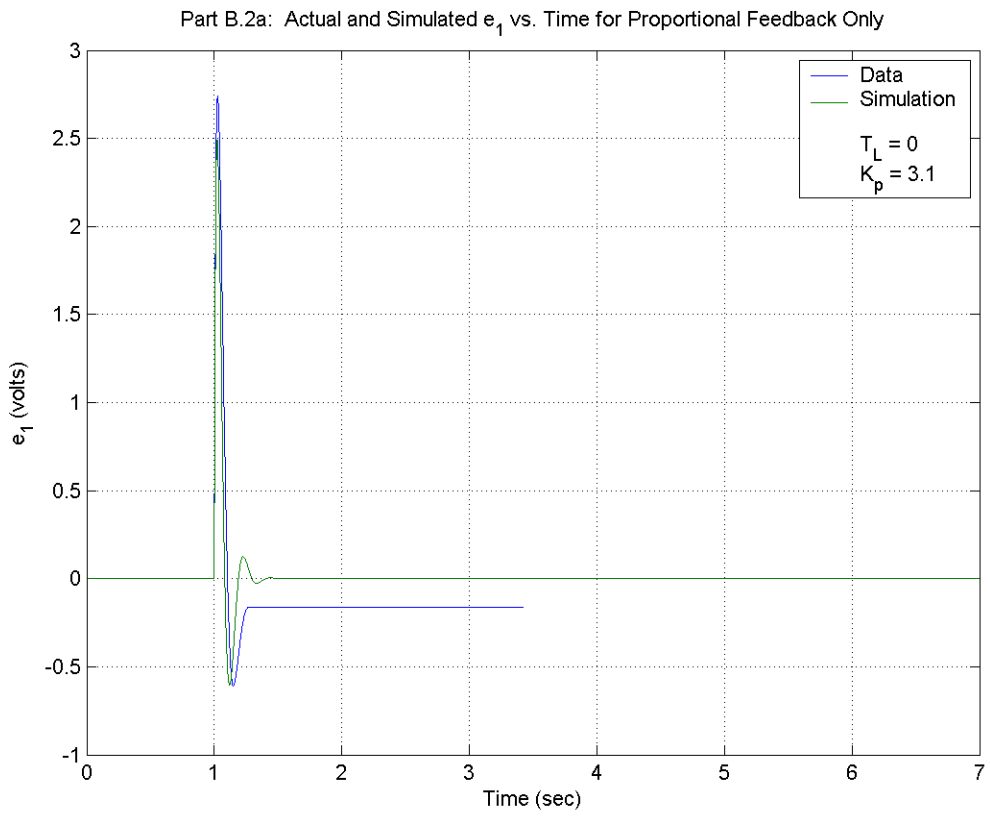


Figure 31: Actual and Simulated e_1 vs. time for proportional feedback only

As can be seen, the simulation and actual data match up well. The only difference is with the steady state error. Since the model does not account for stiction, there is no steady state error and thus, there is no steady state voltage. However, in the actual lab, the stiction produces a steady state error and since there is only a proportional controller, there is a steady state input voltage. During the entire run, the voltage remains between +/- 10 volts.

The voltage with a constant torque of T_L = -1 Nm and a K_P = 16 (Figure 27) is shown below.

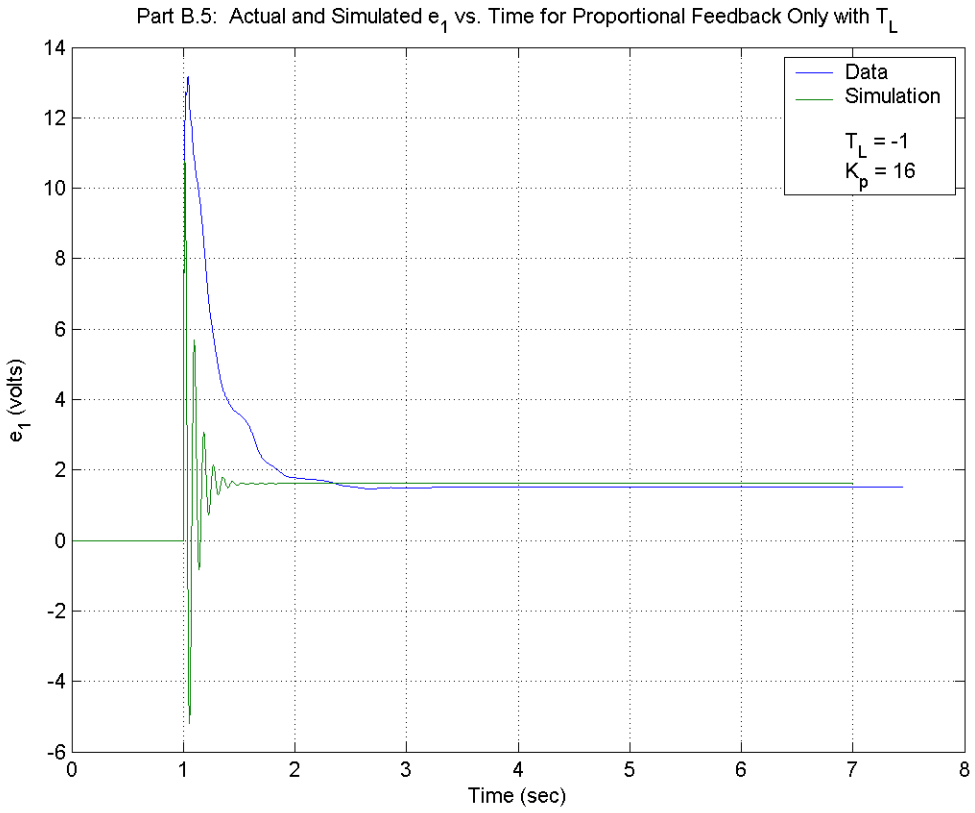


Figure 32: Actual and simulated e_1 vs. time for proportional feedback only with constant T_L

In this situation, the actual and simulated voltage is higher than 10 volts. This is a problem since this means the system saturates. The actual data appears higher than 10 volts because the actual voltage out was not measured, but rather the signal going to the Labview board was recorded. This signal is allowed to be greater than 10 volts, but in actuality, the voltage supplied to the Kepco amplifier is only 10 volts.

This is a problem if the voltage exceeds 10 volts. This means that the Simulink model is no longer accurately predicted the actual lab results. The model must be made to match the actual system. There are two ways to do this. One method is to simply introduce a saturation block with limits of +/- 10 volts. This would exactly model the real system. The input voltage vs. time using a saturation block is shown below.

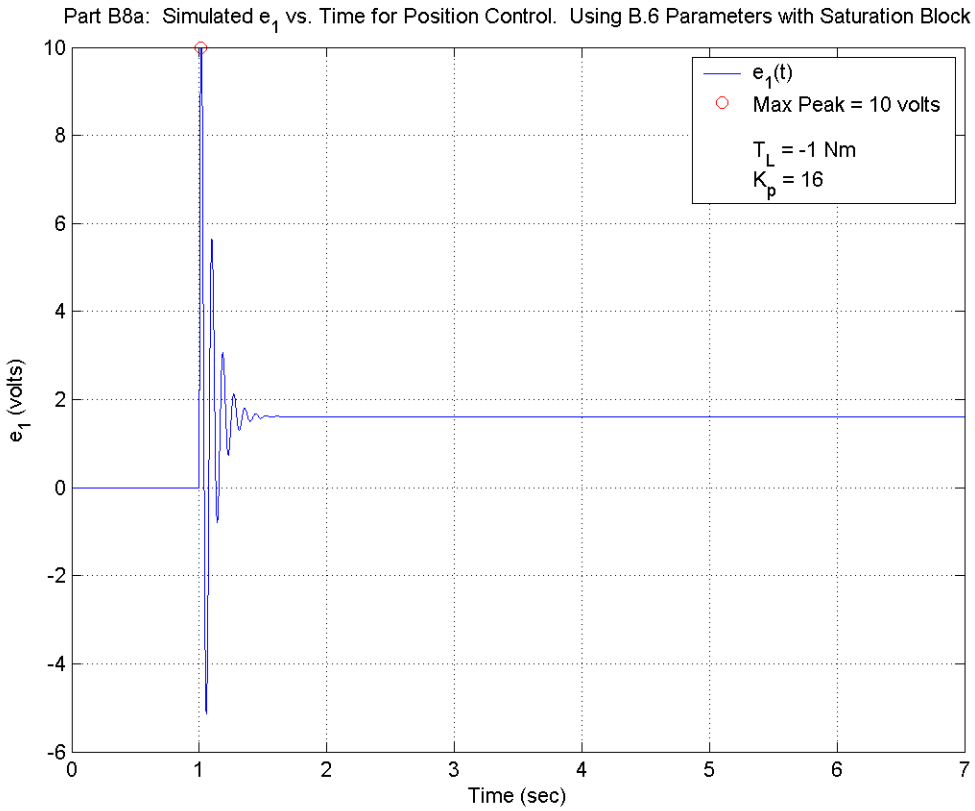


Figure 33: e_1 vs. time with saturation block

As can be seen here, the maximum voltage that occurs is only 10 volts. The saturation block serves to limit the voltage to a maximum of 10 volts. It only needs to work for a small time when the voltage spikes above 10 volts. Since the addition of the saturation block only changes the input voltage for a very small period, it is expected that the response of the system will not change very much. The simulated θ_1 vs. time is shown below.

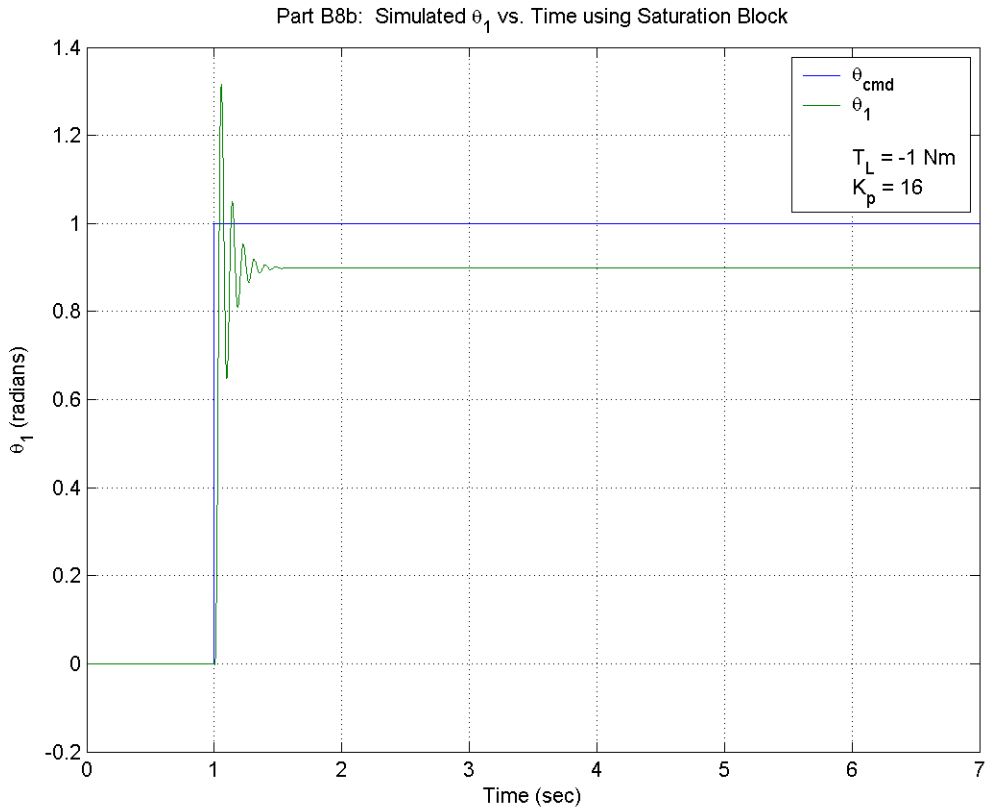


Figure 34: θ_1 vs. time using saturation block

As can be seen by comparing Figure 34 (simulation with saturation block) and Figure 29 (simulation without saturation block), the response is not very different. This makes sense considering that the saturation block did not affect the signal for some reason. As can be seen, the design still meets the performance criterion in section B.4.

Another option to fix the saturation issue is to decrease the break frequency of the prefilter. This will provide a more gentle reference signal. By lowering the break frequency of the prefilter to 50 rad/s, the following plot of e_1 vs. time is shown below in Figure 35

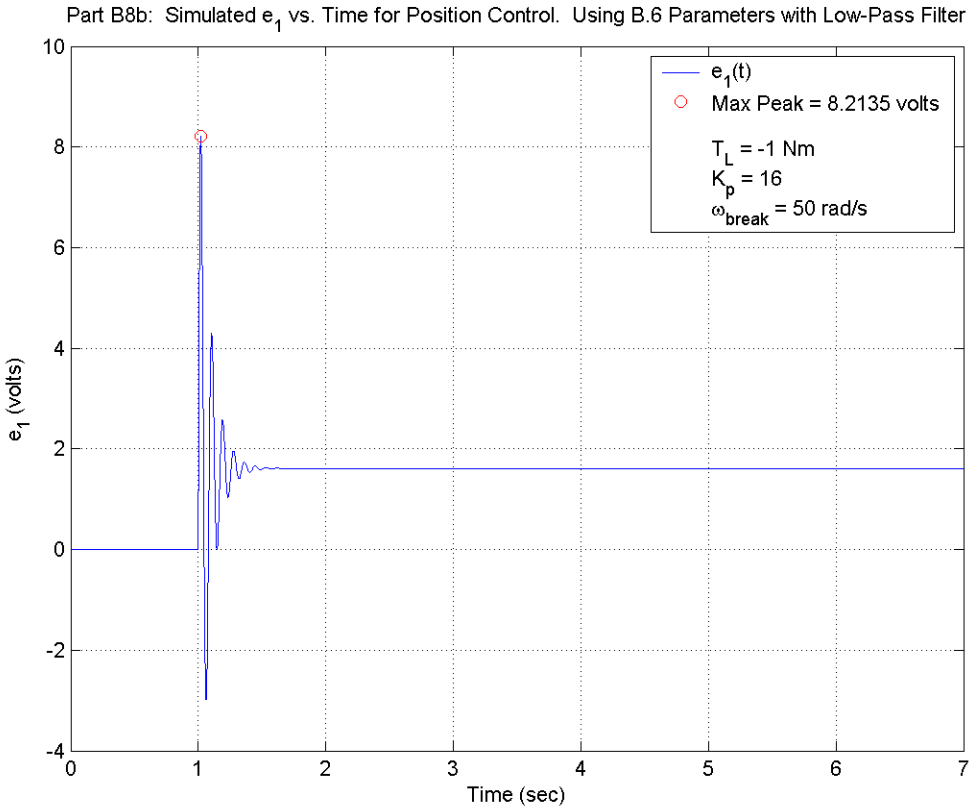


Figure 35: e_1 vs. time using a prefilter with ω_{break} = 50 rad/s

As can be seen from Figure 35, the more aggressive prefilter does in fact reduce the maximum voltage. The voltage does not exceed 10 volts in this situation. Therefore, the saturation block can be removed if this prefilter was in place. The actual system response with this control scheme is shown below

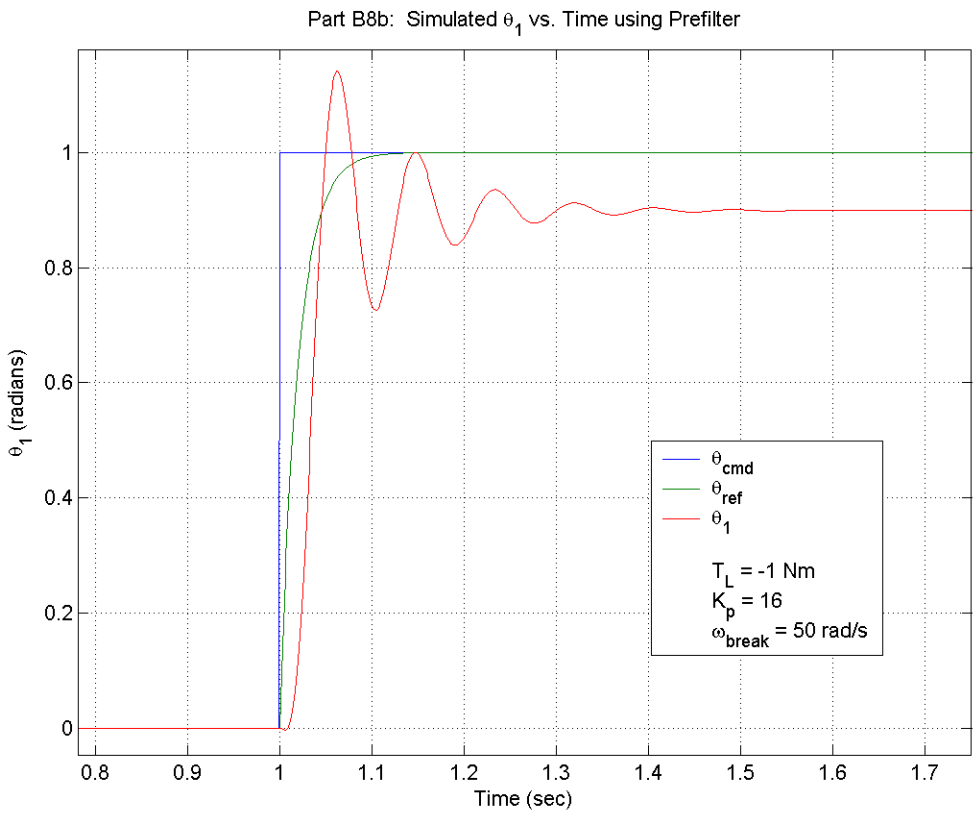


Figure 36: Simulated θ_1 vs. time with prefilter

By comparing Figure 36 (system with prefilter of ω_{break} = 50 rad/s) and Figure 29 (system with prefilter of ω_{break} = 100 rad/s), the response is slightly different, but not by much. The percent overshoot and rise time remain roughly the same.

Both of these methods ensure that the voltage remains below 10 volts. Both of them have advantages and disadvantages. The saturation block is probably the more accurate in terms of modeling the system. Since the system cannot physically produce more and +/- 10 volts, the saturation block ensures that the voltage into the plant never exceeds this. However, this is a non-linear method. Once the system saturates, the system cannot be analyzed using linear tools such as linmod, SISOtool, etc. For the small range of values for T_L and θ_{1cmd} used in part B, it appears that the prefilter with a break frequency of 50 rad/s prevents the voltage from exceeding +/- 10 volts. The benefit of the prefilter is the fact that the system remains linear during the entire simulation. Therefore it is available to analyze with linear tools. However, as the break frequency is lowered, so is the system bandwidth. However, in this situation where there are no dynamic reference signals, the prefilter is the preferred method.

Part C: Proportional/Derivative (PD) Controller

As stated in the lab sheet, a Simulink model without non-linear behavior is constructed as shown below in Figure 37.

