# UNIVERSITY OF WASHINGTON
# Department of Aeronautics and Astronautics

**Modeling and Design of a DC Motor Control System**

**February 21, 2003**

**Christopher Lum**
**Travis Reisner**
**Amanda Stephens**
**Brian Hass**

**LAB EXPERIMENT II**

**Modeling and Design of a DC Motor Control System**

**by**

**Group C-I**

**February 21, 2003**

***Christopher Lum*** **:** Evaluate the accuracy of the model with simulation. Assemble and perform experiments to determine the model parameters. Develop MatLab code for deriving parameters and simulating DSA. Built Simulink model. Answer Laboratory Discussion Items. Objective and conclusion. Assembled report.

**Contribution:** **%**

_____
Christopher Lum

***Travis Reisner*** **:** Control system design. Evaluate different experimental runs. Helped conceptualize Matlab code for simulating DSA. Lab Discussion question 5. Helped develop concepts for manual Bode plot code.

**Contribution:** **%**

_____
Travis Reisner

***Amanda Stephens*** **:** Experimental apparatus. Analyze control design performance.

**Contribution:** **%**

_____
Amanda Stephens

***Brian Hass*** **:** Experimental procedure.

**Contribution:** **%**

_____
Brian Hass

**Estimated total time spent**: 70+ hours

## I. Experiment Objectives

There are several main objectives to this laboratory. The main goal is to become familiar with the procedures and techniques used to mathematically model a DC motor. This involves identifying the important physical parameters that must be measured. The secondary goals are to develop a working model of the system that can be used for analysis and simulation. Lastly, the objective is to become familiar with the concepts of feedback control and to demonstrate its capability to control the position and velocity of the system.

## II. Experiment Apparatus

The schematic shown in Figure 1 below shows the physical apparatus used in this experiment, including the sensors, actuators, and plant (a DC motor with a circular disk attached at its end). The DC motor used is a YASKAWA ELECTRIC (MINI-Minertia Motor) Model# TO3L-QU11. The disk can be locked in position by inserting a key into the lock mechanism located toward the base of the motor stand. The sensor is an optical shaft encoder from Servo Systems, Co. and is used to measure the angle of rotation.

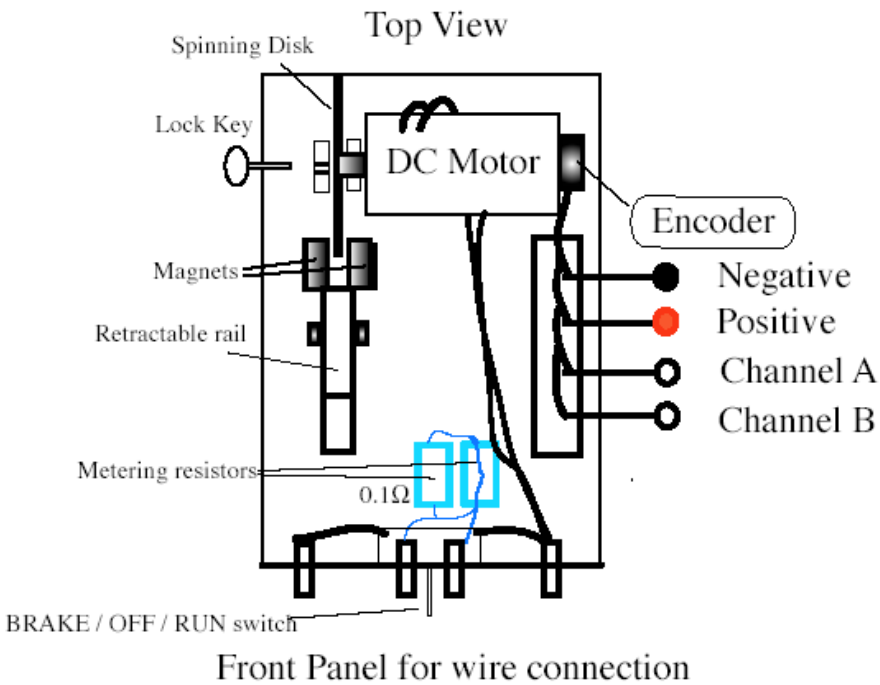


Figure 1: Experimental apparatus.