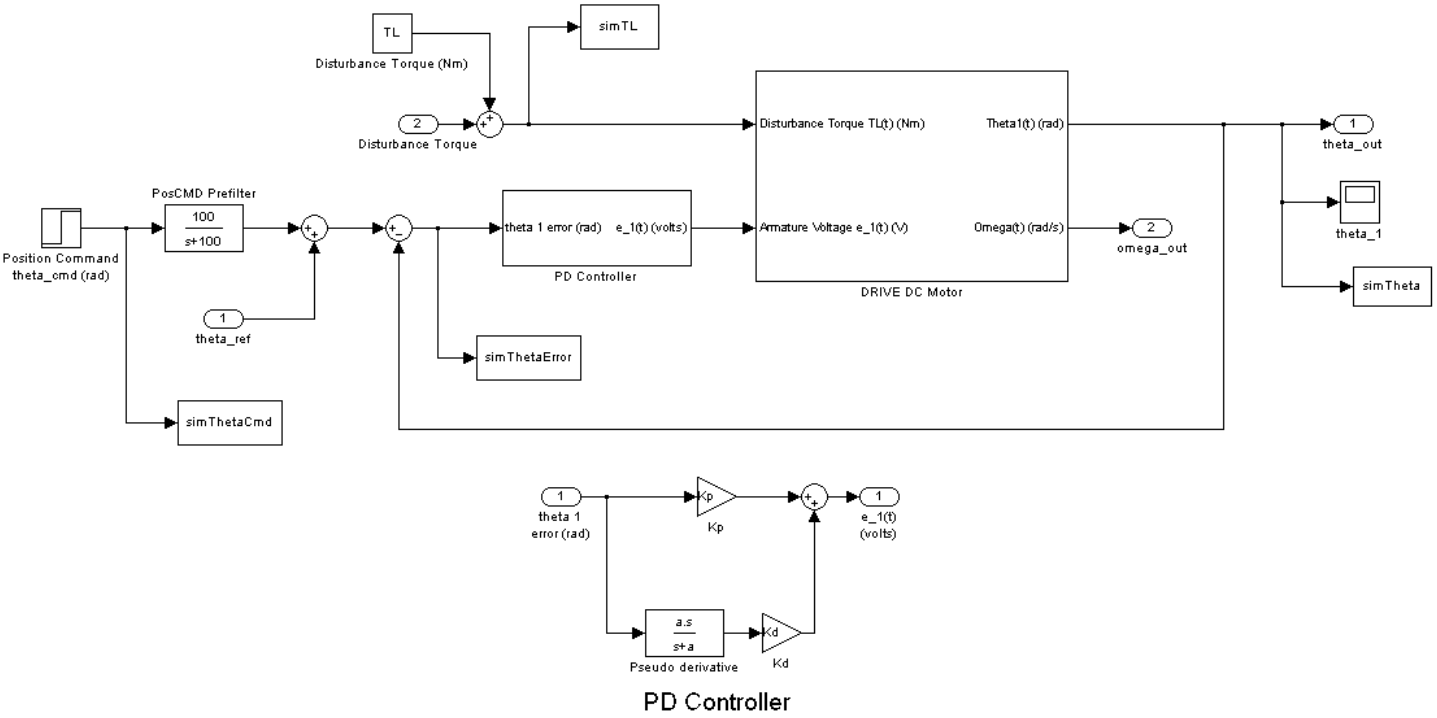


Figure 37: Simulink model with proportional and derivative controller

The gains K_P and K_D can be designed such that there is less than 5% overshoot with good tracking to a step command of 1 radian. The gains were designed in lab through trial and error. The response of the system with $K_P = 12$ and $K_D = 0.2$ is shown below in Figure 38,

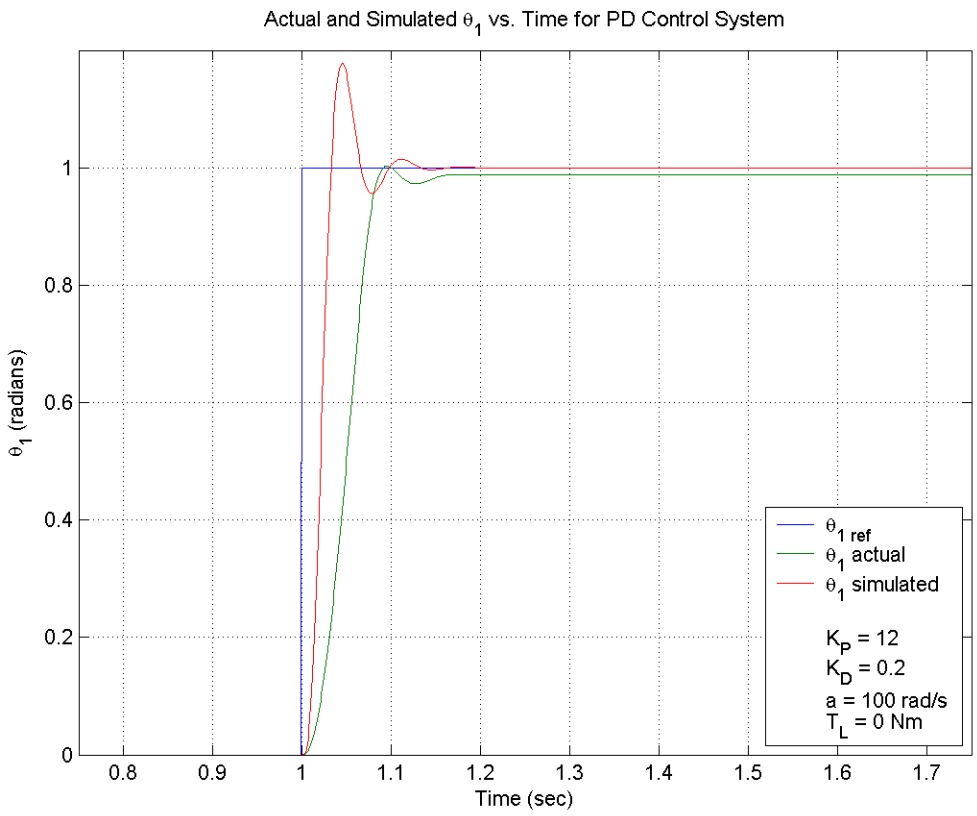


Figure 38: Actual and simulated θ_1 vs. time for PD controller

As can be seen, in lab, the response fits the performance criterion of less than 5% and a steady state error of less than 0.1 radians. However, the simulation does not agree with the actual lab data. The simulation model appears to me much less stiff than the actual system. This was also observed with the model used in Part A shown in Figure 25. The overshoot is approximately 17.8% in the simulation. Also, the simulation does not predict any steady state error since it does not contain any non-linear behavior.

The simulation can now be evaluated to see if the model provides accurate predictions of steady state error in the presence of a constant torque. The actual and simulated response is shown below in Figure 39.

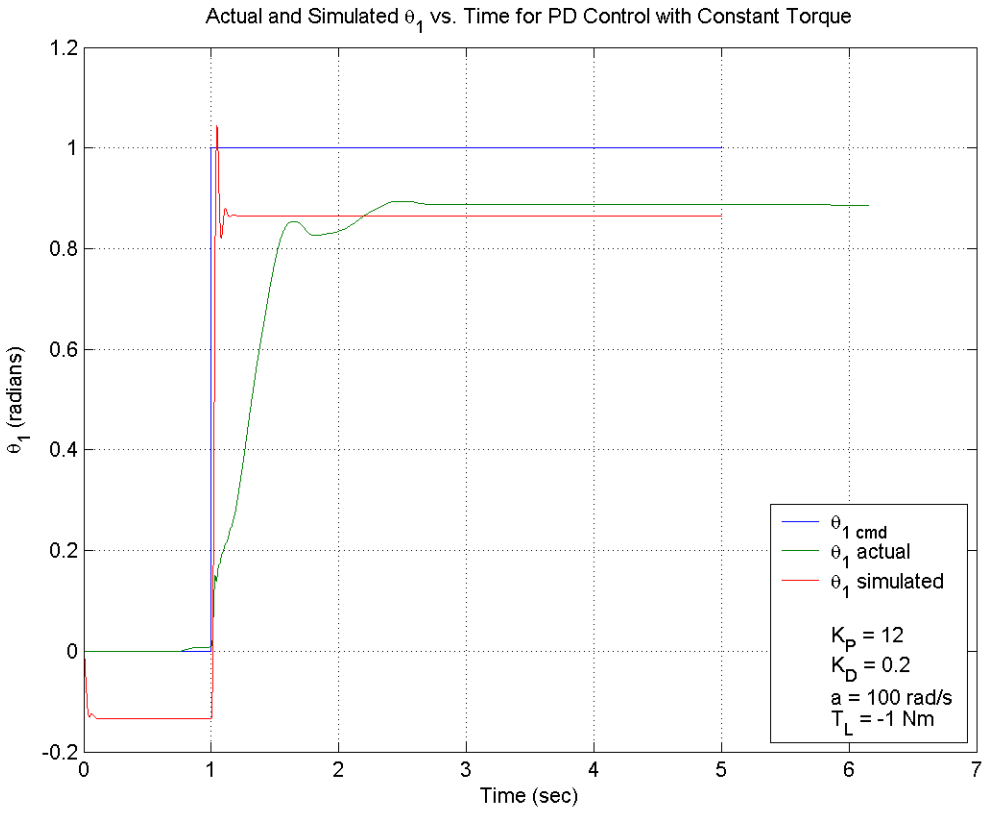


Figure 39: Actual and simulated θ_1 vs. time for PD control with T_L = -1 Nm

As can be seen, the simulation models the steady state error quite well. There is a discrepancy between lab and the simulation for the reasons mentioned above. In addition, the simulation applies the torque at t = 0 and the step command at t = 1 second. This accounts for the initial negative error.

In lab, a steady state error of roughly 0.1 radians was observed with this control setup. In simulation, the steady state error is predicted to be 0.134 radians. It appears that this does not meet the design specifications of less than 0.1 radians. However, in actuality, this control scheme meets the criterion. Also, the simulation predict steady state error it fairly accurately. In order to make the simulation meet the design specifications, the K_P gain could be increased slightly.

The voltage must be between +/- 10 volts in order to ensure that the system is not saturating. The simulated and actual e_1 is shown below in Figure 40.

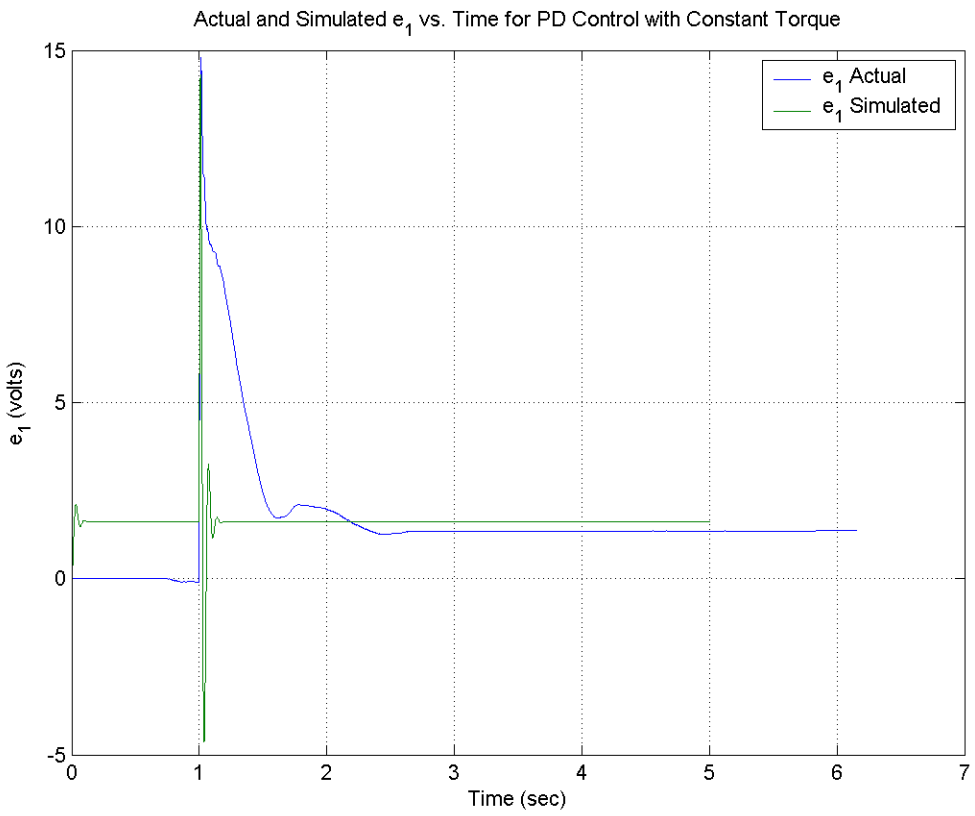


Figure 40: Actual and simulated e_1 vs. time for PD control with constant T_L

As can be seen above, the system exceeds 10 volts initially. However, at steady state, the system is well within the saturation limits. A saturation block could be introduced to make the model match the actual experiment more closely, but this would only affect the initial response of the system. It was already shown that the transient response of the system is difficult to model and since the goal of this is to evaluate the accuracy of the model at steady state, a saturation block is not needed.

Part D: Proportional/Derivative/Integral (PID) Controller

The integral addition serves to reduce the steady state error to zero. However, when the model does not include non-linear behavior such as a dead zone, the steady state error is zero by default. Therefore, in order to evaluate the contribution of an integral controller, a dead zone block is placed in the model as shown below in Figure 41.

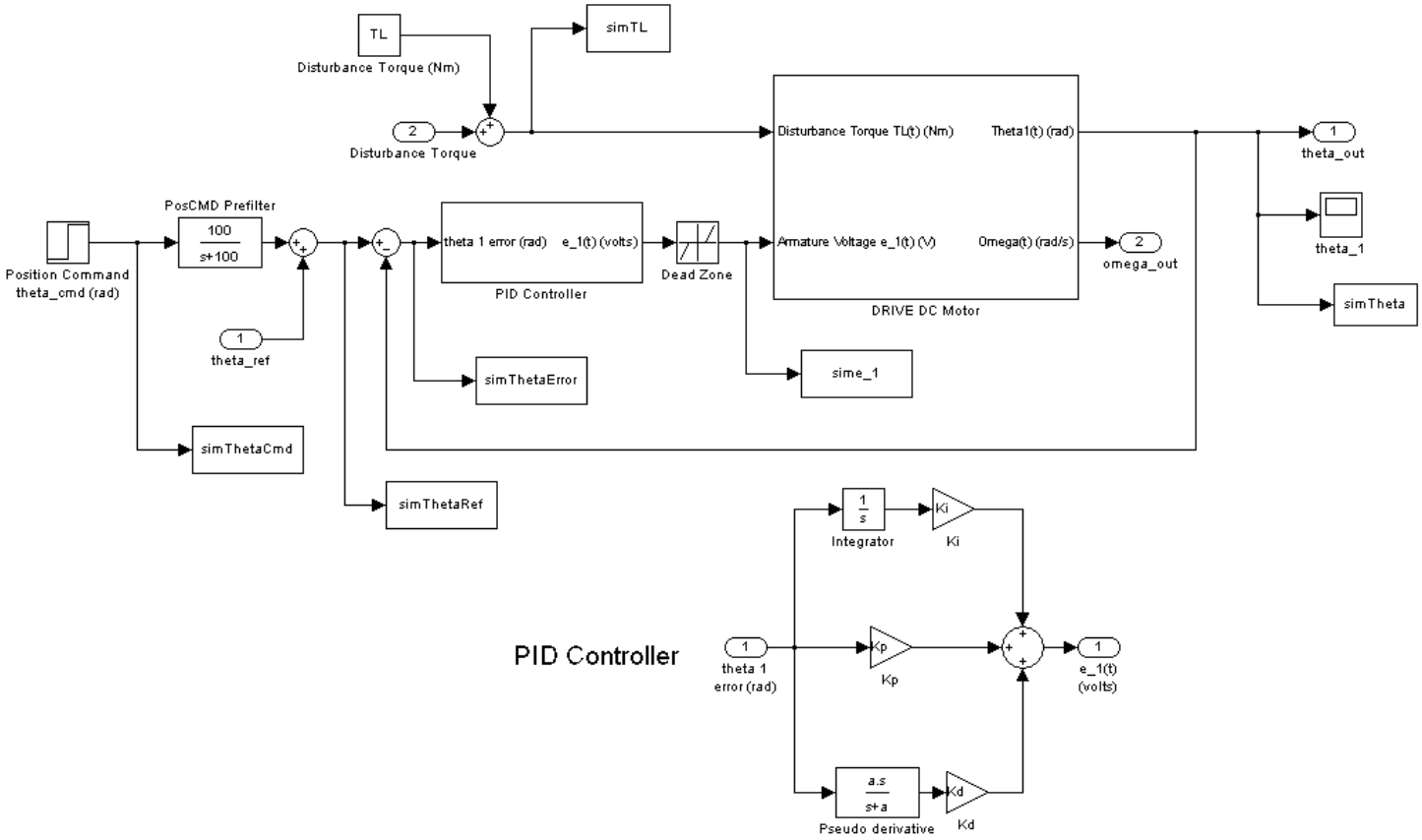


Figure 41: PID controller

The response of the simulation to a step input with no torque can be compared with actual lab results. A gain of $K_I = 3$, $K_P = 12$, and $K_D = 0.2$ is used. Furthermore, the dead zone of +/- 0.2 volts is used. The response is shown below in Figure 54.

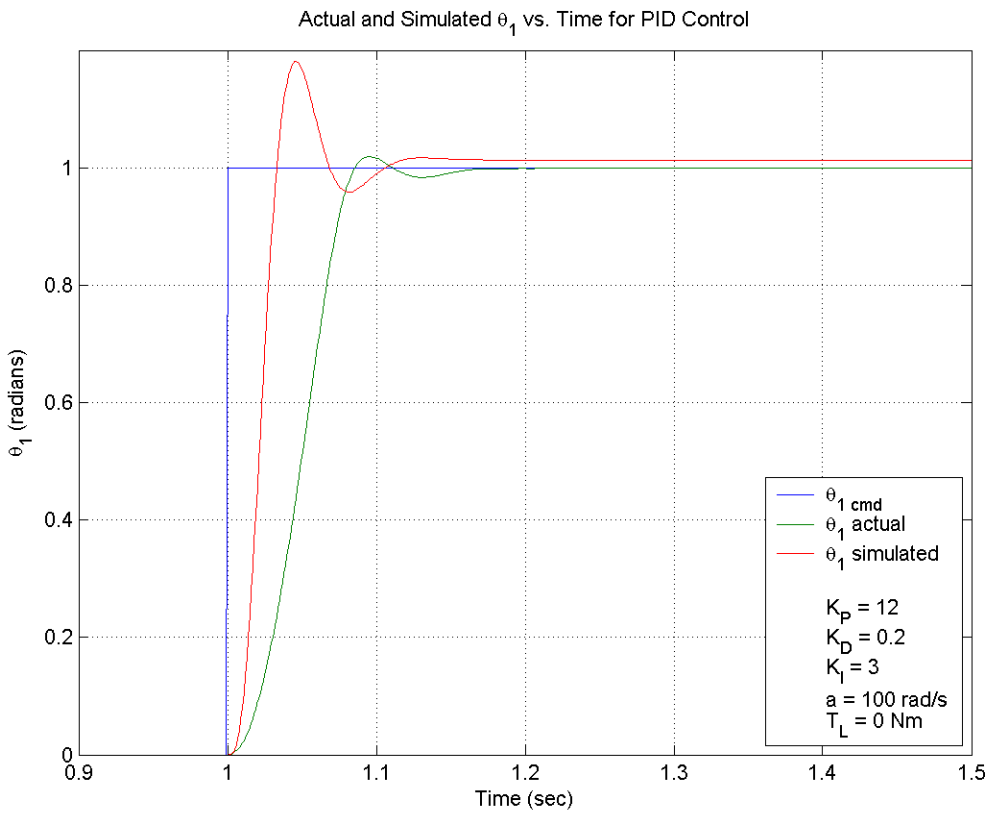


Figure 42: Actual and Simulated θ_1 vs. time for PID control including dead zone.

As can be seen, the simulation response is close to the actual lab response. The most concerning discrepancy is the apparent persistence of the steady state error. In the lab, a K_I value of 3 was sufficient to reduce the steady state error to zero in a sufficient time. However in simulation, it appears that this gain is not aggressive enough. By observing the simulation for a longer amount of time, Figure 43 is obtained.

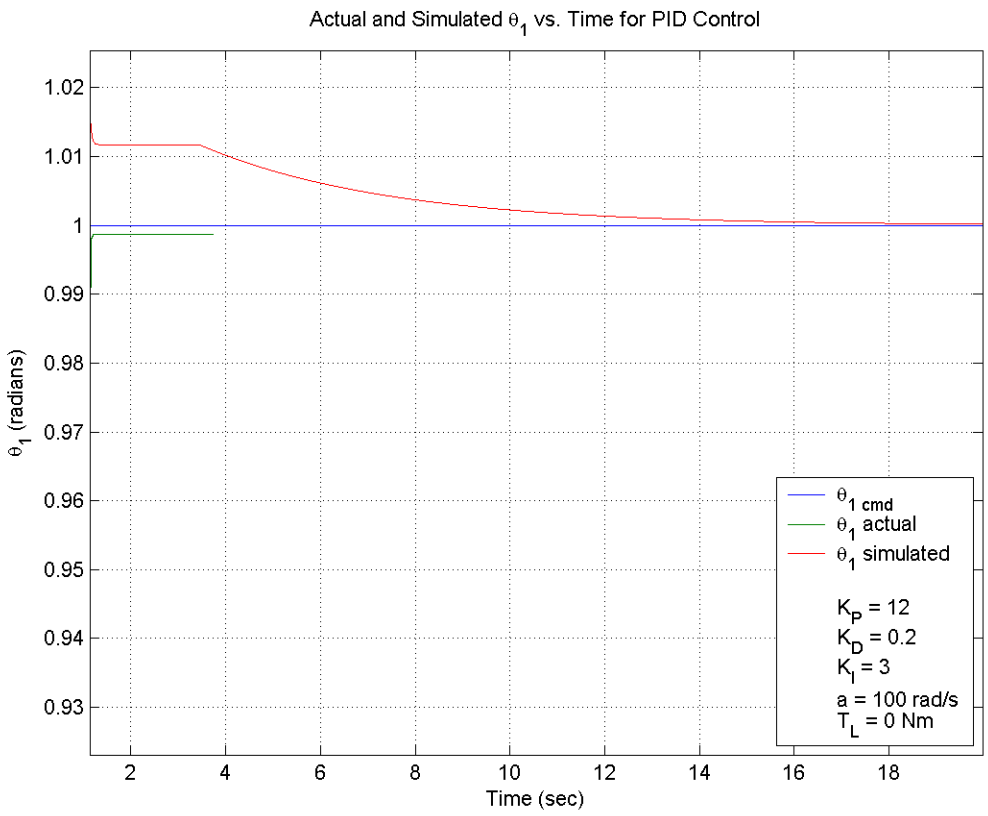


Figure 43: Actual and simulated θ_1 vs. time for PID showing I contribution

As can be seen, in simulation, the integral contribution is indeed working to reduce the steady state error to zero. However, it takes almost 19 seconds to do this where as in lab, the steady state error was reduced much quicker. In this respect, it appears that the model is not very accurate. Further discussion of this is included in the performance section.

The simulation can also be compared to the lab data in the presence of a constant torque. The results are shown below in Figure 44.

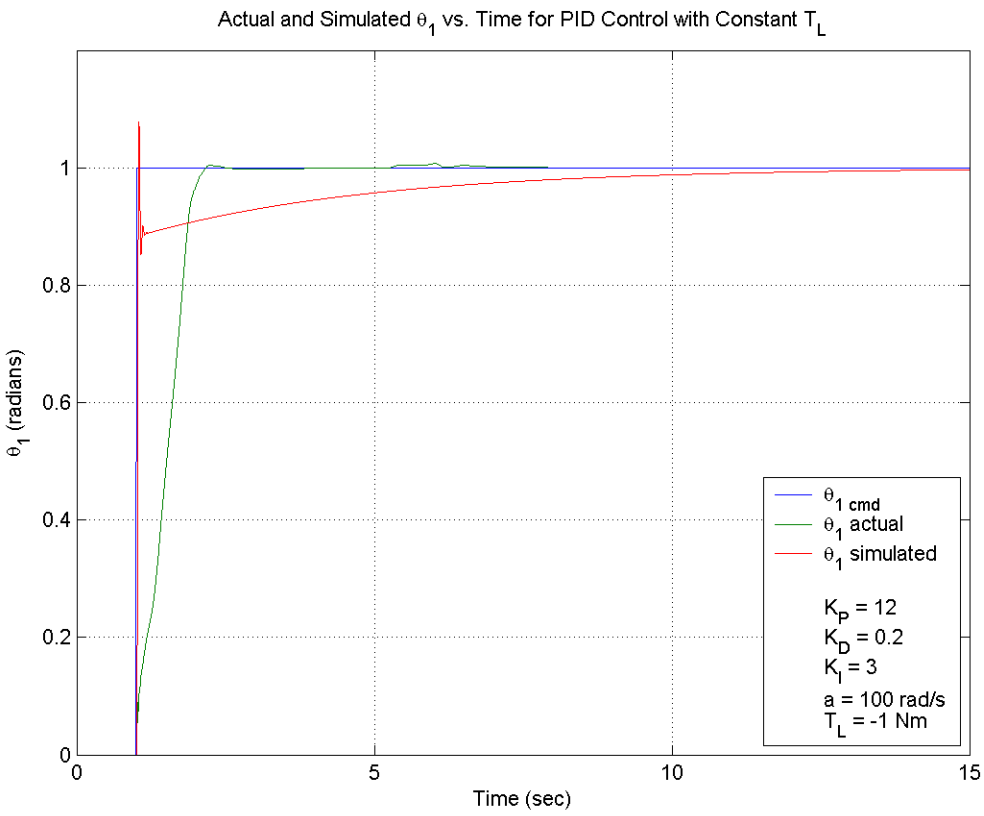


Figure 44: Actual and simulated θ_1 vs. time for PID control with constant T_L

Once again, it appears that the simulation produces results that are somewhat close to the lab results. The integral gain works reduces the steady state error quicker in this case due to the fact that the error is greater initially, therefore the integral term is able to accumulate the signal faster and overcome the dead zone quicker. At steady state, the simulation accurately predicts the actual response of the system, but the dynamic response is somewhat inaccurate. One possible way to fix this is to lower the dead zone such that it matches the lab results better.

While keeping the same controller, the drove motor model and be replaced with the full configuration model of the flexishaft experiment. The load motor and input and encoder position output remain unconnected and not used. Since this was not tested in lab, only the simulated results can be analyzed. The Simulink model used is shown below in Figure 45.

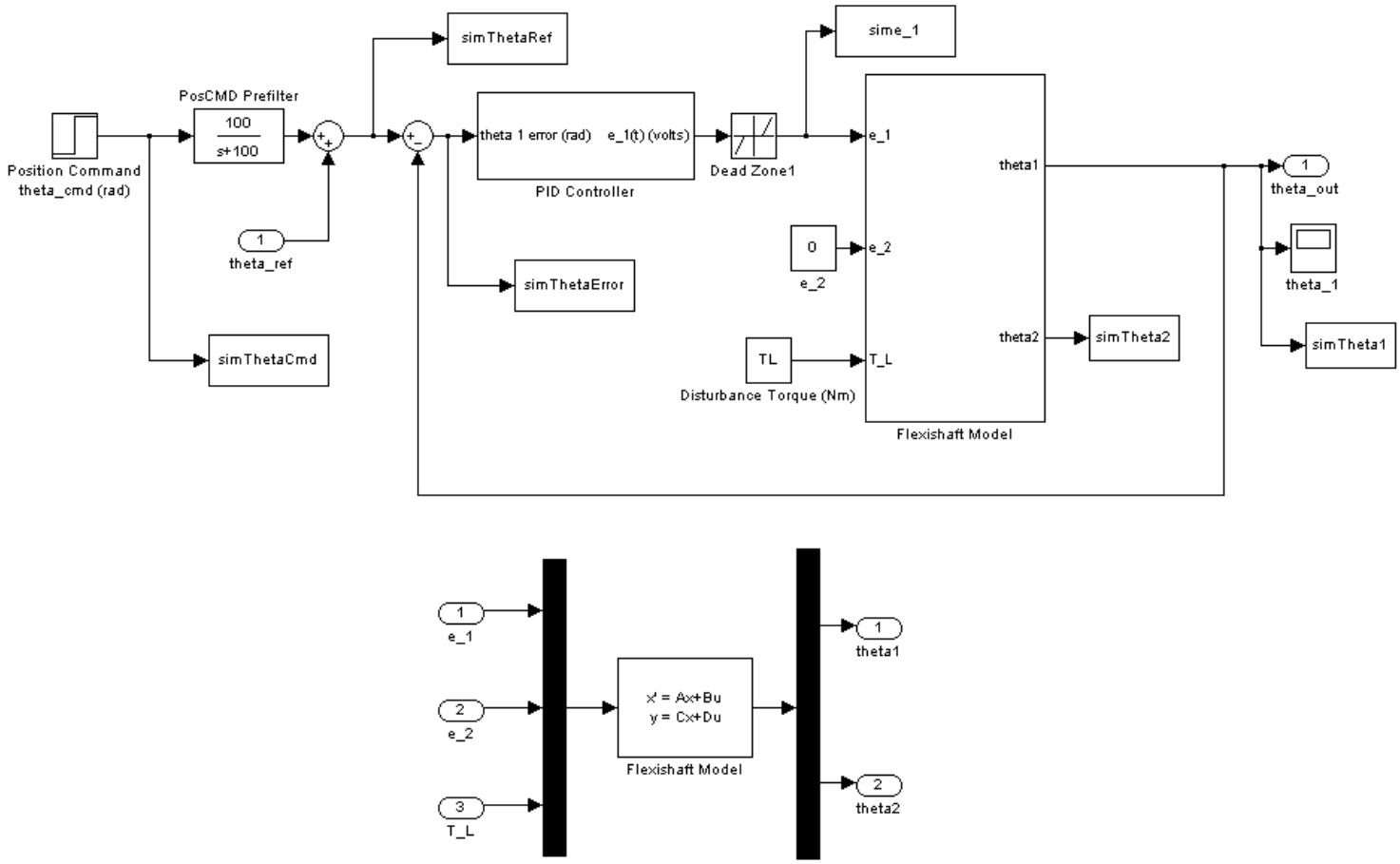


Figure 45: Simulink model used to simulate system when flexible shaft is connected

Using the same PID controller designed in the beginning of this section, the response of the system to a step position command can be plotted as shown below in Figure 46.

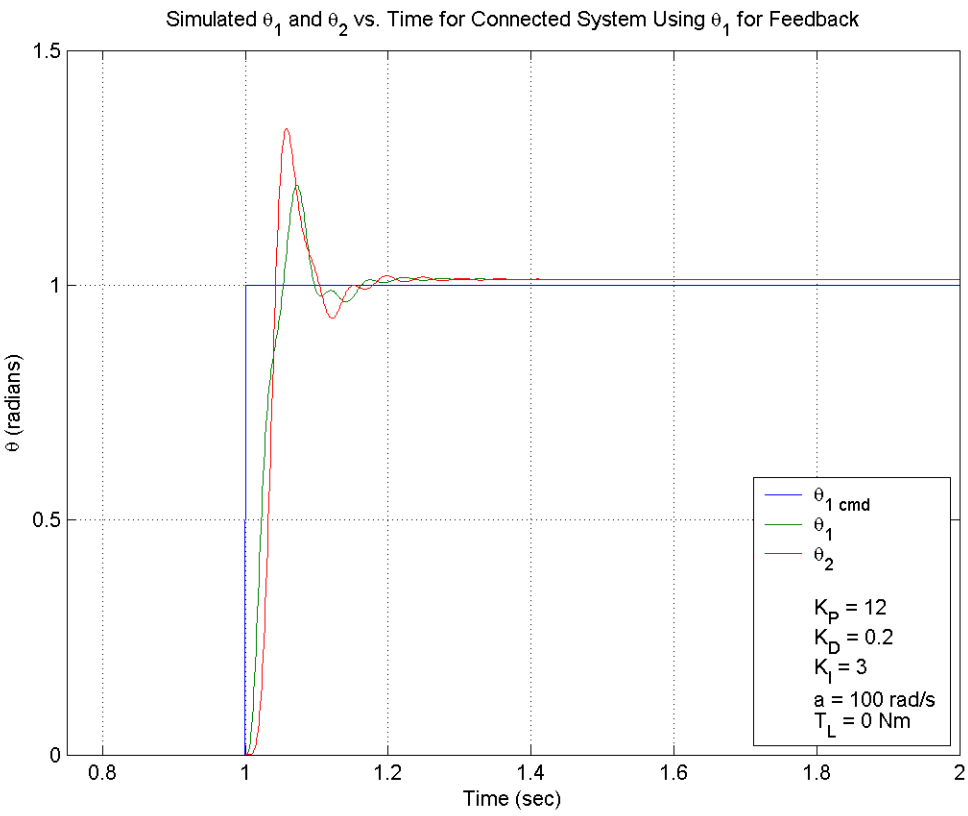


Figure 46: Simulated θ_1 and θ_2 vs. time for coupled system using θ_1 for feedback

As can be seen, due to the flexible modes of the shaft, the response of θ_1 and θ_2 are not exactly equal. The complex poles that represent the flexible mode shows why there are oscillations present in the system. In addition, the presence of the load motor actually changes the dynamics of θ_1. The simulated response of θ_1 with and without the coupling is shown below in Figure 47.

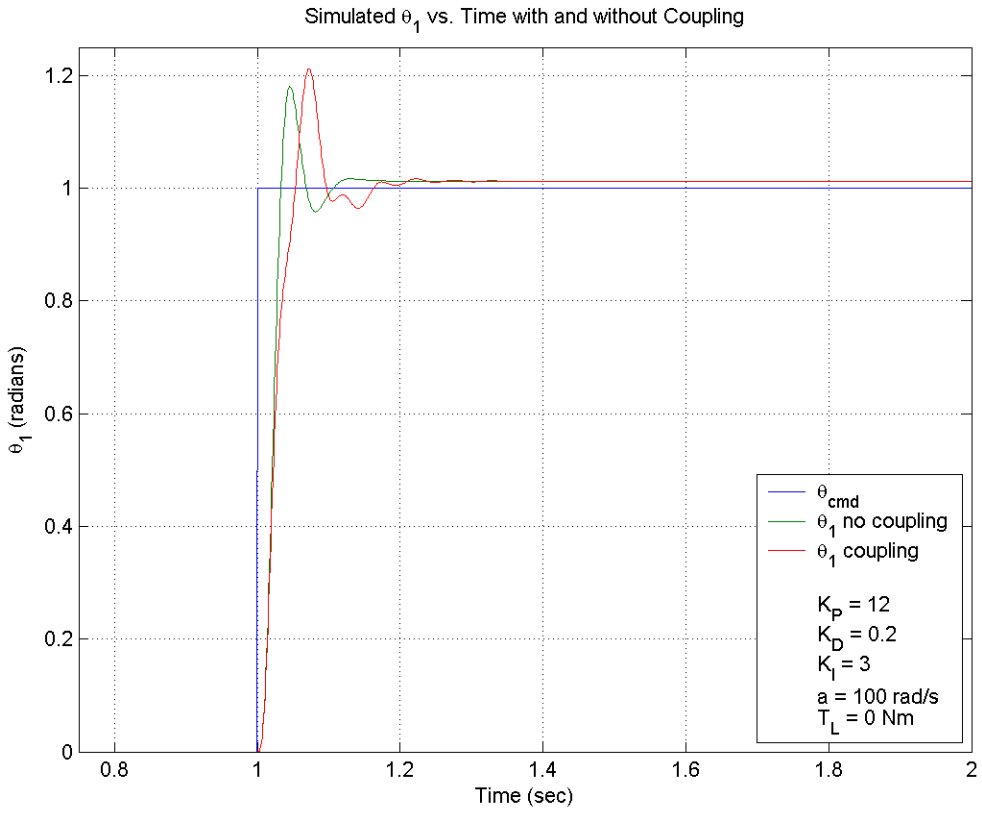


Figure 47: Simulated θ_1 vs. time with and without coupling effects

As can be seen, the presence of the load motor affects the response of θ_1.

The significance of the flexible shaft can be dramatically noticed if θ_2 is used for feedback instead of θ_1. The response of the system as it attempts to track a 1 radian step is shown below in Figure 48.

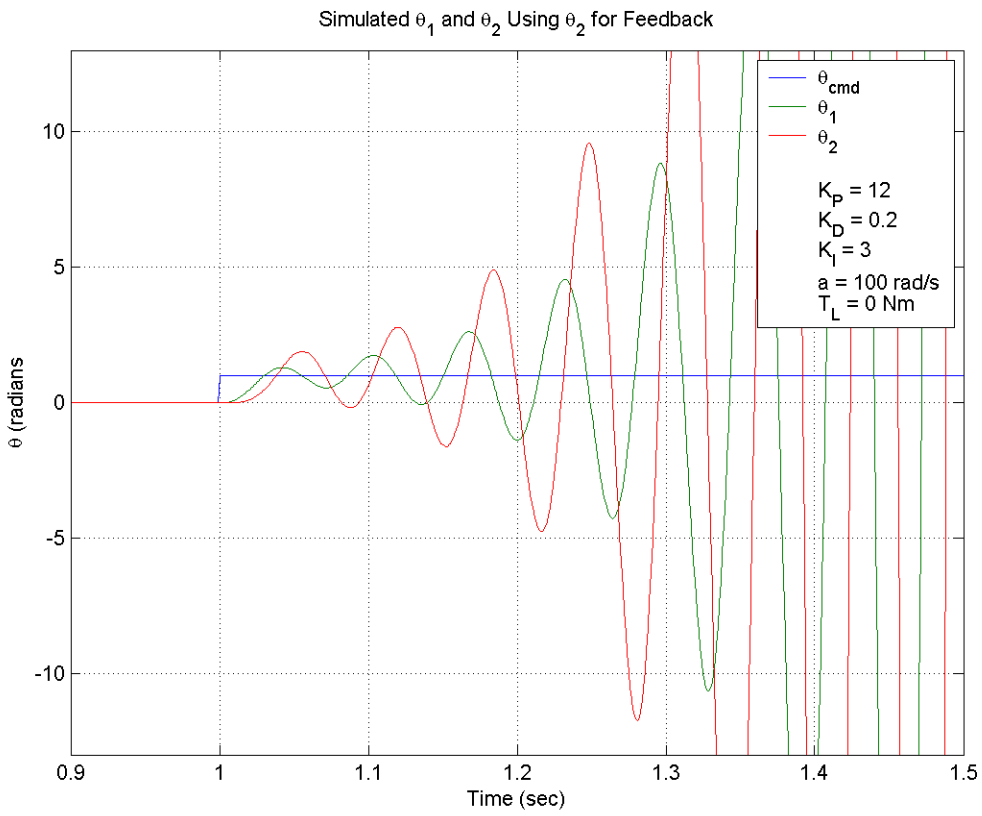


Figure 48: Simulated θ_1 and θ_2 using θ_2 for feedback

As can be seen above, using θ_2 as feedback yields and unstable system. The time delay and flexibility introduced by the flexishaft makes this system unstable when using the PID controller designed above. Keep in mind that the simulating makes use of the state space equation shown above in Eq.3 and Eq.4. These are using the published values given in the gfshaft.m file. However, as proven above in the system modeling section, this is not a very accurate representation of the system. Therefore, the simulated results may not be accurate.

Part E: Notch Filter

The block diagram showing the Simulink model used to simulate the response of part E was already shown above in Figure 17. The control structure of using θ_2 as feedback was chosen so that direct simulation would be possible. The transfer function $G(s) = \theta_2/e_1$ was derived using methods described in the System Modeling Section. In order to validate the accuracy of this transfer function, the notch filter in Figure 17 can be removed and the response of the system to a step command of 1.75 radians can be plotted. Although the notch filter was removed, it is important to leave the constant gain of 0.875 in the model for simulating the response with no notch filter. This is why in Figure 17, the gain is included no matter if the notch is switched on or off. The reason for this is because it was already showed that the amount of overshoot and how much the complex poles is excited is directly proportional to the proportional gain. For example, if the notch and the gain were removed, the response of the system with must the PI controller is shown below in Figure 49.

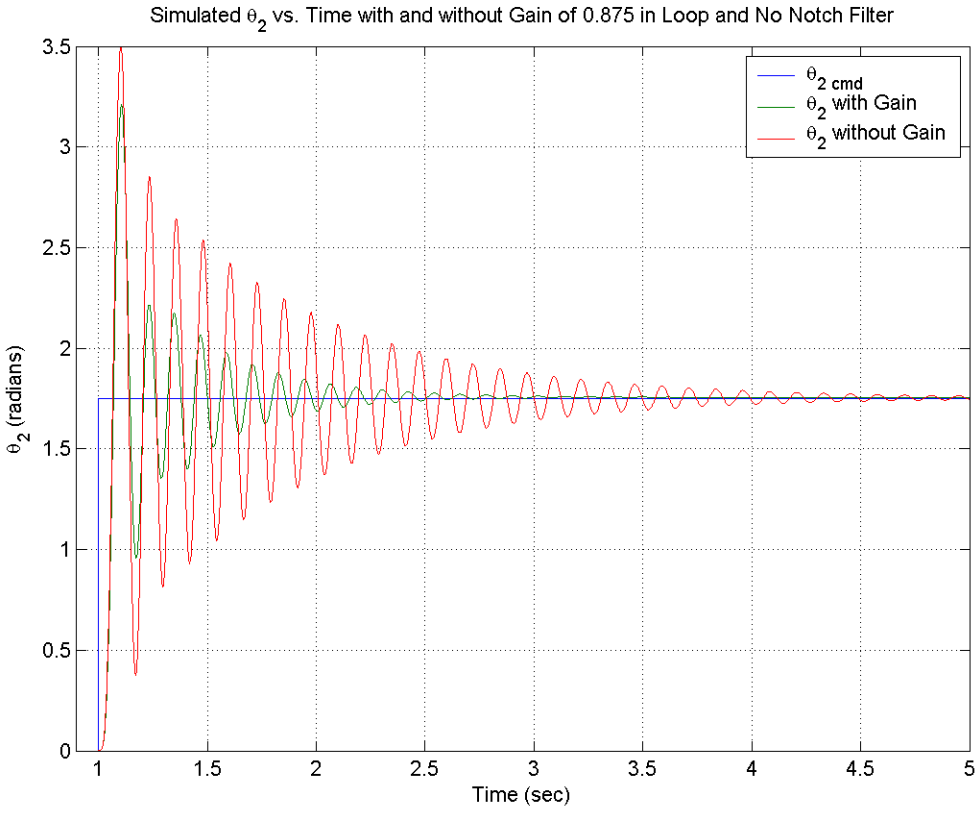


Figure 49: Response of System with no notch filter with and without gain of 0.875 in loop

As can be seen, even without a notch filter, by adding a gain less than 1, the system becomes more damped. It settles out faster and has less oscillations. Therefore, if the system is run without the gain, and then with the gain and the

notch filter, it may appear that the notch filter is damping out the response more than it actually is. In this situation, the damping effect would come from both the notch filter *and* the fact that a gain of less than 1 was introduced. Since the objective of this section is to evaluate how the notch filter adds damping, it is important to leave the gain in the system in order to isolate the effect of the notch filter.

The response of the system with no notch filter and a constant gain of 0.875 is shown below in Figure 50.

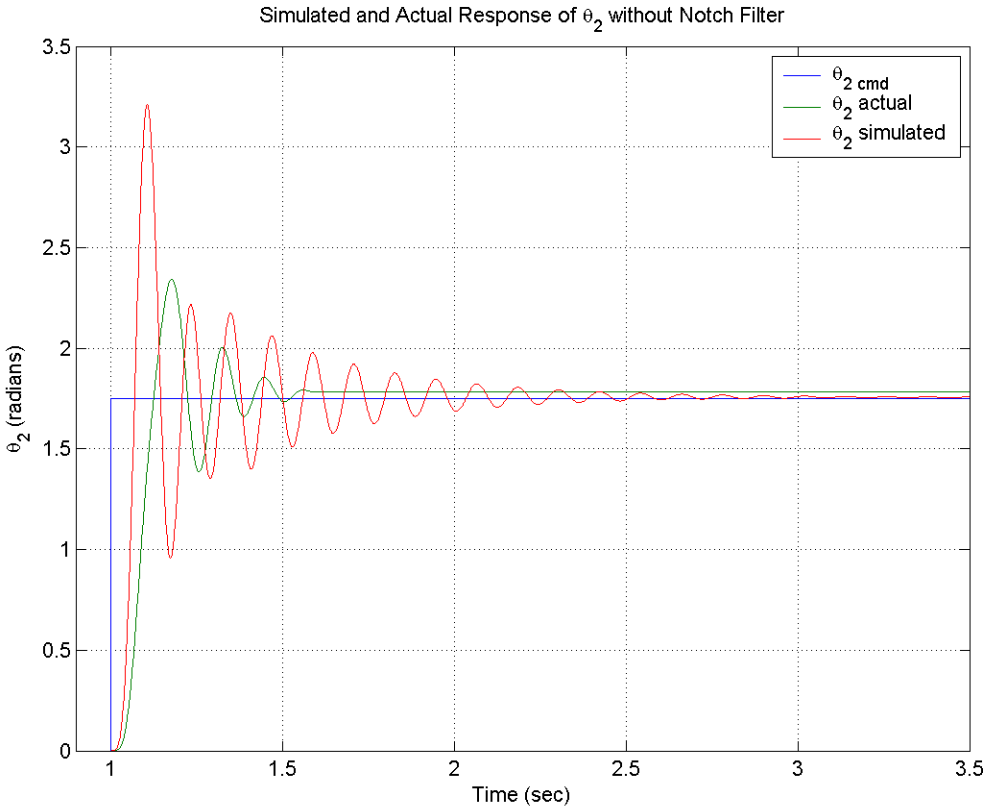


Figure 50: Response of θ_2 vs. time to a step command with no filter

As can be seen above, the simulation does a fair job of simulating the response of the actual system. It predicts that the system will be stable with PI controller with $K_P = 6$ and $K_I = 2$. The simulated system once again appears to be slightly less stiff than the actual system. There could be several reasons for this. One is that the simulation is highly dependent on the derived transfer function G(s). This transfer function was derived using least square's analysis. There are more accurate and sophisticated ways to find a solution for a transfer function that would match the actual DSA output. Furthermore, the simulation does not contain any non-linear blocks to model effects such as stiction. This also could

explain why there is no steady state error and why the simulation is less stiff than the actual response.