The physical apparatus shown above can be theoretically modeled through a derivation using the principle of electromagnetism and looking at the circuit model of the DC motor shown in Figure 2 below.

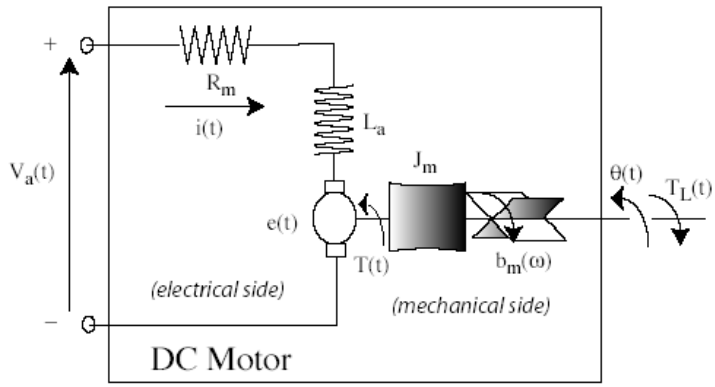

Figure 2: Circuit model of the DC motor.

By relating the equations governing the electrical and mechanical sides of the DC motor circuit model, the DC motor parameters $R_m$, $L_a$, $J_m$, $b_m$, and the current to motor torque constant Kt are to be found, yielding a physical model.

The following is a list of equipment utilized in order to run this laboratory and achieve the objectives discussed previously:

(a) DC power supply Model GPC-3030D from Good Will Instrument Co.
(b) Digital multimeter Model GDM-8145 from Good Will Instrument Co.
(c) Mixed-Signal Oscilloscope Agilent 54621D from Agilent Technologies.
(d) Bipolar Operational Power Supply Kepco Model BOP 36-12M
(d) HP 35665A Dynamic Signal Analyzer.
(e) PC computer with a four-channel quadrature encoder input board PCI-QUAD04 from Measurement Computing Corporation, and a multifunction measurement and control board PCI-DAS1602/16 from ComputerBoards, Inc.
(f) Mathworks softwares: Simulink, Realtime Workshop and xPC target.

No other electronic parts are used in the experimental procedure. A front view of the Kepco Power Supply listed in (d) is shown below in Figure 3.

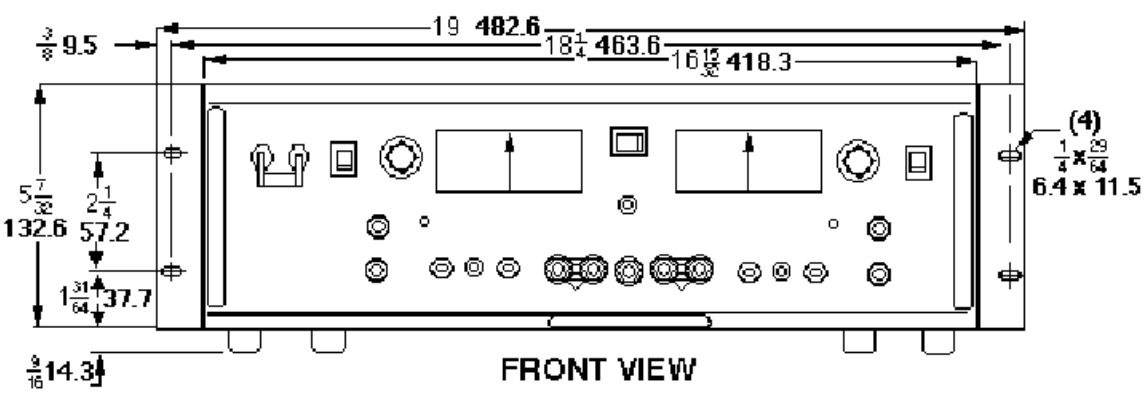

Figure 3: Front view of the Kepco Power Supply Model BOP 36-12M.