The simulated and actual effect of the notch filter is evaluated next. The response of the system to step of 1.75 radians with the notch filter on is shown below in Figure 51.

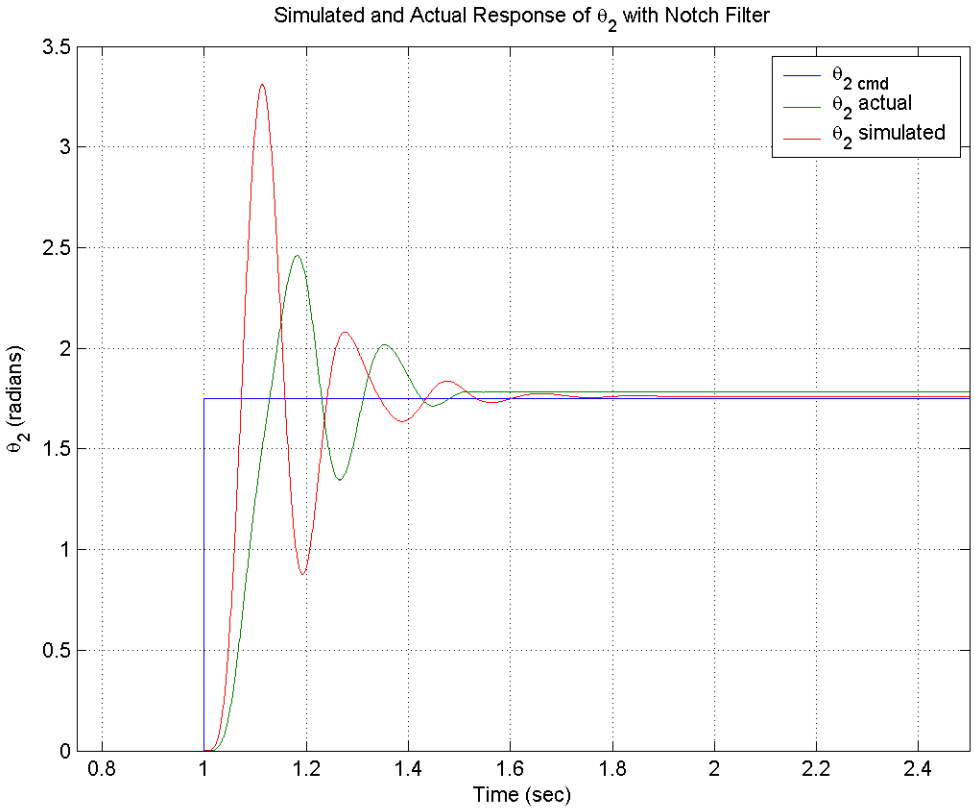


Figure 51: Response of θ_2 vs. time in response to step of magnitude 1.75 radians

As can be seen, with the notch filter on, the simulation predicts actual results much better. Furthermore, the simulation predicts that the number of oscillations will decrease and that the settling time will decrease as well.

Figure 51 yields somewhat mixed results when compared with the simulation with the notch filter off. Figure 51 appears to show that the transfer function derived is very accurate since the simulation matches the actual data. However, Figure 50 shows that the simulation is less stiff than the actual motor. The only difference between the two plots is the presence of the notch filter. One possible reason why this occurs is that in lab, the notch filter served to slow down the oscillations and make the system damp out quicker. This leaves fewer opportunities for the non-linear behavior of the real system to show up. This is why the simulation with the notch filter is closer to the actual data.

In addition to a simple step, the simulated response of the system to a sin wave can be compared with laboratory results. A sin wave of magnitude 1 and of frequency 60 rad/s is used. The actual and simulated response of the system with and without the filter are shown below in Figure 52 and Figure 53, respectively.

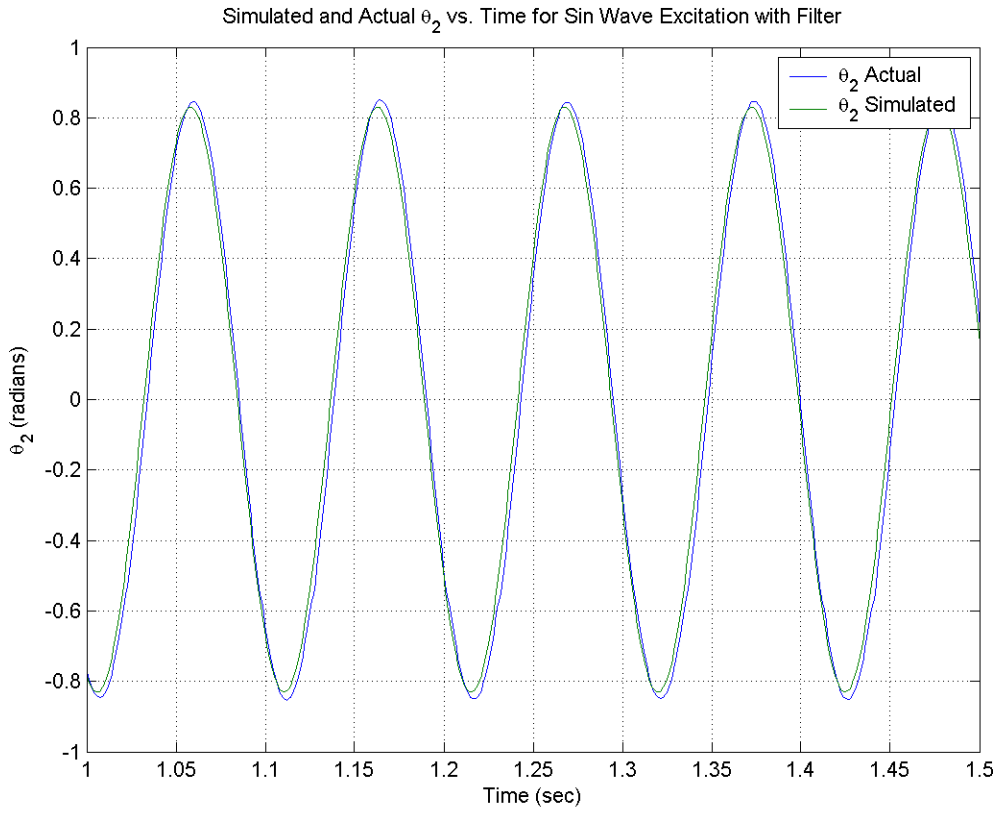


Figure 52: Simulated and actual θ_2 vs. time for sin wave excitation with notch filter

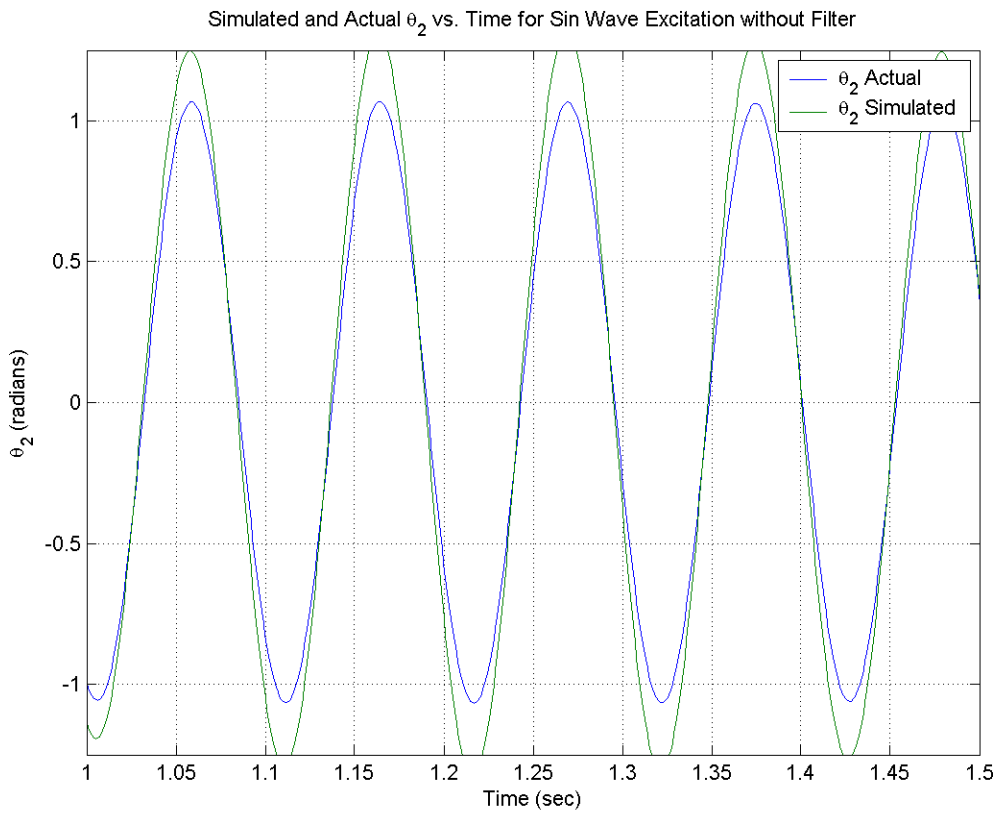


Figure 53: Simulated and actual θ_2 vs. time for sin wave excitation without notch filter

As can be seen, the results above agree with the results of the step response. Without the filter, it appears that the simulation is a more flexible model. This is evident from Figure 53 where it predicts that the structural mode will be excited and the amplitude will be larger than the input amplitude. However, it seems to over predict the actual system response. However, with the filter on, the simulation lines us exactly with the actual laboratory results. Once again, in situations where the non-linear behavior of the system is kept to a minimum, the simulation seems to agree very well with the actual data. However, it has a somewhat narrow range of accuracy as shown above where it may deviate from actual results once the non-linear behavior becomes significant.

VIII. Control Design Performance

The performance of the proportional, proportional-differential, proportional-integral-differential controllers are evaluated in response to a step input. Additionally, the performance of the notch filter based controller used to provide position control of the load motor is analyzed. This analysis focuses on the response's overshoot, rise time, settling time, and average steady-state error. In cases where the system does not attain the commanded position, the response's overshoot and rise time are of no meaning.

a. Open Loop Control

Due to the open loop nature of this, the performance is not applicable. All performance related to this has been previously discussed.

b. Proportional Position Control

The final design for a proportional controller used a proportional gain of 3.1. Figure 54 and Figure 55 show the response of the proportional position controller with and without an applied disturbance torque, respectively. Table 7 summarizes the performance of the two systems.

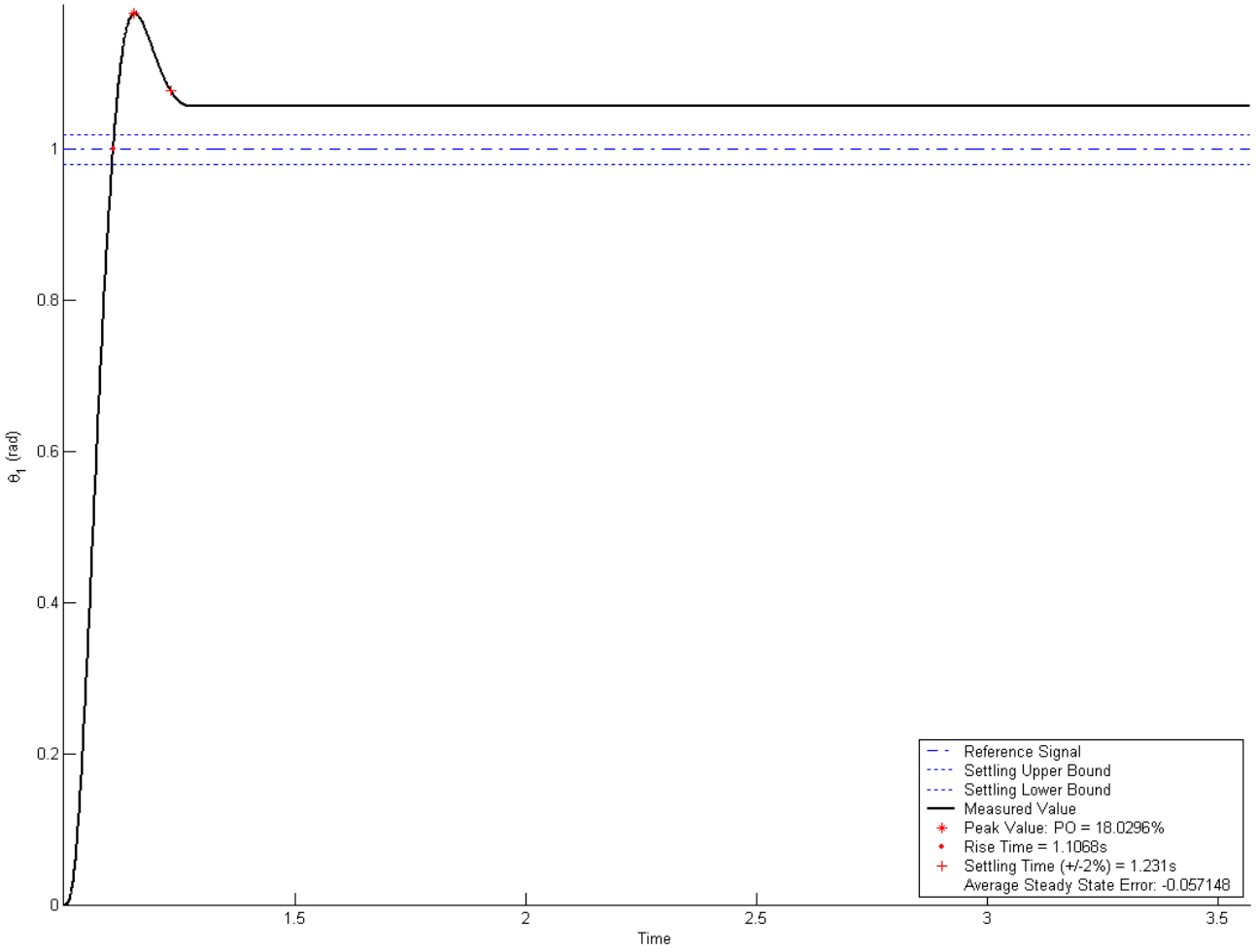


Figure 54 - Step response of proportional controller without disturbance torque (Run 4).

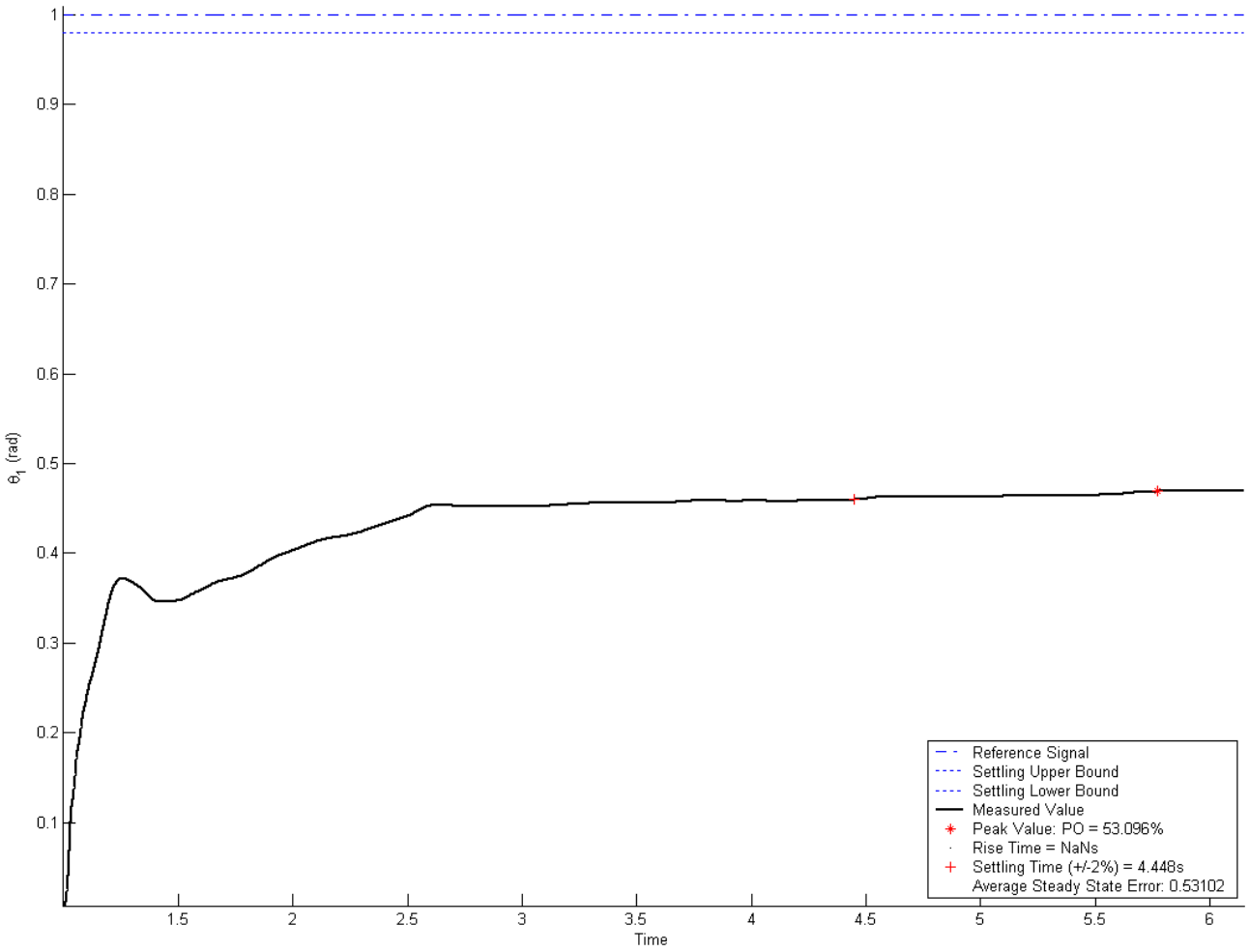


Figure 55 - Proportional position controller step response with applied disturbance torque (Run 5).

Table 7 - Summary of proportional position control performance.

T_L	PO	Rise Time	Settling Time	$Error_{ss}$
0 N-m	18.03%	1.12s	1.23s	-0.057 rads
1 N-m	N/A	∞	N/A	0.531 rads

From the results shown in the above figures the robustness of the proportional controller is observed to be almost non-existent. This is as expected and is inherent in the nature of proportional control. Once the disturbance torque has been applied in a steady manner, the system finds itself in a constant state of error and is unable to correct itself. A similar effect occurs even in the absence of the applied torque. This disturbance comes in the form of stiction in the drive motor. As the position error is decreased the correction applied by the proportional controller is reduced. Eventually, the system lacks enough momentum and correction to overcome the motor's stiction.

To improve upon these results, a differential or lead controller and an integral or lag controller would need to be added. The differential controller would reduce the amount of overshoot while the integral controller would serve to eliminate the steady-state error introduced by stiction and an applied torque.

c. Proportional-Differential Position Control

The final proportional-differential position controller design utilized a proportional gain of 12, a differential gain if 0.2 in conjunction with a pseudo-derivative roll-off frequency of 100 rad/s. This controller was then analyzed for performance using a one radian step command with and without a manually applied disturbance torque of 1 N-m. Figure 56 and Figure 57 show the system's response to the test cases. Additionally, Table 8 summarizes the performance results.

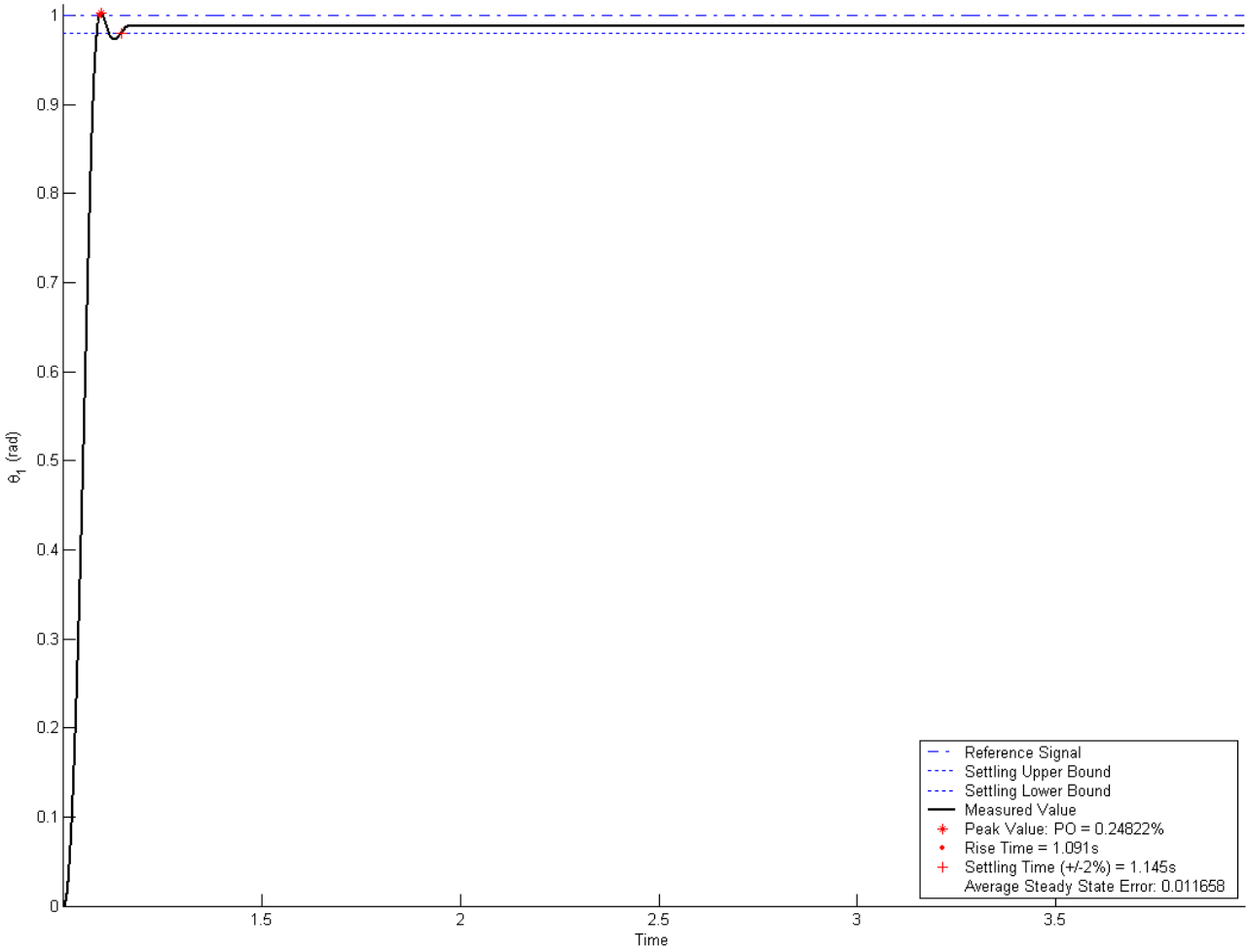


Figure 56 - Proportional-differential controller response to step input without T_L (Run 3).

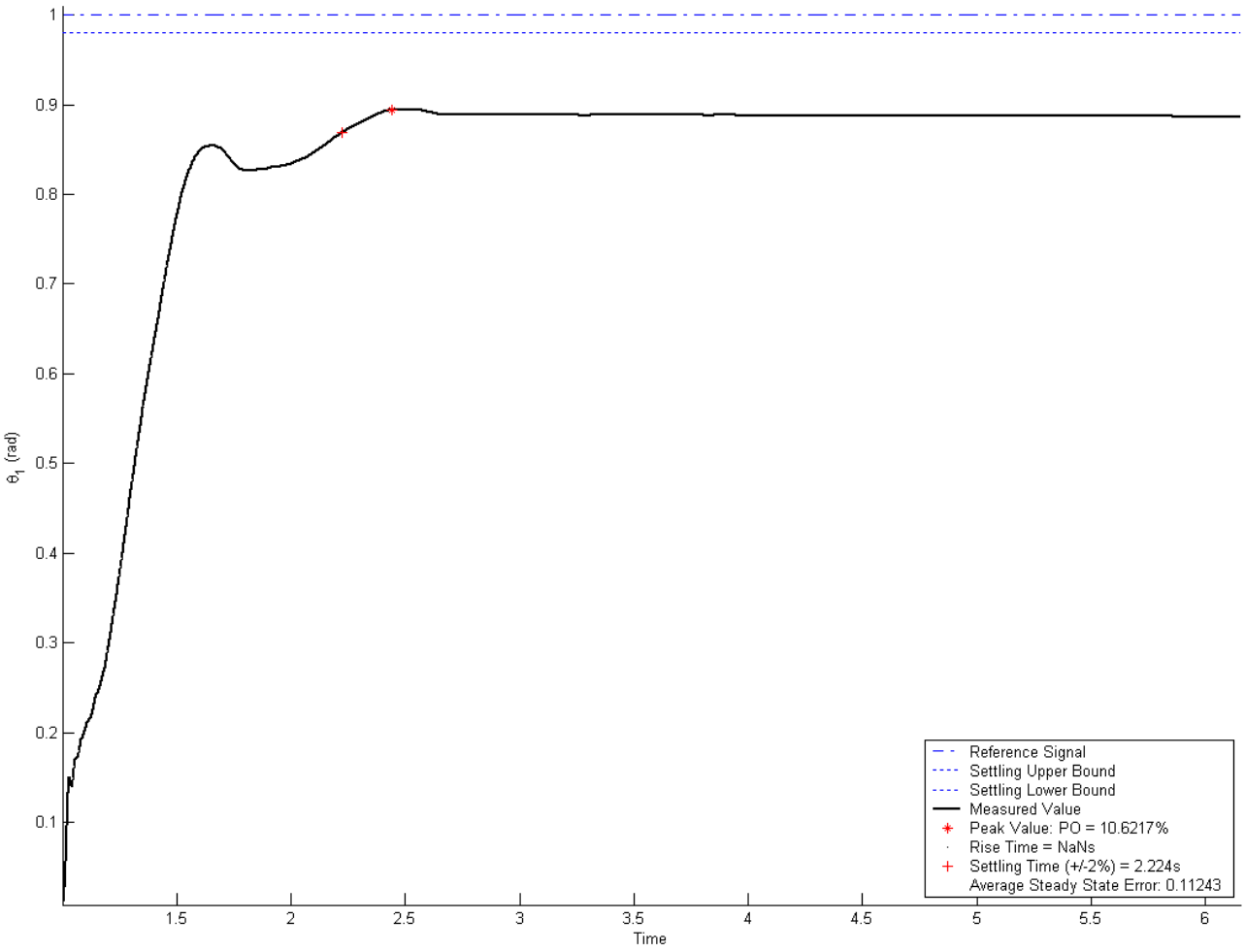


Figure 57 - Proportional-differential position controller response to step command with T_L (Run 4).

Table 8 - Summary of proportional-differential position control performance.

T_L	PO	Rise Time	Settling Time	$Error_{ss}$
0 N-m	0.248%	1.09s	1.15s	0.012rads
1 N-m	N/A	∞	N/A	0.112rads

Although this system performs better than the simple proportional control does, it is still lacking in actual disturbance rejection. The apparent disturbance rejection is simply the effect of a larger proportional gain demanding more correction in response to the applied torque load.

Despite this shortcoming the proportional-differential controller provides the expected benefit of damping out the system's tendency to overshoot. This allows a greater proportional gain to be used which yields two results: a faster response time and greater apparent disturbance rejection.

In cases where a full PID controller would be infeasible due to implementation restraints or cost, the P-D controller could be utilized if the some estimate of the magnitude of disturbances could be made. This estimate would allow the proportional gain to be increased to the point that it would provide an adequate level of rejection to the expected disturbances. The consequence is still a limited robustness. If disturbances occurred that were of greater magnitude than those originally expected, rejection effectiveness would increase with the magnitude of disturbance.

d. Proportional-Integral-Differential Position Control

The design of a proportional-integral-differential (PID) controller was expanded from the previous P-D controller design. Using the same proportional and differential gains as in the previous section, and integral gain of 3 was incorporated into the controller. Figure 58 shows the system's response when tested with a position step command on one radian. The addition of a 1 N-m disturbance torque to the same step command is shown Figure 59. Results of both tests are summarized in Table 9.

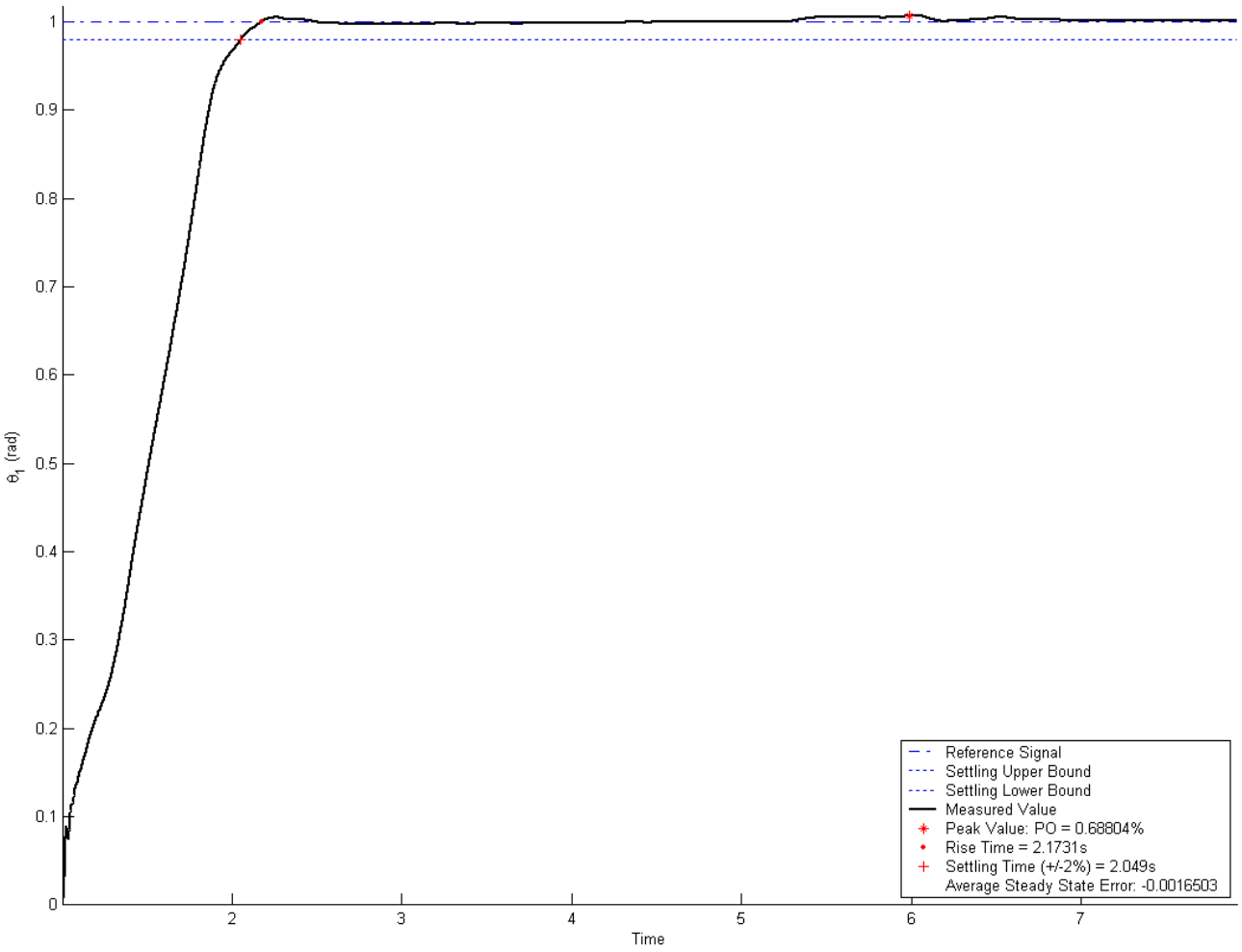


Figure 58 - PID position controller response to step command without disturbance torque (Run 1).

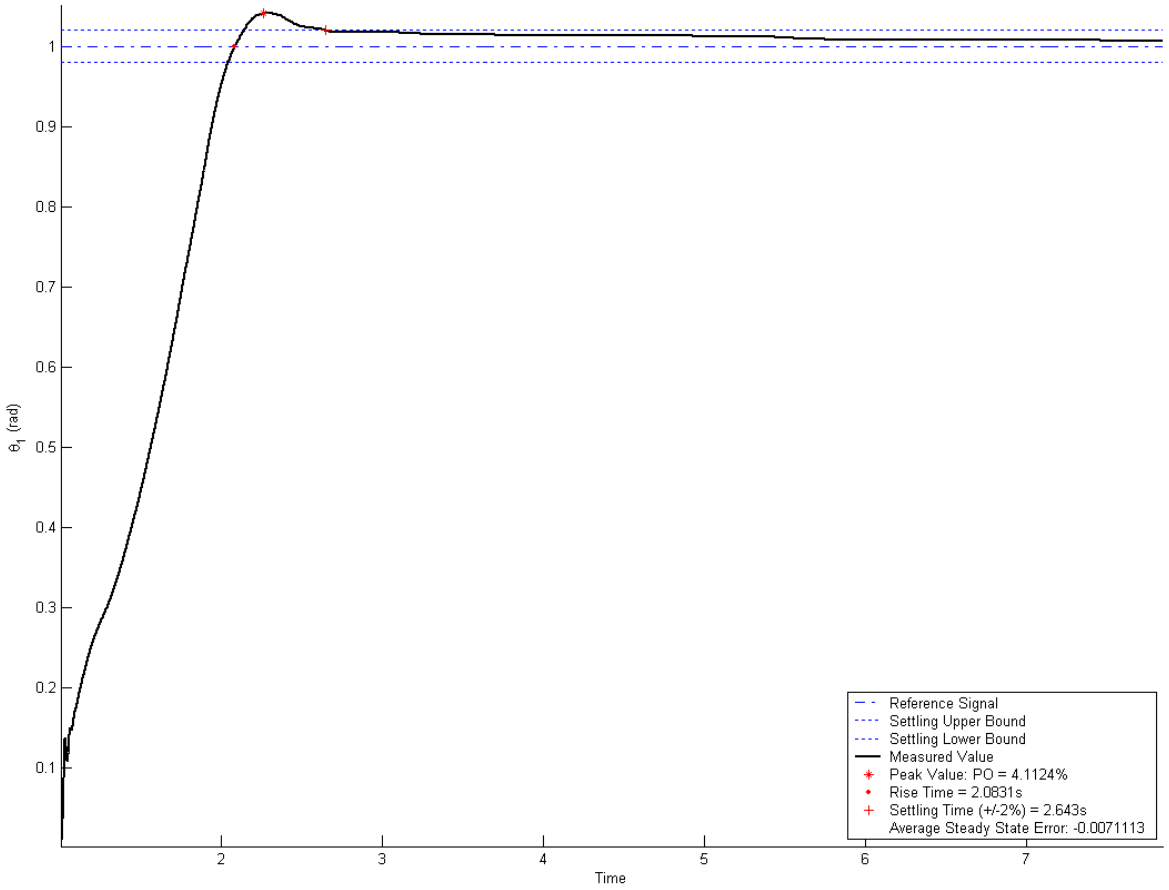


Figure 59 - PID position controller response to step input in the presence of constant T_L (Run 2).

Table 9 - Summary of PID position control performance.

T_L	PO	Rise Time	Settling Time	$Error_{ss}$
0 N-m	0.688%	2.17s	2.05s	-0.00165 rads
1 N-m	4.1124%	2.08s	2.64s	-0.00711 rads

At last true disturbance rejection is acquired. By inspecting the trend of the system's response to a disturbance, one can easily see the integral controller coming into play. The results shown in Table 9 are slightly misleading. Due to the manual nature of the applied torque disturbance, the applied load is not consistent. This effect causes the oscillatory behavior witnessed in the first portion of the system's response. Additionally, the steady-state error for both systems is calculated as an average error from the time the system reaches the commanded position ±2%, to when the run is halted.

This controller does an adequate job of responding to a step command and rejecting disturbances. However, one drawback that arises due to the introduction of the integral portion of the controller is the increase in oscillation and overshoot during the transient response. Even though, if robust disturbance rejection is desired then the integral component is a must.

e. Structural Mode Damping

Control of the load motor position from the drive motor via an aluminum shaft is accomplished through the flexi-shaft controller. This controller is evaluated in its step response both with and without the notch filter engaged. Figure 60 and Figure 61 show the system's response to the two test cases with the performance results being summarized in Table 10.

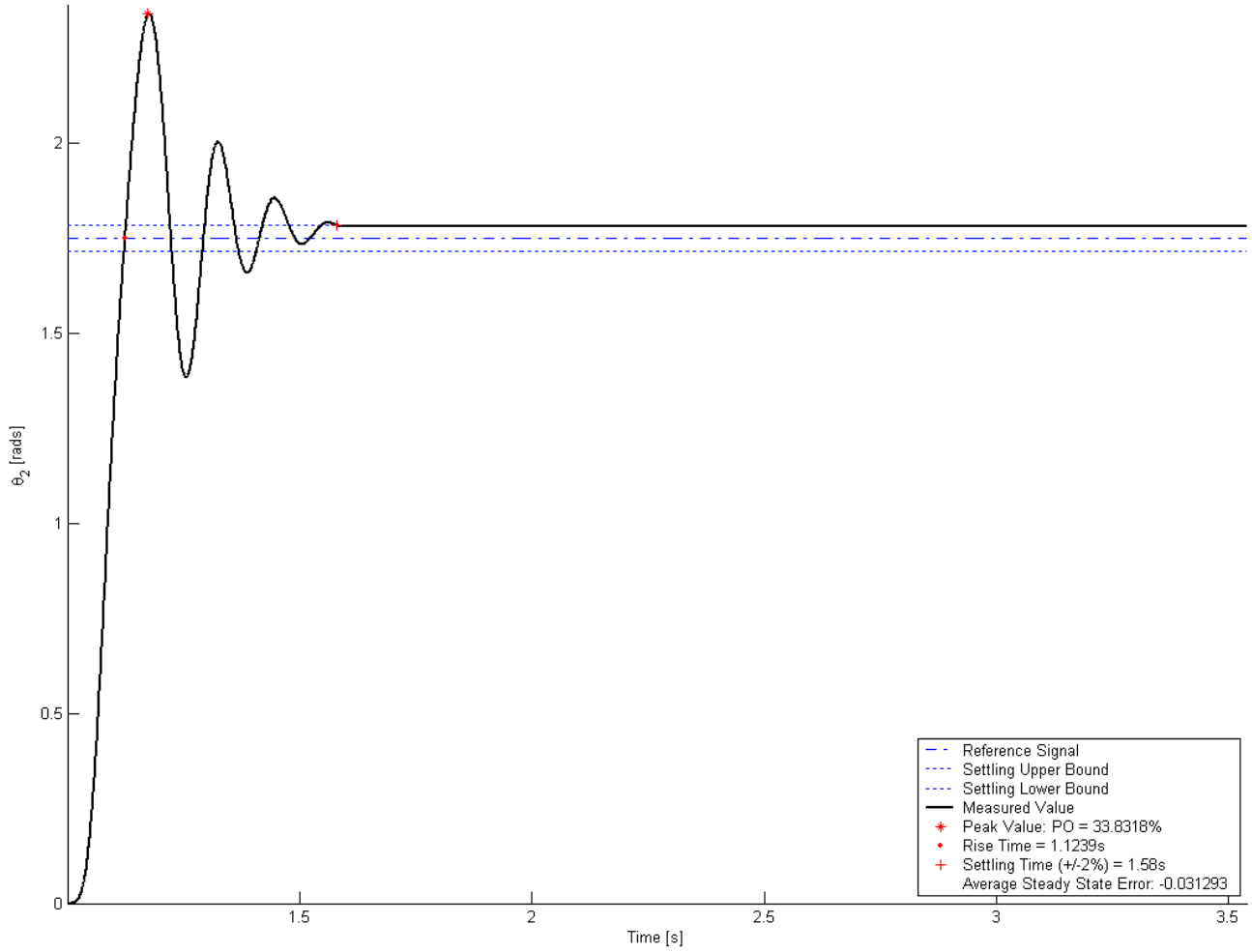


Figure 60 - Flexible shaft position controller step response using proportional-integral controller only; notch filter disengaged.

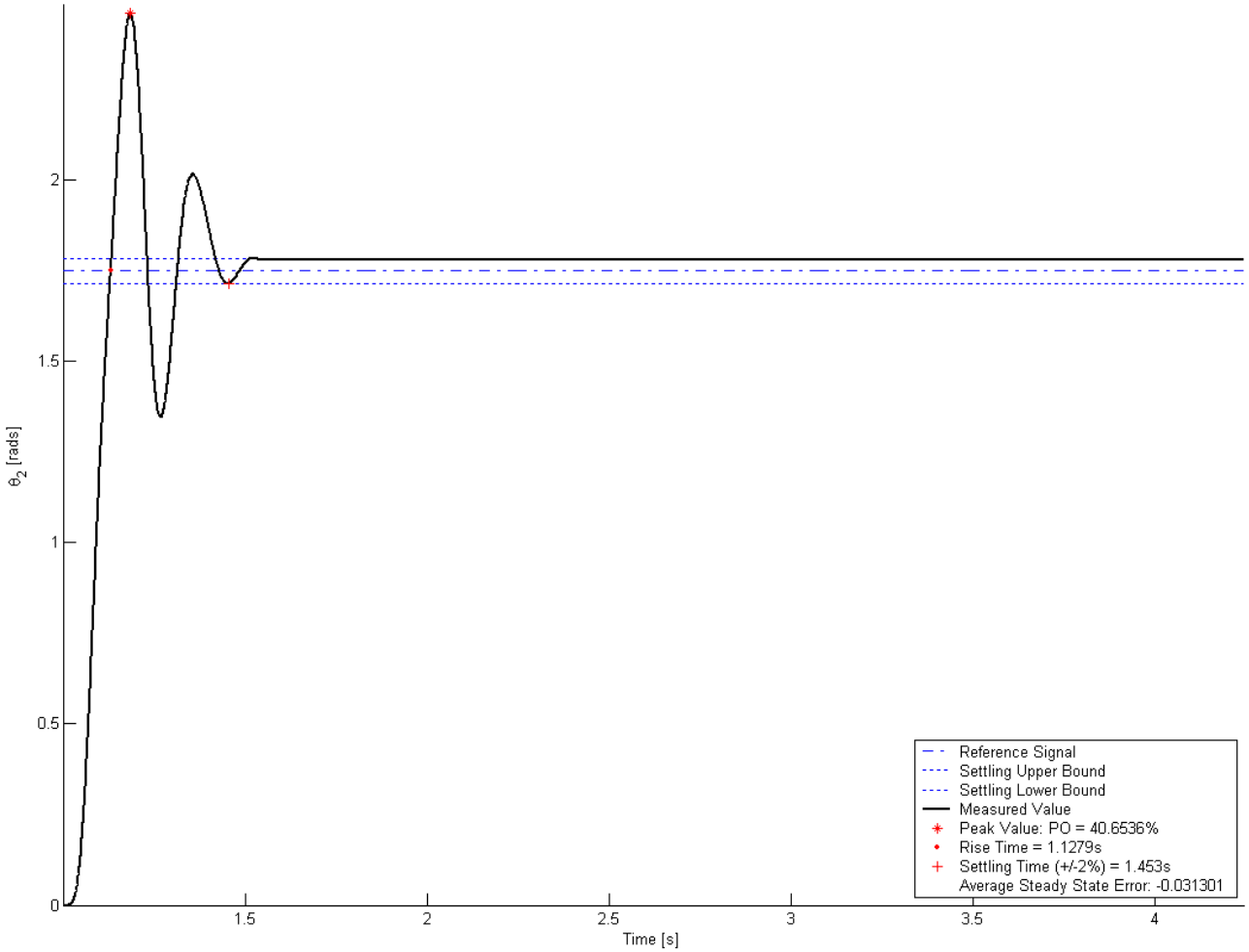


Figure 61 - Flexible shaft position controller with notch filter and PI control.