## III. List of Symbols

The symbols and definitions used in this lab are shown below in Table 1

Table 1: Nomenclature used in this report

| Symbol | Definition and Units |
|---|---|
| $V_a$ | Voltage applied across armature of motor (volts) |
| $R_m$ | Resistance of motor (ohms) |
| R | Resistance of measurement resistor (0.05 ohms) |
| $L_a$ | Inductance of motor (henries) |
| $e(\omega)$ | Back emf voltage (volts) |
| $K_v$ | Electrical machine constant (volt sec) |
| $T_L(t)$ | Torque due to external loadings (Nm) |
| $J_m$ | Moment of inertia about axis of rotation (kg $m^2$) |
| $\theta(t)$ | Angle of rotation (radians) |
| $K_T$ | Mechanical machine constant (Nm/amp) |
| $V_r(t)$ | Voltage measured across R (volts) |
| $u_o$ | Magnitude of step change in armature voltage (volts) |
| $\Delta i_{ss}$ | Change in steady state current due to step change in $V_a$ |
| $t_r$ | Time for signal to rise to critical value (sec) |
| $\omega_{break}$ | Break frequency (rad/s) |
| f | Frequency of pulses (pulses/sec or lines/sec) |
| N | Number of lines on encoder per 1 revolution |
| $V_{in}$ | Voltage supplied to Kepco amplifier (volts) |
| $K_a$ | Gain of Kepco amplifier |
| $J_m$ | Moment of inertia of wheel and flywheel (kg $m^2$) |
| $K_{vp}$ | Proportional gain used in inner velocity control loop |
| $K_{vi}$ | Integral gain used in inner velocity control loop |

## IV. System Modeling

In order to develop a mathematical model of the system, the governing equations of the system must first be derived. A schematic of the DC motor is shown below in Figure 4.