Table 10 - Summary of PID position control performance.

	Percent Overshoot	**Rise Time**	**Settling Time**	**Steady-State Error**
Notch Filter - Off	33.83%	1.124s	1.580s	-0.0313 rads
Notch Filter - On	40.65%	1.128s	1.453s	-0.0313 rads
Delta	*+20.16%*	*0.35%*	*-8.74%*	*0.00%*

The effect of the notch filter is clearly shown by examining the oscillation of the two responses. In the case where the notch filter is disengaged, the system oscillates at a significantly higher frequency for a longer duration. However, with the filter response one can see a slower frequency of oscillation.

Although the filter only provides an 9% decrease in the system's settling time, this is actually a significant change when one considers the fact that the controller is only a third order controller being applied to a sixth order system. This results in a reduced number of system states requiring observation and would

consequently provide a less costly and less complex implementation when compared to a sixth order controller.

When the system is deliberately excited at the shaft's natural modes, the effect of the filter becomes much more apparent. Figure 62 shows the response of the system with and without the notch filter when driven with a sine wave. The sine wave used had an amplitude of one radian, and a frequency of 60 rad/s. The frequency is approximately the first natural frequency of the aluminum shaft.

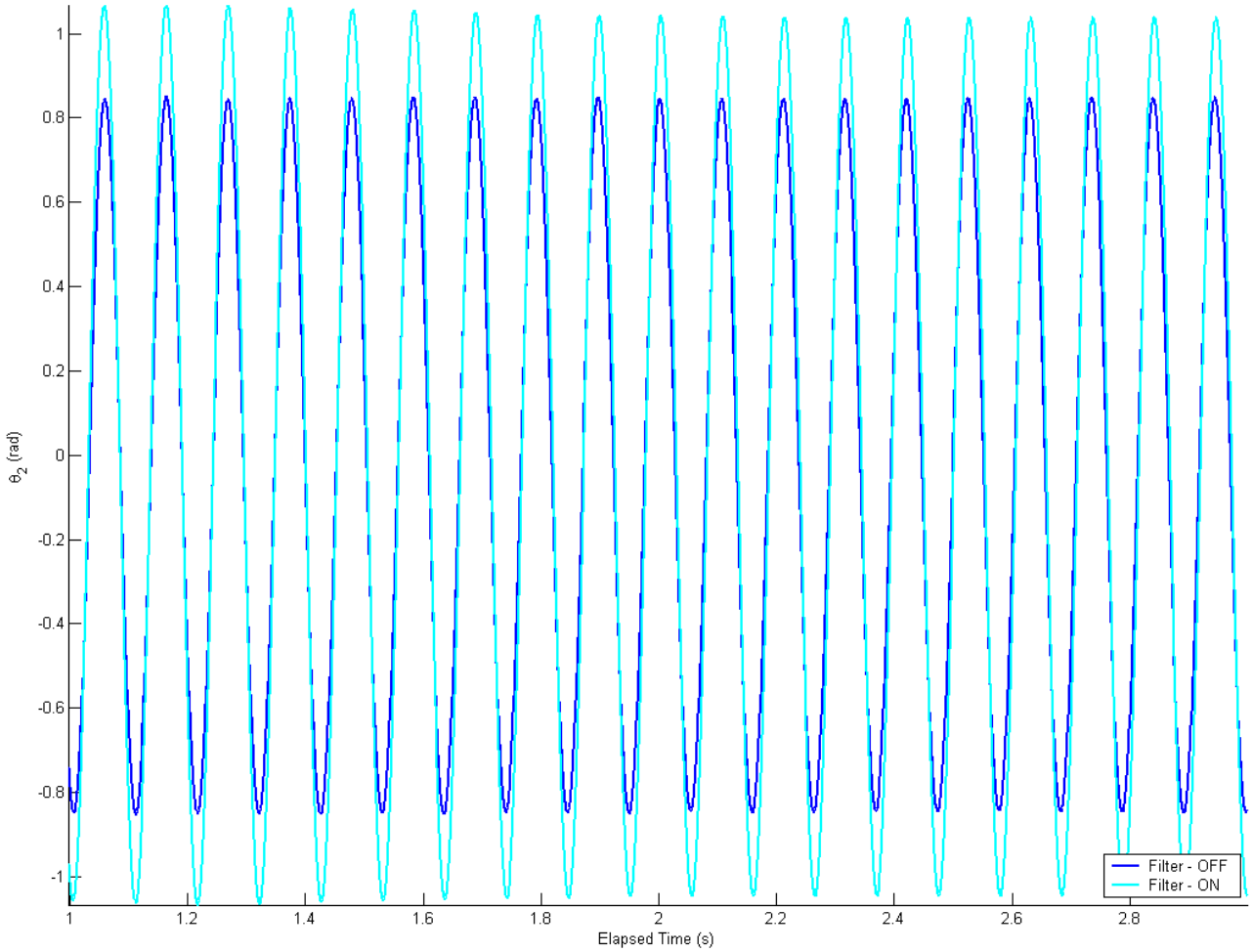


Figure 62 - System response to forced excitation of structural modes with a 60rad/s sine wave.

Inspection of the two responses shows that the filter provides a response of almost 2dB less than the unfiltered response. The major effect of this is that instead of magnifying the signal as is done without the filter, the signal is in fact attenuated when the filter is engaged.

Another dramatic indicator of the effect of this filter is visible when recording the response to a pulse disturbance. This disturbance was generated by manually moving the load motor and releasing in a rapid motion. Figure 63 shows

the system's response to this type of disturbance. Note that since the disturbance is manually applied the magnitude of the disturbance and response may vary.

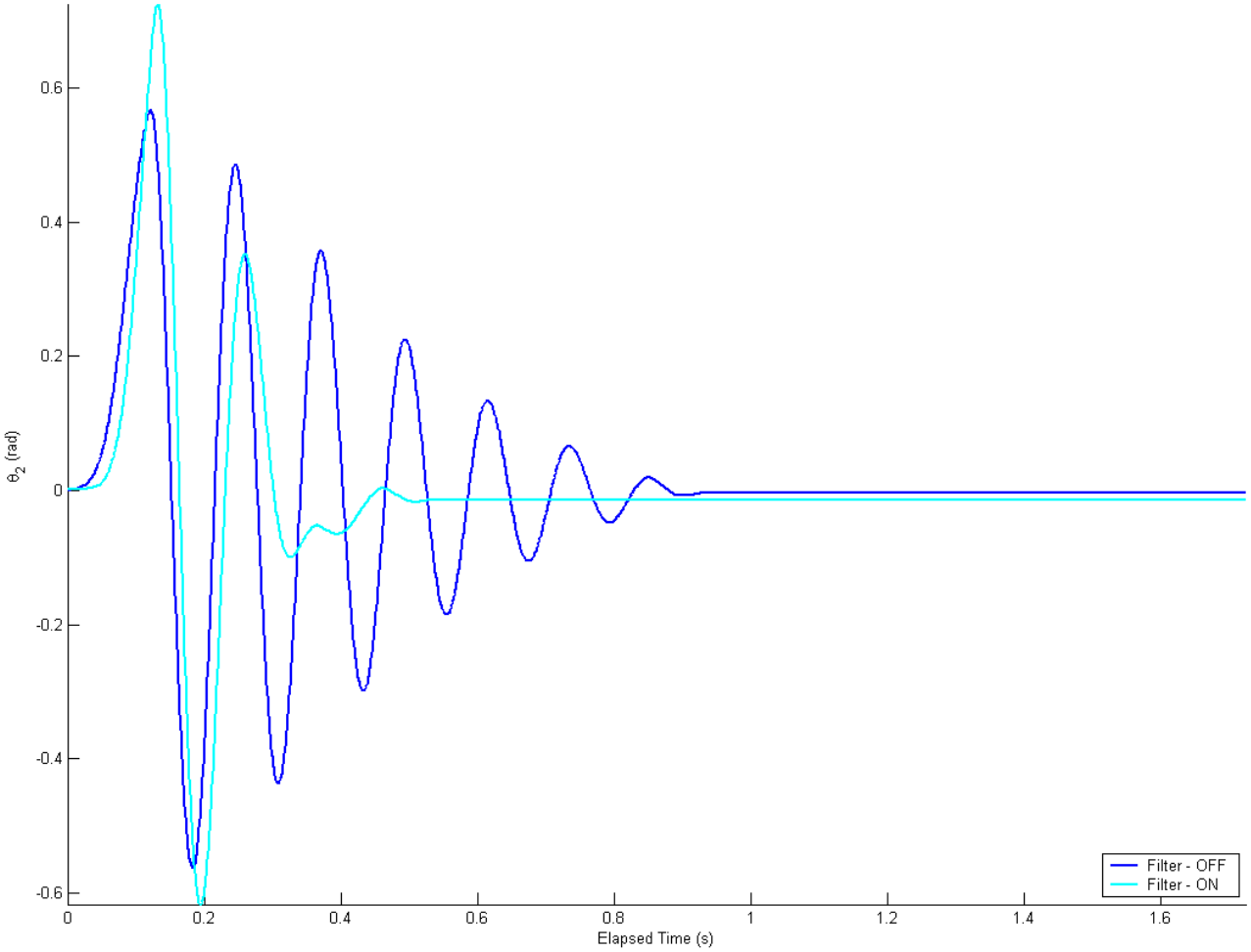


Figure 63 - System response to manual shock disturbance.

By examining these responses the filter's effect is obvious. In the case shown, the system was given a greater disturbance with the filter on and still returned to its steady-state in a shorter time and with fewer oscillations. In fact the response without the filter oscillates twice as much as the response with the filter on and takes almost twice as long to settle.

An additional benefit lies in the change of the oscillatory behavior of the system. Since the system responds with a slower oscillation, the overall intensity of the oscillation is reduced. This results in less likelihood of damage or excessive wear being incurred to the physical system. Of key importance is the impact to aluminum drive shaft. Failure of this component would render the system useless. Since the shaft's natural modes are not being excited the internal structure of the shaft is being subjected to less stress.

One shortcoming of this controller is that it is designed specifically for an aluminum drive shaft of a given size. Large changes in length or diameter of the shaft could shift the natural modes of the shaft out of the effect of the notch filter. This is almost a certainty if the material the shaft is made of is changed. If the shaft's natural modes of vibration were shifted outside the effectiveness of the notch filter, the system would suffer two effects. First, the additional oscillatory response due to excitation of the shaft's natural modes would return. Secondly, the system would attenuate and lose responsiveness in the band where the notch filter has effect.

Even with this potential shortfall, the notch filter is still an adequate control scheme for this situation. If applied to a real-world case, it is highly likely that the dimensions and composition of the drive shaft could be ensured and regulated to enough of an extent that the previously described situation would not occur. This would allow the notch filter design to be a simple, yet eloquent solution.

IX. Conclusions

In general, this lab was a successful and rewarding experience. Although most of the design was done online and in the laboratory, the results obtained experimentally could be reproduced and predicted with simulation. The major discrepancy between the lab results and the simulation was due to the lack of non-linear blocks to model effects such as stiction. This yields a simulation model that appears to be less stiff than the actual model.

The results and trends in the lab were repeatable and also agreed with the simulation even through the simulation neglected non-linear behavior. The notch filter implemented in part E functioned as designed and was able to increase the damping of the system. However, its effect was relatively small. There are many different ways in which the damping could be increased even more. One of these methods would be to use full state feedback, thus allow the poles of the system to be placed anywhere on the imaginary/real plane. The problem with this is that this requires 6 states to be fed back, which requires sensors for the current through the motor and estimators for the angular velocity. However, considering that the 6th order system could be approximated as a third order system, it may be possible to use state feedback with the sensors that are already in place on the model. This would not allow placement of all of the poles, but possibly it would allow the complex poles which limit damping to be moved to more desirable locations.