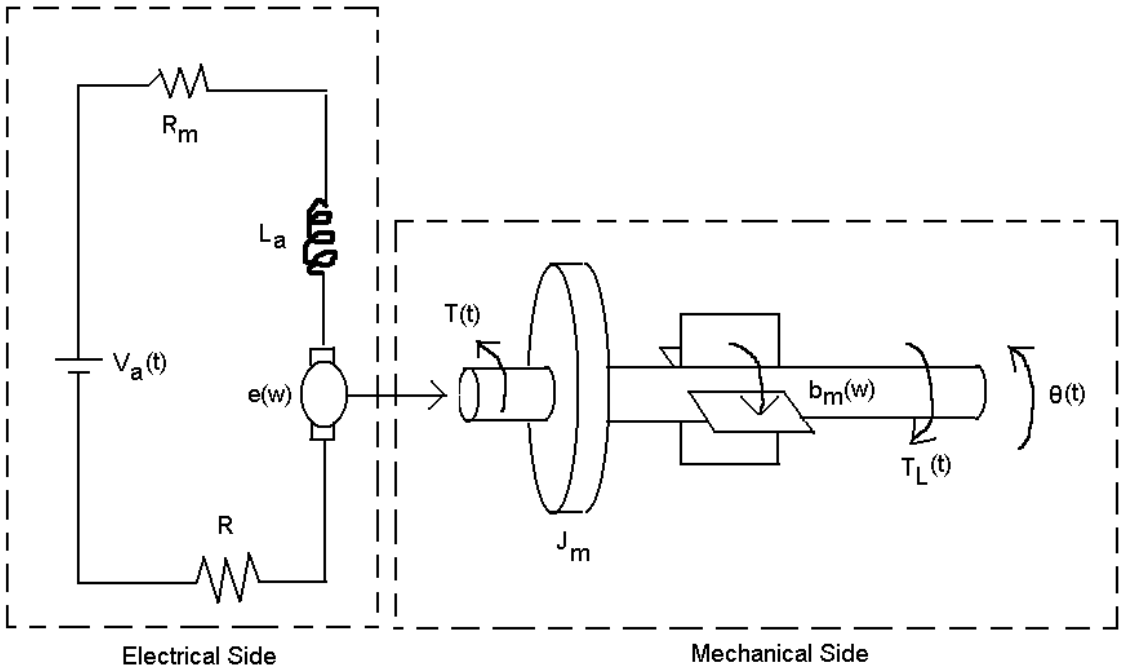


Figure 4: Schematic of DC motor and its systems

The DC motor can be electrically modeled as a resistor ($R_m$), and inductor ($L_a$), and a back emf voltage ($e(\omega)$). The back emf voltage is found experimentally to be directly proportional to the angular velocity of the motor and is given by.

$$e(\omega) = K_v \omega(t) \qquad \textbf{\textit{(Eq.1)}}$$

The idea of the DC motor is that a voltage is applied across the armature of the motor ($V_a$) which causes it to spin.. Writing the voltage law around the loop yields the electrical equation as shown below.

$$V_a(t) = (R_m + R)i(t) + L_a \frac{di(t)}{dt} + K_v \omega(t) \qquad \textbf{\textit{(Eq.2)}}$$

When the voltage is applied across the armature, the motor then produces a torque, which is directly proportional to the current and related through the motor constant as shown below.

$$T(t) = K_T i(t) \qquad \textbf{(Eq.3)}$$

This torque is used to spin the flywheel. The rotation is opposed by a friction torque which is proportional to the angular velocity ($b_m(\omega)$) and any load torques ($T_L(t)$). The mechanical equation can be written using Newton's second law for rotation and is shown below.

$$J_m \ddot{\theta} = K_T i(t) - b_m(\omega) - T_L(t) \qquad \textbf{(Eq.4)}$$

Eq.2 and Eq.4 are the governing equations for the DC motor. Using these equations, various tests can be performed in order to determine the unknown values. The first parameter to be solved for is $R_m$, the effective resistance of the motor.

**Solve for $R_m$**

In order to solve for this parameter, the rotor is locked so that it cannot rotate ($\omega = 0$). Also, it is allow to reach steady state until data is taken ($di/dt = 0$). The electrical equation (Eq.2) becomes

$$V_a(t) = (R_m + R)\, i(t) \qquad \textbf{\textit{(Eq.5)}}$$

Since R is known, the voltage across the armature and R can be measured. Since the current through the loop must be constant, the current is directly related to the voltage across R and the resistant R. The measured values are shown below in Table 2.

Table 2: Armature voltage, measurement voltage, rotor position, and calculated current for locked rotor.[1]

| Va (volts) | Vr (mV) | i(t) (amps) | rotor position |
|---|---|---|---|
| 3.689 | 122.68 | 2.4536 | 1 |
| 3.698 | 122.5 | 2.45 | 2 |
| 3.713 | 116 | 2.32 | 3 |
| 3.727 | 113 | 2.26 | 4 |
| -3.71 | -118 | -2.36 | 4 |
| -3.71 | -120 | -2.4 | 1 |
| -3.71 | -120.8 | -2.416 | 2 |
| -3.71 | -118.9 | -2.378 | 3 |
| -1.879 | -62.36 | -1.2472 | 3 |
| -1.901 | -54.5 | -1.09 | 4 |
| -1.895 | -56.9 | -1.138 | 1 |
| -1.9 | -55 | -1.1 | 2 |
| 1.901 | 54 | 1.08 | 2 |
| 1.891 | 56.23 | 1.1246 | 3 |
| 1.91 | 44 | 0.88 | 4 |
| 1.879 | 60 | 1.2 | 1 |

As can be seen, the values vary slightly for each rotor position. Therefore, for each armature voltage, the values are averaged to produce one test point. The armature voltage vs. the current can be plotted. The slope should be $R_m + R$.