X. Appendix

Appendix A: PowerPoint Slides

Appendix B: MatLab Code

Section A and B

```
%Christopher Lum
%9923916
%AA448 Ly
%March 6, 2003
%
%------Lab 3:  Design and Control of a Flexishaft
System------------------
%
tic
clear
clc
close all

%Call matlab script given by Professor Ly to derive
state space models of
%the drive motor, load motor, and coupled system.
[Gdrive, Gload, Gfshaft, FS] = fshaft;
clc

disp('Do you want to display the graphs for
determining parameters?')
disp('  1 = yes')
disp('  2 = no')
disp('')
disp('')
graphselection = input('Enter selection:  ');




%----------------------Part A:  Open-Loop Control-----
------------------
disp('------------------------Open Loop Control------
----------------------')

%SECTION A.5

%load data.  Loads variables time, e_1, theta_1_rads,
and theta_2_rads
load run_1_A;
time_run_1 = time;
theta_1_run_1 = theta_1_rads;

load run_2_A;
time_run_2 = time;
theta_1_run_2 = theta_1_rads;

load run_3_A;
time_run_3 = time;
theta_1_run_3 = theta_1_rads;

%find which simulation ran the longest and had the
highest peak.
maxtime = max([max(time_run_1) max(time_run_2)
max(time_run_3)]);
maxpeak = max([max(theta_1_run_1) max(theta_1_run_2)
max(theta_1_run_3)]);


if graphselection==1
    figure

plot(time_run_1,theta_1_run_1,time_run_2,theta_1_run_2
,time_run_3,theta_1_run_3)
    title('Part A:  Actual \theta_1 vs. Time for Open
Loop Control System')
    xlabel('Time (sec)')
    ylabel('\theta_1 (rads)')
    legend('Run 1','Run 2','Run 3',4)
    axis([0 maxtime 0 maxpeak*1.1])
    grid
else
end



%Now that we have looked at the actual data, enter
parameters for
%simulation to verify answer.
TL = 0;                             %Nm
negative_dead_zone = -0.2;
positive_dead_zone = 0.2;
pulse_amplitude = 2.3;          %volts (used this in
lab week 2)
pulse_time = 0.177;             %sec (used this is lab
week 2)

sprintf('Dead zone for %2.2f < e_1 < %2.2f
volts.\nKPulse Amplitude (before amp) = %2.2f
volts.\nPulse Length = %2.2f
seconds.',negative_dead_zone,positive_dead_zone,pulse_
amplitude,pulse_time)

sim('parta_open_loop_control')

sim_time_A_6 = simTheta.time;
sim_theta_A_6 = simTheta.signals.values;
sim_e_1_cmd_A_6 = sime_1_cmd.signals.values;
sim_e_1_A_6 = sime_1.signals.values;

if graphselection==1
    figure

plot(time_run_1,theta_1_run_1,sim_time_A_6,sim_theta_A
_6)
    title('Part A.6:  \theta_1 vs. Time for Open Loop
Control System')
    xlabel('Time (sec)')
    ylabel('\theta_1 (radians)')
    legend('\theta_1 data','\theta_1
simulation','',['Pulse Amplitude (before amp) =
',num2str(pulse_amplitude),' Volts'],['Pulse time =
',num2str(pulse_time),' sec'],4)
    grid

    figure

plot(sim_time_A_6,sim_e_1_cmd_A_6,'rx',sim_time_A_6,si
m_e_1_A_6)
    title('Part A.6:  e_1_ _c_m_d and e_1 vs. Time for
Open Loop.')
    xlabel('Time (sec)')
    ylabel('Voltage before amp (volts)')
    legend('Command Voltage','Armature
Voltage','',['Pulse Amplitude (before amp) =
',num2str(pulse_amplitude),' Volts'],['Pulse time =
',num2str(pulse_time),' sec'])
    grid
else
end



%------Part B:  Position Control System:  Proportional
(P) Controller----
disp('------------------------Position Control
System----------------------')


%SECTION B.2.a
disp('SECTION B.2.a')

%Good tracking reponse with less that 20% overshoot

%Design for the simulated version that achieves this
performance.
%enter parameters
theta_cmd_initial = 0;      %radians
theta_cmd_final = 1;        %radians
TL = 0;                     %Nm

Maximum_PO = 20;            %percent

%We want to vary Kp to achieve less than the maximum
percent overshoot
Kp_start = 3.2;
Kp_end = 2.9;
Kp_interval = -0.01;

for Kp=Kp_start:Kp_interval:Kp_end
    %run simulation
    sim('partb_position_control')

    %extract simulated parameters
    sim_time_B_2 = simTheta.time;
    sim_theta_B_2 = simTheta.signals.values;
    sim_theta_cmd_B_2 = simThetaCmd.signals.values;

    %Calculate percent overshoot
    [max_peak,max_peak_index] = max(sim_theta_B_2);
    peak_time = sim_time_B_2(max_peak_index);

    PercentOvershoot =
CalculatePercentOvershoot(theta_cmd_initial,theta_cmd_
final,max_peak);

    if PercentOvershoot<Maximum_PO
        break
    else
    end
end


if graphselection==1
    figure

plot(sim_time_B_2,sim_theta_B_2,sim_time_B_2,sim_theta
_cmd_B_2,peak_time,max_peak,'ro')
```

```
    title('Part B.2a:  Simulated \theta_1 vs. time for
Proportional Feedback Only.')
    xlabel('Time (sec)')
    ylabel('\theta_1')
    legend('\theta_1','\theta_c_m_d',['Max Peak.  PO
=',num2str(PercentOvershoot,4),'%'],'',['K_p
=',num2str(Kp)],4)
    grid
else
end



%Now lets load the lab data
%RUN 1:  Kp = 3.5
load run_1_B;
time_run_1 = time;
theta_1_run_1 = theta_1_rads;
e_1_run_1 = e_1;                %voltage before amp
error_run_1 = theta_error_rads;

%RUN 2:  Kp = 3.2
load run_2_B;
time_run_2 = time;
theta_1_run_2 = theta_1_rads;
e_1_run_2 = e_1;
error_run_2 = theta_error_rads;

%RUN 3:  Kp = 3.0
load run_3_B;
time_run_3 = time;
theta_1_run_3 = theta_1_rads;
e_1_run_3 = e_1;
error_run_3 = theta_error_rads;

%FUN 4:  Kp = 3.1
load run_4_B;
time_run_4 = time;
theta_1_run_4 = theta_1_rads;
e_1_run_4 = e_1;
error_run_4 = theta_error_rads;

%Find the steady state error of the lab data with Kp =
3.1
steady_state_error_B_2 =
error_run_4(length(error_run_4));

disp(' ')
disp('Actual Conditions')
disp(['With Kp = ',num2str(Kp),' and TL =
',num2str(TL),', Maximum steady state error =
',num2str(steady_state_error_B_2)])
disp(' ')

%Run the simulation with the value of Kp used in Lab
Kp = 3.1;
TL = 0;
sim('partb_position_control');

sim_time_B_2 = simTheta.time;
sim_theta_B_2 = simTheta.signals.values;
sim_theta_cmd_B_2 = simThetaCmd.signals.values;
sim_e_1_B_2 = sime_1.signals.values;
sim_theta_error = simThetaError.signals.values;

%Find the simulated steady state error with Kp = 3.1
sim_steady_state_error_B_2 =
sim_theta_error(length(sim_theta_error));

if sim_steady_state_error_B_2<0.00001
    sim_steady_state_error_B_2 = 0;
else
end

disp(' ')
disp('Simulated Conditions')
disp(['With Kp = ',num2str(Kp),' and TL =
',num2str(TL),', Maximum steady state error =
',num2str(sim_steady_state_error_B_2)])
disp(' ')

if graphselection==1
    figure

plot(sim_time_B_2,sim_theta_cmd_B_2,time_run_2,theta_1
_run_2,sim_time_B_2,sim_theta_B_2)
    title('Part B.2a:  Actual and Simulated \theta_1
vs. Time for Proportional Feedback Only')
    xlabel('Time (sec)')
    ylabel('\theta_1 (rads)')
    legend('\theta_{cmd}','Actual \theta_1','Simulated
\theta_1',['Actual Error_s_s =
',num2str(steady_state_error_B_2),' rads'],['Simulated
Error_s_s = ',num2str(sim_steady_state_error_B_2),'
rads'],'',['K_p = ',num2str(Kp)],['T_L
=',num2str(TL),' Nm'],4)
    grid

    figure

plot(time_run_2,e_1_run_2,sim_time_B_2,sim_e_1_B_2)
    title('Part B.2a:  Actual and Simulated e_1 vs.
Time for Proportional Feedback Only')
    xlabel('Time (sec)')
    ylabel('e_1 (volts)')
    legend('Data','Simulation','',['T_L =
',num2str(TL)],['K_p = ',num2str(Kp)])
    grid
else
end


%SECTION B.2.d
disp(' ')
disp('SECTION B.2.d')
disp(' ')

%Using the Kp gain found above, apply a disturbance
torque and find steady state error
%We actually applied a torque in the negative
direction since the torque we
%applied resisted the initial motion.
Kp = Kp;
TL = -1;         %Nm
sim('partb_position_control')
sim_time_B_2 = simTheta.time;
sim_theta_B_2 = simTheta.signals.values;
sim_theta_cmd_B_2 = simThetaCmd.signals.values;
sim_theta_error_B_2 = simThetaError.signals.values;
sim_e_1_B_2 = sime_1.signals.values;

%Calculate the simulated steady state error
sim_steady_state_error_B_2 =
sim_theta_error_B_2(length(sim_theta_error_B_2));
disp(' ')
disp('Simulated Conditions')
disp(['With Kp = ',num2str(Kp),' and TL =
',num2str(TL),', Maximum steady state error =
',num2str(sim_steady_state_error_B_2)])
disp(' ')


%Load the lab data
load run_5_B;
time_run_5 = time;
theta_1_run_5 = theta_1_rads;
e_1_run_5 = e_1;
error_run_5 = theta_error_rads;

%Calculate the steady state error
steady_state_error_B_2 =
error_run_5(length(error_run_5));
disp(' ')
disp('Actual Conditions')
disp(['With Kp = ',num2str(Kp),' and TL =
',num2str(TL),', Maximum steady state error =
',num2str(steady_state_error_B_2)])
disp(' ')

if graphselection==1
    figure

plot(sim_time_B_2,sim_theta_cmd_B_2,time_run_5,theta_1
_run_5,sim_time_B_2,sim_theta_B_2)
    title('Part B.2d:  Actual Simulated \theta_1 vs.
Time for Proportional Feedback Only with T_L')
    xlabel('Time (sec)')
    ylabel('\theta_1')
    legend('\theta_{cmd}','Actual \theta_1','Simulated
\theta_1',['Actual Error_s_s =
',num2str(steady_state_error_B_2),' rads'],['Simulated
Error_s_s = ',num2str(sim_steady_state_error_B_2),'
rads'],['K_p = ',num2str(Kp)],['T_L =',num2str(TL),'
Nm'])
    grid

    figure

plot(time_run_5,e_1_run_5,sim_time_B_2,sim_e_1_B_2)
    title('Part B.2d:  Actual and Simulated e_1 vs.
Time for Proportional Feedback Only with T_L')
    xlabel('Time (sec)')
    ylabel('e_1 (volts)')
    legend('Data','Simulation','',['T_L =
',num2str(TL)],['K_p = ',num2str(Kp)])
    grid
else
end




%SECTION B.5
disp('SECTION B.5')

%Experimentally find the Kp that causes 0.1 radians
with TL = -1.
%RUN 7:  Kp = 16
load run_7_B;
time_run_7 = time;
theta_1_run_7 = theta_1_rads;
e_1_run_7 = e_1;
error_run_7 = theta_error_rads;

%Run simulation to verify this
TL = -1;
Kp = 16;
sim('partb_position_control')
sim_time_B_5 = simTheta.time;
```

```
sim_theta_B_5 = simTheta.signals.values;
sim_theta_cmd_B_5 = simThetaCmd.signals.values;
sim_theta_error_B_5 = simThetaError.signals.values;
sim_e_1_B_5 = sime_1.signals.values;

%Calculate the simulated steady state error
sim_steady_state_error_B_5 =
sim_theta_error_B_5(length(sim_theta_error_B_5));
disp(' ')
disp('Simulated Conditions')
disp(['With Kp = ',num2str(Kp),' and TL =
',num2str(TL),', Maximum steady state error =
',num2str(sim_steady_state_error_B_5)])
disp(' ')

%Calculate the steady state error
steady_state_error_B_5 =
error_run_7(length(error_run_7));
disp(' ')
disp('Actual Conditions')
disp(['With Kp = ',num2str(Kp),' and TL =
',num2str(TL),', Maximum steady state error =
',num2str(steady_state_error_B_5)])
disp(' ')



if graphselection==1

plot(sim_time_B_5,sim_theta_cmd_B_5,time_run_7,theta_1
_run_7,sim_time_B_5,sim_theta_B_5)
    title('Part B.5:  Actual Simulated \theta_1 vs.
Time for Proportional Feedback Only with T_L')
    xlabel('Time (sec)')
    ylabel('\theta_1')
    legend('\theta_{cmd}','Actual \theta_1','Simulated
\theta_1','',['Actual Error_s_s =
',num2str(steady_state_error_B_5),' rads'],['Simulated
Error_s_s = ',num2str(sim_steady_state_error_B_5),'
rads'],'',['K_p = ',num2str(Kp)],['T_L
=',num2str(TL),' Nm'])
    grid

    figure

plot(time_run_7,e_1_run_7,sim_time_B_5,sim_e_1_B_5)
    title('Part B.5:  Actual and Simulated e_1 vs.
Time for Proportional Feedback Only with T_L')
    xlabel('Time (sec)')
    ylabel('e_1 (volts)')
    legend('Data','Simulation','',['T_L =
',num2str(TL)],['K_p = ',num2str(Kp)])
    grid
else
end



%Find transfer function b/w theta_1 and TL.
[a_B_5,b_B_5,c_B_5,d_B_5]=linmod('partb_position_contr
ol');

%We want between input 2 (TL) and output 1 (theta_1)
b_second_input_B_5 = b_B_5(:,2);
c_first_output_B_5 = c_B_5(1,:);
d_good_B_5 = d_B_5(1,2);

[num_B_5,den_B_5] =
ss2tf(a_B_5,b_second_input_B_5,c_first_output_B_5,d_go
od_B_5);

sprintf('Transfer Function (theta_1/TL) without
Modification')
%transfer function without modification
G_B_5_bad = tf(num_B_5,den_B_5)

%calculate the poles and zeros of this transfer
function
zeros_B_5_bad = roots(num_B_5)
poles_B_5_bad = roots(den_B_5)

%Lets modify the transfer function slightly.
tolerance_B_5 = 0.001;
[clean_num_B_5,clean_den_B_5] =
CleanUpTransferFunction(num_B_5,den_B_5,tolerance_B_5)
;

sprintf('Removing any poles/zeros smaller than
%1.6f.\n\nModified Transfer Function
(theta_1/TL)',tolerance_B_5)
G_B_5_good = tf(clean_num_B_5,clean_den_B_5)

%Calculate the step response
[step_response_B_5,time_B_5] = step(G_B_5_good);

%Keep in mind that this is the response to a unit
step, if we desire to
%apply a torque of TL = 1 Nm, then we need to multiply
the output by 1.
step_response_B_5 = step_response_B_5*TL;

%Calculate the poles and zeros of this transfer
function
zeros_B_5_good = roots(clean_num_B_5)
poles_B_5_good = roots(clean_den_B_5)

%now compare with simulation

TL = -1;
Kp = 16;
theta_cmd_initial = 0;
theta_cmd_final = 0;
sim('partb_position_control')

sim_time_B_5 = simTheta.time;
sim_theta_B_5 = simTheta.signals.values;
sim_TL_B_5 = simTL.signals.values;

%find the final value
final_value_B_5 =
sim_theta_B_5(length(sim_theta_B_5));

if graphselection==1
    figure

plot(time_B_5,step_response_B_5,sim_time_B_5,sim_theta
_B_5)
    title('Part B5:  Step Response of System to T_L')
    xlabel('Time (sec)')
    ylabel('\theta_1 (radians)')
    legend('Calculated Step Reponse','Simulated
Response',['Kp =',num2str(Kp)],['T_L
=',num2str(TL)],['Final Value =
',num2str(final_value_B_5),'rads'],4)
    grid
else
end



%SECTION B.6
disp('SECTION B.6')
disp('Vary Kp to see how steady state error varies
with a constant TL')
%We want to modify the gain Kp and see how this
changes the
%steady state error.
toc
start_Kp = 1;
end_Kp = 20;
Kp_interval = 0.5;
Kp_array = [start_Kp:Kp_interval:end_Kp];

TL = -1;
theta_cmd_initial = 0;
theta_cmd_final = 1;


%Start a loop that varies Kp and records error.
for counter=1:length(Kp_array)
    Kp = Kp_array(counter);

    sim('partb_position_control')

    sim_time_B_6 = simTheta.time;
    sim_theta_B_6 = simTheta.signals.values;
    sim_TL_B_6 = simTL.signals.values;
    sim_theta_error_B_6 =
simThetaError.signals.values;

    steady_state_error_B_6 =
sim_theta_error_B_6(length(sim_theta_error_B_6));

    data_array_B_6(counter,1) = Kp;
    data_array_B_6(counter,2) =
steady_state_error_B_6;
end
toc
%find where the the error drops below 0.1 rad.
tolerance_B_6 = 0.1;        %radians

steady_state_error_B_6 = abs(data_array_B_6(:,2));

for counter=1:length(steady_state_error_B_6)
    if steady_state_error_B_6(counter)<tolerance_B_6
        break
    else
    end
end

Kp_critical = data_array_B_6(counter,1);
steady_state_error_B_6_critical =
data_array_B_6(counter,2);


if graphselection==1
    figure

plot(data_array_B_6(:,1),data_array_B_6(:,2),Kp_critic
al,steady_state_error_B_6_critical,'rx')
    title('Part B6:  Steady State Error vs. K_p with a
constant T_L')
    xlabel('K_p')
    ylabel('Steady State Error (radians)')
    legend('Steady State Error',['Critical Value.
Error =
',num2str(steady_state_error_B_6_critical,3),'rads'],'
',['T_L = ',num2str(TL),' Nm'],['K_p _c_r_i_t_i_c_a_l
= ',num2str(Kp_critical)],4)
    grid
```

```
else
end

%Simulate the response with the critical values
Kp = Kp_critical;
TL = -1;
theta_cmd_initial = 0;
theta_cmd_final = 1;

sim('partb_position_control')

sim_time_B_6 = simTheta.time;
sim_theta_B_6 = simTheta.signals.values;
sim_TL_B_6 = simTL.signals.values;
sim_theta_error_B_6 = simThetaError.signals.values;

%Find the overshoot with this Kp value
[Peak_B_6,PO_B_6_index] = max(sim_theta_B_6);
PO_time_B_6 = sim_time_B_6(PO_B_6_index);

PO_B_6 = (Peak_B_6 -
theta_cmd_final)/(theta_cmd_final-
theta_cmd_initial)*100;

if graphselection==1
    figure

plot(sim_time_B_6,sim_theta_B_6,PO_time_B_6,Peak_B_6,'
ro')
    title('Part B6:  Simulated \theta_1 vs. Time
Response to Constant Torque Only')
    xlabel('Time (sec)')
    ylabel('\theta_1 (radians)')
    legend('\theta_1',['Max Peak.  PO =
',num2str(PO_B_6),'%'],'',['T_L = ',num2str(TL),'
Nm'],['K_p = ',num2str(Kp)],0)
    grid
else
end

sprintf('With TL = %2.2f Nm, to obtain steady state
error less than %2.2f following parameters are
needed:\n\nKp = %2.2f   (yields PO = %2.2f
percent)',TL,tolerance_B_6,Kp_critical,PO_B_6)



%Now see what happens if we try to reject a 2Nm
torque.  This requires a Kp
%= 32.2.  What happens if we use this gain with no
torque
Kp = 32.2;
TL = 0;
theta_cmd_initial = 0;
theta_cmd_final = 1;

sim('partb_position_control')

sim_time_B_6_2 = simTheta.time;
sim_theta_B_6_2 = simTheta.signals.values;
sim_TL_B_6_2 = simTL.signals.values;
sim_theta_error_B_6_2 = simThetaError.signals.values;
sim_e_1_B_6_2 = sime_1.signals.values;
sim_theta_cmd_B_6_2 = simThetaCmd.signals.values;

if graphselection==1
    figure

plot(sim_time_B_6_2,sim_theta_cmd_B_6_2,sim_time_B_6_2
,sim_theta_B_6_2)
    title('Part B6:  Simulated \theta_1 vs. Time
Response with High Gain')
    xlabel('Time (sec)')
    ylabel('\theta_1 (radians)')
    legend('\theta_{ref}','\theta_1','',['T_L =
',num2str(TL),' Nm'],['K_p = ',num2str(Kp)])
    grid
else
end




%SECTION B.8
disp('SECTION B.8')

%We want to see if the armature voltage exceeds 10
volts during the
%simulation.

%In part B.2, we found by simulation that the gain of
Kp = 3.1 is the
%necessary gain to create less than 20% overshoot
TL = 0;
Kp = 3.1;
theta_cmd_initial = 0;
theta_cmd_final = 1;

sim('partb_position_control')

sim_time_B_8 = simTheta.time;
sim_theta_B_8 = simTheta.signals.values;
sim_TL_B_8 = simTL.signals.values;
sim_theta_error_B_8 = simThetaError.signals.values;
sim_e_1_B_8 = sime_1.signals.values;

%Find the maximum voltage
[max_e_1_B_8,max_e_1_B_8_index] = max(sim_e_1_B_8);
max_e_1_time_B_8 = sim_time_B_8(max_e_1_B_8_index);

disp(' ')
disp(['With B.2 parameters, maximum e_1 =
',num2str(max_e_1_B_8)])
disp(' ')

if graphselection==1
    figure

plot(sim_time_B_8,sim_e_1_B_8,max_e_1_time_B_8,max_e_1
_B_8,'ro')
    title('Part B8:  Simulated e_1 vs. Time for
Position Control.  Using B.2 Parameters')
    xlabel('Time (sec)')
    ylabel('e_1 (volts)')
    legend('e_1(t)',['Max Peak =
',num2str(max_e_1_B_8),' volts'],'',['T_L =
',num2str(TL),' Nm'],['K_p = ',num2str(Kp)])
    grid
else
end

%In part B.6, we found that with the TL = -1 Nm, we
need a gain of Kp = 16
%was needed to obtain less that 0.1 rads
TL = -1;
Kp = 16;
theta_cmd_initial = 0;
theta_cmd_final = 1;

disp(' ')
sim('partb_position_control')
disp(' ')

sim_time_B_8 = simTheta.time;
sim_theta_B_8 = simTheta.signals.values;
sim_TL_B_8 = simTL.signals.values;
sim_theta_error_B_8 = simThetaError.signals.values;
sim_e_1_B_8 = sime_1.signals.values;

%Find the maximum voltage
[max_e_1_B_8,max_e_1_B_8_index] = max(sim_e_1_B_8);
max_e_1_time_B_8 = sim_time_B_8(max_e_1_B_8_index);

disp(' ')
disp(['With B.6 parameters, maximum e_1 =
',num2str(max_e_1_B_8)])
disp(' ')

if graphselection==1
    figure

plot(sim_time_B_8,sim_e_1_B_8,max_e_1_time_B_8,max_e_1
_B_8,'ro')
    title('Part B8:  Simulated e_1 vs. Time for
Position Control.  Using B.6 Parameters')
    xlabel('Time (sec)')
    ylabel('e_1 (volts)')
    legend('e_1(t)',['Max Peak =
',num2str(max_e_1_B_8),' volts'],'',['T_L =
',num2str(TL),' Nm'],['K_p = ',num2str(Kp)])
    grid
else
end



%As can be seen, the voltage exceeds 10 volts (also
shown in section B5).
%Therefore we need to explore the use of both a
saturation block and a
%low-pass filter.  First we can use the saturation
block.
disp(' ')
disp('Explore the use of the saturation block.')
disp(' ')

Kp = 16;
TL = -1;
sim('partb_position_control_with_saturation')

sim_time_B_8sat = simTheta.time;
sim_theta_B_8sat = simTheta.signals.values;
sim_TL_B_8sat = simTL.signals.values;
sim_theta_error_B_8sat = simThetaError.signals.values;
sim_e_1_B_8sat = sime_1.signals.values;
sim_theta_cmd_B_8sat = simThetaCmd.signals.values;

%Find the maximum voltage
[max_e_1_B_8sat,max_e_1_B_8_index] =
max(sim_e_1_B_8sat);
max_e_1_time_B_8sat =
sim_time_B_8sat(max_e_1_B_8_index);

disp(' ')
disp(['With B.6 parameters using a saturation block,
maximum e_1 = ',num2str(max_e_1_B_8sat)])
disp(' ')

if graphselection==1
    figure

plot(sim_time_B_8sat,sim_theta_cmd_B_8sat,sim_time_B_8
sat,sim_theta_B_8sat)
    title('Part B8b:  Simulated \theta_1 vs. Time
using Saturation Block')
```

```
    xlabel('Time (sec)')
    ylabel('\theta_1 (radians)')
    legend('\theta_{cmd}','\theta_1','',['T_L =
',num2str(TL),' Nm'],['K_p = ',num2str(Kp)])
    grid

    figure

plot(sim_time_B_8sat,sim_e_1_B_8sat,max_e_1_time_B_8sa
t,max_e_1_B_8sat,'ro')
    title('Part B8a:  Simulated e_1 vs. Time for
Position Control.  Using B.6 Parameters with
Saturation Block')
    xlabel('Time (sec)')
    ylabel('e_1 (volts)')
    legend('e_1(t)',['Max Peak =
',num2str(max_e_1_B_8sat),' volts'],'',['T_L =
',num2str(TL),' Nm'],['K_p = ',num2str(Kp)])
    grid
else
end



%Now see what happens if we use the low-pass filter
disp(' ')
disp('Explore the use of the low-pass filter block.')
disp(' ')

a = 50;
Kp = 16;
TL = -1;
sim('partb_position_control_with_filter')

sim_time_B_8filter = simTheta.time;
sim_theta_B_8filter = simTheta.signals.values;
sim_TL_B_8filter = simTL.signals.values;
sim_theta_error_B_8filter =
simThetaError.signals.values;
sim_e_1_B_8filter = sime_1.signals.values;
sim_theta_cmd_B_8filter = simThetaCmd.signals.values;
sim_theta_ref_B_8filter = simThetaRef.signals.values;

%Find the maximum voltage
[max_e_1_B_8filter,max_e_1_B_8_index] =
max(sim_e_1_B_8filter);
max_e_1_time_B_8filter =
sim_time_B_8filter(max_e_1_B_8_index);

disp(' ')
disp(['With B.6 parameters using a low-pass filter
with a = ',num2str(a),', maximum e_1 =
',num2str(max_e_1_B_8filter)])
disp(' ')

if graphselection==1
    figure

plot(sim_time_B_8filter,sim_theta_cmd_B_8filter,sim_ti
me_B_8filter,sim_theta_ref_B_8filter,sim_time_B_8filte
r,sim_theta_B_8filter)
    title('Part B8b:  Simulated \theta_1 vs. Time
using Prefilter')
    xlabel('Time (sec)')
    ylabel('\theta_1 (radians)')

legend('\theta_{cmd}','\theta_{ref}','\theta_1','',['T
_L = ',num2str(TL),' Nm'],['K_p =
',num2str(Kp)],['\omega_{break} = ',num2str(a),'
rad/s'])
    grid

    figure

plot(sim_time_B_8filter,sim_e_1_B_8filter,max_e_1_time
_B_8filter,max_e_1_B_8filter,'ro')
    title('Part B8b:  Simulated e_1 vs. Time for
Position Control.  Using B.6 Parameters with Low-Pass
Filter')
    xlabel('Time (sec)')
    ylabel('e_1 (volts)')
    legend('e_1(t)',['Max Peak =
',num2str(max_e_1_B_8filter),' volts'],'',['T_L =
',num2str(TL),' Nm'],['K_p =
',num2str(Kp)],['\omega_{break} = ',num2str(a),'
rad/s'])
    grid
else
end
```

toc

Section C

```
%----------------------Part C:  P/D Control-----------
-----------------


clear
clc
close all

[Gdrive, Gload, Gfshaft, FS] = fshaft;
clc

disp('Do you want to display the graphs for
determining parameters?')
disp('  1 = yes')
disp('  2 = no')
disp('')
disp('')
graphselection = input('Enter selection:  ');

%SECTION C.1
%Load the lab data

load run_3_C

%Enter parameters
TL = 0;
theta_cmd_initial = 0;
theta_cmd_final = 1;
a = 100;

Kp = 12;
Kd = .2;

sim('partc_position_control_PD')

sim_time_C_1 = simTheta.time;
sim_theta_C_1 = simTheta.signals.values;
sim_theta_cmd_C_1 = simThetaCmd.signals.values;

if graphselection==1
    figure

plot(sim_time_C_1,sim_theta_cmd_C_1,time,theta_1_rads,
sim_time_C_1,sim_theta_C_1)
    title('Actual and Simulated \theta_1 vs. Time for
PD Control System')
    xlabel('Time (sec)')
    ylabel('\theta_1 (radians)')
    legend('\theta_{1 cmd}','\theta_1
actual','\theta_1 simulated','',...
        ['K_P = ',num2str(Kp)],['K_D =
',num2str(Kd)],['a = ',num2str(a),' rad/s'],...
        ['T_L = ',num2str(TL),' Nm'],4)
    axis([0.75 1.75 0 1.2])
    grid
else
end


disp('Section C.3.c')
%load actual run
load run_4_C

TL = -1;
theta_cmd_initial = 0;
theta_cmd_final = 1;
a = 100;
Kp = 12;
Kd = .2;

sim('partc_position_control_PD')

sim_time_C_3 = simTheta.time;
sim_theta_C_3 = simTheta.signals.values;
sim_theta_cmd_C_3 = simThetaCmd.signals.values;
sim_error_C_3 = simThetaError.signals.values;
sim_e_1_C_3 = sime_1.signals.values;

sim_error = sim_error_C_3(length(sim_error_C_3));

disp(['With constant torque = ',num2str(TL),' Nm.
Steady state error = ',...
        num2str(sim_error),' radians'])

%find the steady state error

if graphselection==1
    figure

plot(sim_time_C_3,sim_theta_cmd_C_3,time,theta_1_rads,
...
        sim_time_C_3,sim_theta_C_3)
    title('Actual and Simulated \theta_1 vs. Time for
PD Control with Constant Torque')
    xlabel('Time (sec)')
    ylabel('\theta_1 (radians)')
    legend('\theta_{1 cmd}','\theta_1
actual','\theta_1 simulated','',...
        ['K_P = ',num2str(Kp)],['K_D =
',num2str(Kd)],['a = ',num2str(a),' rad/s'],...
        ['T_L = ',num2str(TL),' Nm'],4)
    grid

    figure
    plot(time,e_1,sim_time_C_3,sim_e_1_C_3)
    title('Actual and Simulated e_1 vs. Time for PD
Control with Constant Torque')
    xlabel('Time (sec)')
    ylabel('e_1 (volts)')
    legend('e_1 Actual','e_1 Simulated')
    grid
else
end
```

Section D

```
%----------------------Part D:  PID Control-----------
-----------------


clear
clc
close all

[Gdrive, Gload, Gfshaft, FS] = fshaft;
clc

disp('Do you want to display the graphs for
determining parameters?')
disp('  1 = yes')
disp('  2 = no')
disp('')
disp('')
graphselection = input('Enter selection:  ');

%SECTION D.1
%Load the lab data for step with no torque

load run_3_D

%Enter parameters
TL = 0;
theta_cmd_initial = 0;
theta_cmd_final = 1;
a = 100;

Kp = 12;
Kd = .2;
Ki = 3;

sim('partd_position_control_PID',[0 20])

sim_time_D_1a = simTheta.time;
sim_theta_D_1a = simTheta.signals.values;
sim_theta_cmd_D_1a = simThetaCmd.signals.values;

if graphselection==1
    figure

plot(sim_time_D_1a,sim_theta_cmd_D_1a,time,theta_1_rad
s,...
        sim_time_D_1a,sim_theta_D_1a)
    title('Actual and Simulated \theta_1 vs. Time for
PID Control')
    xlabel('Time (sec)')
    ylabel('\theta_1 (radians)')
    legend('\theta_{1 cmd}','\theta_1
actual','\theta_1 simulated','',...
        ['K_P = ',num2str(Kp)],['K_D =
',num2str(Kd)],...
        ['K_I = ',num2str(Ki)],['a = ',num2str(a),'
rad/s'],...
        ['T_L = ',num2str(TL),' Nm'],4)
    axis([0.9 1.5 0 1.2])
    grid
else
end


%constant torque
load run_1_D

TL = -1;
theta_cmd_initial = 0;
theta_cmd_final = 1;
a = 100;

Kp = 12;
Kd = .2;
Ki = 3;

sim('partd_position_control_PID',[0 20])

sim_time_D_1 = simTheta.time;
sim_theta_D_1 = simTheta.signals.values;
sim_theta_cmd_D_1 = simThetaCmd.signals.values;

if graphselection==1
    figure

plot(sim_time_D_1,sim_theta_cmd_D_1,time,theta_1_rads,
...
        sim_time_D_1,sim_theta_D_1)
    title('Actual and Simulated \theta_1 vs. Time for
PID Control with Constant T_L')
    xlabel('Time (sec)')
    ylabel('\theta_1 (radians)')
    legend('\theta_{1 cmd}','\theta_1
actual','\theta_1 simulated','',...
        ['K_P = ',num2str(Kp)],['K_D =
',num2str(Kd)],...
        ['K_I = ',num2str(Ki)],['a = ',num2str(a),'
rad/s'],...
        ['T_L = ',num2str(TL),' Nm'],4)
    axis([0 15 0 1.2])
    grid
else
end

%Now simulate when the flexishaft is connected
TL = 0;
theta_cmd_initial = 0;
theta_cmd_final = 1;
a = 100;

Kp = 12;
Kd = .2;
Ki = 3;

sim('partd_position_control_PID_gfshaft')

sim_time = simTheta1.time;
sim_theta_1 = simTheta1.signals.values;
sim_theta_2 = simTheta2.signals.values;
sim_theta_cmd = simThetaCmd.signals.values;
sim_e_1 = sime_1.signals.values;

if graphselection==1
    figure

plot(sim_time,sim_theta_cmd,sim_time,sim_theta_1,sim_t
ime,sim_theta_2)
    title('Simulated \theta_1 and \theta_2 vs. Time
for Connected System Using \theta_1 for Feedback')
    xlabel('Time (sec)')
    ylabel('\theta (radians)')
    legend('\theta_{1
cmd}','\theta_1','\theta_2','',...
        ['K_P = ',num2str(Kp)],['K_D =
',num2str(Kd)],...
        ['K_I = ',num2str(Ki)],['a = ',num2str(a),'
rad/s'],...
        ['T_L = ',num2str(TL),' Nm'],4)
    axis([0.75 2 0 1.5])
    grid

    figure

plot(sim_time,sim_theta_cmd,sim_time_D_1a,sim_theta_D_
1a,sim_time,sim_theta_1)
    title('Simulated \theta_1 vs. Time with and
without Coupling')
    xlabel('Time (sec)')
    ylabel('\theta_1 (radians)')
    legend('\theta_{cmd}','\theta_1 no
coupling','\theta_1 coupling','',...
        ['K_P = ',num2str(Kp)],['K_D =
',num2str(Kd)],...
        ['K_I = ',num2str(Ki)],['a = ',num2str(a),'
rad/s'],...
        ['T_L = ',num2str(TL),' Nm'],4)
    axis([.75 2 0 1.25])
    grid
else
end

TL = 0;
theta_cmd_initial = 0;
theta_cmd_final = 1;
a = 100;

Kp = 12;
Kd = .2;
Ki = 3;

sim('partd_position_control_PID_gfshaft2')

sim_time = simTheta1.time;
sim_theta_1 = simTheta1.signals.values;
sim_theta_2 = simTheta2.signals.values;
sim_theta_cmd = simThetaCmd.signals.values;
sim_e_1 = sime_1.signals.values;

if graphselection==1
    figure

plot(sim_time,sim_theta_cmd,sim_time,sim_theta_1,sim_t
ime,sim_theta_2)
    title('Simulated \theta_1 and \theta_2 Using
\theta_2 for Feedback')
    xlabel('Time (sec)')
    ylabel('\theta (radians)')
    legend('\theta_{cmd}','\theta_1','\theta_2','',...
        ['K_P = ',num2str(Kp)],['K_D =
',num2str(Kd)],...
        ['K_I = ',num2str(Ki)],['a = ',num2str(a),'
rad/s'],...
        ['T_L = ',num2str(TL),' Nm'])
    axis([0.9 1.5 -13 13])
    grid
else
end
```

end

Least Squares DSA Fitting

```
%Christopher Lum
%9923916
%AA448 Ly
%March 15, 2003
%
%We need to try and match up the DSA response with a
transfer function.
%This involves identifying the location of the poles
of the system.  We
%already have a model of the system, fshaft.
clear
clc
close all

%load the DSA data provided by professor Ly.
load Alflexishaft

%Extract the frequencies that the DSA used to generate
its plots
DSA_frequencies_Hz = o2i1x;
DSA_frequencies = DSA_frequencies_Hz*2*pi;      %rad/s

%Extract the data from the DSA (in the form of a
complex number)
DSA_complex_number = o2i1;

%Calculate the phase of each complex number
DSA_phase_rads = unwrap(angle(-DSA_complex_number));
%rads
DSA_phase_deg = DSA_phase_rads*(180/pi);
%degrees



%We need to account for the fact that the constant
gain fact of c.  This
%serves to decrease the magnitude by c, but not affect
the phase.  Therefore, we
%can write
c = 0.016098e3;
%rad/sec/V

%Calculate the magnitude of the complex number from
the DSA
DSA_magnitude_unmodified = abs(DSA_complex_number);
DSA_magnitude = DSA_magnitude_unmodified*c;
DSA_magnitude_dB = 20*log10(DSA_magnitude);     %dB


%Given the magnitude and the phase, calculate the real
and imaginary parts.
DSA_real = DSA_magnitude.*cos(DSA_phase_rads);
DSA_imag = DSA_magnitude.*sin(DSA_phase_rads);

%We want to make sure that the real and imaginary
parts do indeed create
%the correct bode plot.
magnitude = ((DSA_real.^2)+(DSA_imag.^2)).^0.5;
magnitude_dB = 20*log10(magnitude);

phase = atan(DSA_imag./DSA_real);
phase_deg = phase*(180/pi);


%Plot the data to make sure that we obtain the correct
DSA bode plot
figure
subplot(211);
semilogx(DSA_frequencies,DSA_magnitude_dB,DSA_frequenc
ies,magnitude_dB,'rx')
xlabel('\omega (rad/sec)')
ylabel('Magnitude (dB)');
title('Aluminium Flexishaft: G(s) =
(d\theta_2/dt)/e_1');
legend('Using ''abs'' and ''angle'' function','Using
Real and Imaginary Parts',3)
grid

subplot(212);
semilogx(DSA_frequencies,DSA_phase_deg)
xlabel('\omega (rad/sec)')
ylabel('Phase (deg)');
grid


%compare with professor Ly's code
flexishaftBode



%We see that we have successfully extracted the real
and imaginary parts of
%the complex number that creates this bode plot.  We
would like to only
%analyze the frequencies which give good results.
From inspection of the
%graph, we see that the highest frequency which still
yields good results
%is about 200 rad/s
w_cutoff = 200;     %rad/s

for counter=1:length(DSA_frequencies)
    if DSA_frequencies(counter) > w_cutoff
        break
    else
    end
end

w = DSA_frequencies(1:counter);
real_parts = DSA_real(1:counter);
imag_parts = DSA_imag(1:counter);
phase_short = DSA_phase_deg(1:counter);

DSA_complex_numbers = real_parts + imag_parts*j;

dB = 20*log10((real_parts.^2 + imag_parts.^2).^0.5);

%Plot the appended bode plot
figure
subplot(211)
semilogx(w,dB)
xlabel('\omega (rad/sec)')
ylabel('Magnitude (dB)');
title('Aluminium Flexishaft: G(s) =
(d\theta_2/dt)/e_1')
legend('Magnitude (dB)','',['\omega_{cutoff} =
',num2str(w_cutoff),' rad/s'],3)
axis([1 1000 -60 40])
grid

subplot(212)
semilogx(w,phase_short)
xlabel('\omega (rad/sec)')
ylabel('Phase (deg)')
axis([1 1000 -400 0])
grid



%Call matlab script given by Professor Ly to derive
state space models of
%the drive motor, load motor, and coupled system.
[Gdrive, Gload, Gfshaft, FS] = fshaft;
clc

%We would like to obtain the state space model of the
entire system.  This
%is the Gfshaft model
A = Gfshaft.a;
B = Gfshaft.b;
C = Gfshaft.c;
D = Gfshaft.d;

%We can obtain the transfer function between theta_2
and e_1.  This is
%between the first input and the second output.
[dirty_num,dirty_den] = ss2tf(A,B(:,1),C(2,:),D(1,1));

%clean up the numerator
tolerance = 0.0001;
[num,den] =
CleanUpTransferFunction(dirty_num,dirty_den,tolerance)
;

disp(' ')
disp('Transfer Function from Gfshaft')
G = tf(num,den)
disp(' ')

%plot the poles and zeros
figure
pzmap(G)
title('Pole Zero Map of G(s) = \theta_2/e_1 from
Gfshaft')

%we would like to find the roots of the transfer
function
all_roots = roots(den);

%We know that one of the roots is a pole at the origin
since theta_2 is
%obtained by integrating dtheta_2/dt.  We want to set
this pole to zero to
%correct for numerical roundoff error.
for counter=1:length(all_roots)
    if abs(all_roots(counter)) < 0.0001
        all_roots(counter) = 0;
        break
    else
    end
end

%Calculate the complex number for this transfer
function
gfshaft_imaginary_number =
CalculateGfshaftImaginaryNumber(w);

gfshaft_magnitude_dB =
20*log10(abs(gfshaft_imaginary_number));
gfshaft_phase_deg = unwrap(angle(-
gfshaft_imaginary_number))*180/pi;

figure
```

```
subplot(211)
semilogx(w,dB,w,gfshaft_magnitude_dB)
xlabel('\omega (rad/sec)')
ylabel('Magnitude (dB)');
title('Aluminium Flexishaft: G(s) =
(d\theta_2/dt)/e_1')
legend('DSA Data','Gfshaft',3)
axis([6 250 -60 40])
grid

subplot(212)
semilogx(w,phase_short,w,gfshaft_phase_deg)
xlabel('\omega (rad/sec)')
ylabel('Phase (deg)')
legend('DSA Data','Gfshaft',3)
axis([6 250 -400 0])
grid



%There are two imaginary poles.  We would like to vary
these two poles
%locations until we get a bode plot that matches the
DSA data.  We need to
%vary the gain K and the damping ratio zeta until we
the plot matches.
a = linspace(20,22,10);
zeta = linspace(0.06,0.1,10);
K = linspace(550000,650000,10);
wn = linspace(61,63,5);                        %rad/s


%We have an expression for the transfer function with
s = jw from
%mathematica.  We need to iterate through with
different values
counter = 1;
tic
for m=1:length(a)
    for n=1:length(zeta)
        for k=1:length(K);
            for p=1:length(wn);
                current_a = a(m);
                current_zeta = zeta(n);
                current_K = K(k);
                current_wn = wn(p);

                current_complex_numbers =
CalculateComplexNumber(...

current_a,current_zeta,current_K,current_wn,w);

                %calculate the error between this set
of complex numbers and the
                %DSA complex numbers
                error = (abs(current_complex_numbers)
- abs(DSA_complex_numbers)).^2;
                error = sum(error);

                data_array(counter,1) = counter;
                data_array(counter,2) = current_a;
                data_array(counter,3) = current_zeta;
                data_array(counter,4) = current_K;
                data_array(counter,5) = current_wn;
                data_array(counter,6) = error;

                counter = counter + 1;
            end
        end
    end
end

disp('Time to perform least squared analysis')
toc

%find the minimum error
[min_error,min_error_index] = min(data_array(:,6));
num_iterations = max(data_array(:,1));

disp(' ')
disp(['Number of iterations =
',num2str(num_iterations)])
disp(['Min error = ',num2str(min_error)])
disp(' ')

%Calculate the complex number with these critical
values of sigma and wd
min_error_a = data_array(min_error_index,2);
min_error_zeta = data_array(min_error_index,3);
min_error_K = data_array(min_error_index,4);
min_error_wn = data_array(min_error_index,5);


min_error_complex_number =
CalculateComplexNumber(min_error_a,...
    min_error_zeta,min_error_K,min_error_wn,w);

%calculate the magnitude
min_error_magnitude = abs(min_error_complex_number);
min_error_magnitude_dB =
20*log10(min_error_magnitude);

%calculate the angle
min_error_phase_rads =
unwrap(angle(min_error_complex_number));    %rads
min_error_phase_deg = min_error_phase_rads*(180/pi);
%degrees


%plot the bode plot
figure
semilogx(w,dB,w,min_error_magnitude_dB)
title('Magnitude Plot Showing DSA Acquired Data and
Least Squared Fit Data')
xlabel('Frequency (rad/s)')
ylabel('Magnitude (dB)')
legend('DSA Data','Fitted Data','',['a =
',num2str(min_error_a)],...
    ['\xi = ',num2str(min_error_zeta)],['K =
',num2str(min_error_K)],...
    ['\omega_n = ',num2str(min_error_wn)])
grid

figure
semilogx(w,phase_short,w,min_error_phase_deg)
title('Phase Plot Showing DSA Acquired Data and Least
Squared Fit Data')
xlabel('Frequency (rad/s)')
ylabel('Phase (deg)')
legend('DSA Data','Fitted Data','',['a =
',num2str(min_error_a)],...
    ['\xi = ',num2str(min_error_zeta)],['K =
',num2str(min_error_K)],...
    ['\omega_n = ',num2str(min_error_wn)])
grid


%Now the transfer function without the high frequency
poles and zeros is
%given by
num_fit = [min_error_K/(min_error_a*min_error_wn^2)];

den_fit1 = (1/(min_error_a*min_error_wn^2));
den_fit2 =
(1/(min_error_wn^2))+((2*min_error_zeta)/(min_error_a*
min_error_wn));
den_fit3 =
(1/min_error_a)+((2*min_error_zeta)/min_error_wn);
den_fit4 = 1;

den_fit = [den_fit1 den_fit2 den_fit3 den_fit4];

disp(' ')
disp('Transfer funtion of simplified G(s) (without
high frequency poles and zeros)')
G_new = tf(num_fit,den_fit)
disp(' ')

figure
pzmap(G_new)
title('Pole Zero Map of Fitted Transfer Function
Without High Frequency Poles/Zeros')


%Now add back the high frequency poles and zeros
num_total = [0.00870535 7.46208];
den_total = [1.78281e-11 3.01925e-8 0.0000133066
0.000500426 0.0522831 1];

disp(' ')
disp('Transfer function of G(s) with high frequency
poles/zeros')
G_total = tf(num_total,den_total)
disp(' ')

figure
pzmap(G_total)
title('Pole Zero Map of Fitted Transfer Funtion')

%Add pole at origin
Plant = tf([1],[1 0])*G_total;

figure
pzmap(Plant)
title('Pole Zero Map of Fitted Transfer Function vs.
Original Transfer Function')
hold on
pzmap(G)
legend('Fitted','Original')

%now we want to plot the bode plot of this function,
so we need to evaluate
%the imaginary number.
G_total_imaginary_numbers =
CalculateTotalImaginaryNumber(w);

%calculate the magnitude and the phase
total_magnitude = abs(G_total_imaginary_numbers);
total_magnitude_dB = 20*log10(total_magnitude);

total_phase =
unwrap(angle(G_total_imaginary_numbers));
total_phase_deg = total_phase*(180/pi);

%Plot the bode plot of the original DSA data, the
fitted data without high
%frequency poles/zeros, and the fitted data with the
high frequency
%poles/zeros.
figure
semilogx(w,dB,w,min_error_magnitude_dB,w,total_magnitu
de_dB)
```

```
title('Magnitude of DSA data, Fitted Data (without
high frequency poles/zeros), and Total Fitted Data')
xlabel('Frequency (rads/s)')
ylabel('Magnitude (dB)')
legend('DSA Data','Fitted without High Frequency
Poles/Zeros','Fitted Data with High Frequency
Poles/Zeros',3)
grid

figure
semilogx(w,phase_short,w,min_error_phase_deg,w,total_p
hase_deg)
title('Phase of DSA data, Fitted Data (without high
frequency poles/zeros), and Total Fitted Data')
xlabel('Frequency (rad/s)')
ylabel('Phase (deg)')
legend('DSA Data','Fitted without High Frequency
Poles/Zeros','Fitted Data with High Frequency
Poles/Zeros',3)
grid

disp(' ')
disp('Do you want to create multiple different graphs
for the previous graphs?')
disp(' ')
disp('      1 = yes')
disp('      2 = no')
disp(' ')
disp('Enter selection:  ')
graphselection = input(' ')

if graphselection==1
    figure
    subplot(211)
    semilogx(w,dB)
    title('Bode Plot of DSA data and Fitted Data')
    ylabel('Magnitude (dB)')
    axis([6 200 -25 25])
    grid

    subplot(212)
    semilogx(w,phase_short)
    xlabel('Frequency (rad/s)')
    ylabel('Phase (deg)')
    axis([6 200 -300 0])
    grid




    figure
    subplot(211)
    semilogx(w,dB,w,min_error_magnitude_dB)
    title('Bode Plot of DSA data and Fitted Data')
    ylabel('Magnitude (dB)')
    axis([6 200 -25 25])
    grid

    subplot(212)
    semilogx(w,phase_short,w,min_error_phase_deg)
    xlabel('Frequency (rad/s)')
    ylabel('Phase (deg)')
    axis([6 200 -300 0])
    grid



    figure
    subplot(211)

semilogx(w,dB,w,min_error_magnitude_dB,w,total_magnitu
de_dB)
    title('Bode Plot of DSA data and Fitted Data')
    ylabel('Magnitude (dB)')
    axis([6 200 -25 25])
    legend('DSA','3rd order fit','6th order fit',3)
    grid

    subplot(212)

semilogx(w,phase_short,w,min_error_phase_deg,w,total_p
hase_deg)
    xlabel('Frequency (rad/s)')
    ylabel('Phase (deg)')
    axis([6 200 -300 0])
    legend('DSA','3rd order fit','6th order fit',3)
    grid
else
end

%Find the location of the complex roots
disp(' ')
disp('Roots of simplified fitted transfer function
(including high frequency poles/zeros)')
fitted_roots = roots(den_total)
disp(' ')



%Finally, we can obtain a transfer function between
theta_2 and e_1 simply
%by adding a pole at the origin
den_total_integrator = [1.78281e-11 3.01925e-8
0.0000133066 0.000500426 0.0522831 1 0];

disp(' ')
disp('theta_2 to e_1 transfer function')
theta_2_to_e_1 = tf(num_total,den_total_integrator)

disp(' ')

disp(' ')
disp('Roots of theta_2 to e_1 transfer function')
theta_2_to_e_1_roots = roots(den_total_integrator)
disp(' ')

%Now compare the results with the results that
professor Ly obtained.
A_ly = [-26.84 -30.3602 -4.1184;128 0 0;0 128 0];
B_ly = [8;0;0];
C_ly = [0 0 4.0185];
D_ly = 0;

[num_ly,den_ly] = ss2tf(A_ly,B_ly,C_ly,D_ly);

[num_ly,den_ly] =
CleanUpTransferFunction(num_ly,den_ly,tolerance);

disp(' ')
disp(' Tranfer Function Derived by Professor Ly')
ly_tf = tf(num_ly,den_ly)
disp(' ')

%Find the roots
disp(' ')
disp('Roots Found by Professor Ly')
roots_ly = eigs(A_ly)
disp(' ')

figure
bode(ly_tf)
title('Bode Plot of Transfer Function Derived by
Professor Ly')
grid



%We want to incoporate the compensator used in lab
Ki = 2;
Kp = 6;

PInum=[6 2];
PIden=[1 0];

disp(' ')
disp('PI Compensator Used in Lab')
PI = tf(PInum,PIden)
disp(' ')

%combine the PI and the plant
PI_and_plant = theta_2_to_e_1*PI;

%Finally, add notch filter that we used
notchnum = [0.000196 .0056 1];
notchden = [0.000225 .0092 1];
gain = 0.875;

notch = gain*tf(notchnum,notchden);

%now formulate the complex number for the plant and
the notch filter to see
%if it helped
notch_and_plant_complex_numbers =
CalculateNotchComplexNumber(w);

notch_dB =
20*log10(abs(notch_and_plant_complex_numbers));
notch_phase =
unwrap(angle(notch_and_plant_complex_numbers))*180/pi;

figure
subplot(211)
semilogx(w,dB,w,notch_dB)
title('Bode Plot of DSA data and Filtered Signal')
ylabel('Magnitude (dB)')
axis([6 200 -25 25])
legend('DSA','Filtered')
grid

subplot(212)
semilogx(w,phase_short,w,notch_phase)
xlabel('Frequency (rad/s)')
ylabel('Phase (deg)')
axis([6 200 -300 0])
legend('DSA','Filtered')
grid
```