[1] Varying significant figures reflects confidence in measurement. The values fluctuated erratically during data recording.

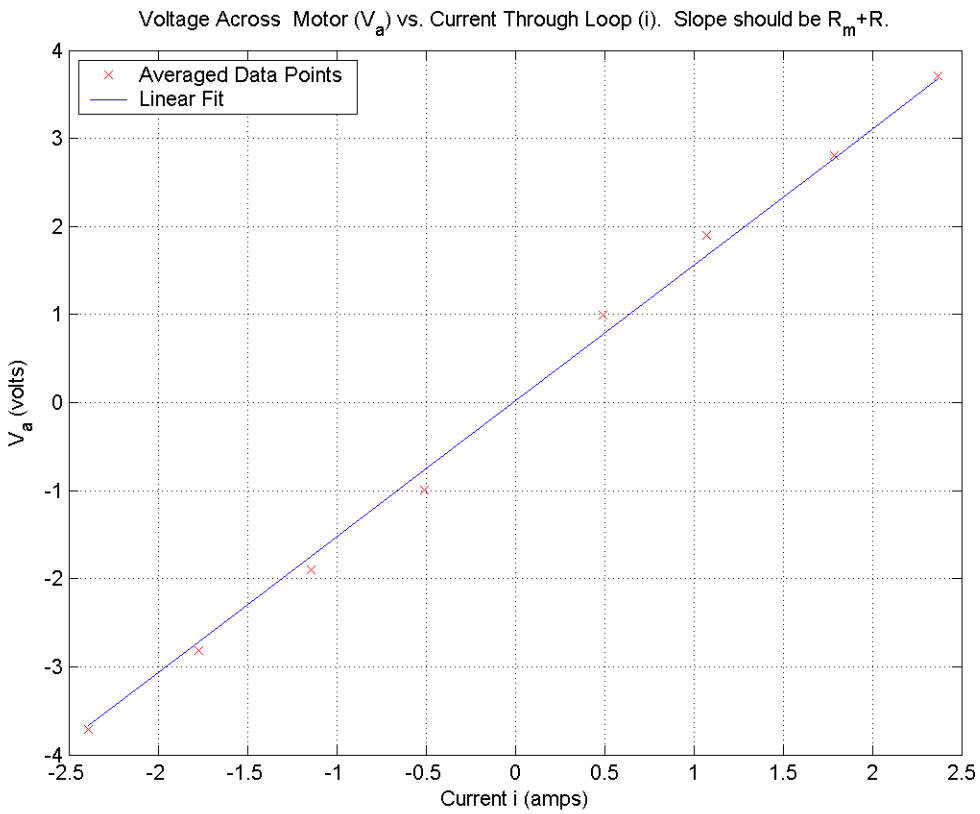


Figure 5: Armature voltage vs current

As can be seen, the data seems to fit a linear line fairly well. From this, the effective resistance is determined by subtracting R from the slope of the line. This yields

**$R_m$ = 1.543 ohms**

**Solve for $L_a$**

The inductance of the motor ($L_a$) can now be determined in two methods. The first method involves capturing the transient response of the motor current with a locked rotor. In this situation, the electrical equation (Eq.2) becomes

$$V_a(t) = (R_m + R)i(t) + L_a \frac{di(t)}{dt}$$

The transfer function can be solved for by taking the Laplace transform of this equation this yields

$$V_a(s) = (R_m + R)I(s) + sL_a I(s)$$

The transfer function of the system between the current and the armature voltage can now be solved for, this yields

$$G(s) = \frac{I(s)}{V_a(s)} = \frac{1}{sL_a + (R_m + R)} \qquad \textbf{\textit{(Eq.6)}}$$

As can be seen, this is a first order transfer function. The response to a step of magnitude $u_o$ is given by

$$I(s) = \left[\frac{1}{sL_a + (R_m + R)_a}\right]\frac{u_o}{s}$$

The time response of the current can be found by simply taking the inverse Laplace transform. This yields.

$$i(t) = \frac{u_o}{R_m + R}\left(1 - e^{-\frac{(R_m+R)}{L_a}t}\right) \qquad \textbf{\textit{(Eq.7)}}$$

We know that as t goes to infinity, the current must change by a magnitude of $\Delta i_{ss}$. Therefore, the coefficient $u_o/(R_m+R)$ must equal $\Delta i_{ss}$. Eq.7 can now be rewritten as

$$i(t) = \Delta i_{ss}\left(1 - e^{-\frac{(R_m+R)}{L_a}t}\right) \qquad \textbf{\textit{(Eq.8)}}$$

Eq.8 describes the current as the temperature begins to rise. At t = $t_r$ = $L_a/(R_m+R)$ Eq.8 can be written as

$$i(t) = \Delta i_{ss}\left(1-e^{-1}\right) \approx 0.632\Delta i_{ss} \qquad \textbf{\textit{(Eq.9)}}$$

Since $\Delta i_{ss}$ can be found by looking at the transient response, one simply needs to find out at what time $0.632\Delta i_{ss}$ occurs. This time corresponds to At t = $L_a/(R_m+R)$, which can then be solved for $L_a$. The transient response of current vs. time is shown below in Figure 6.

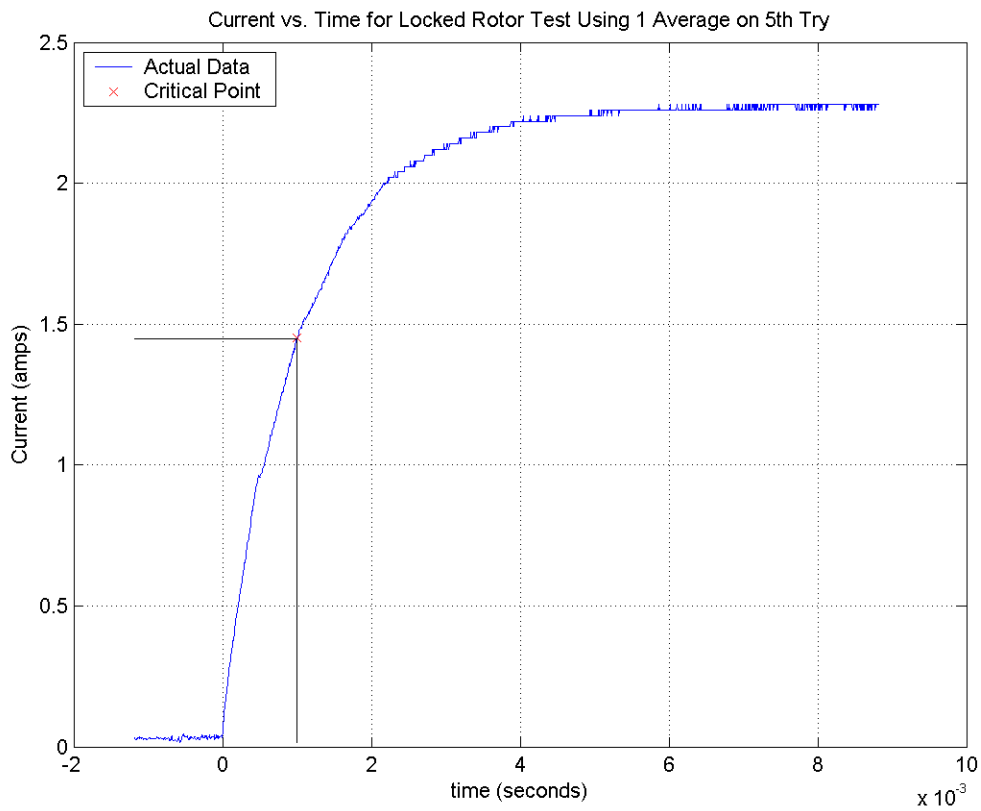


Figure 6: Transient response of current vs. time to a step input

As can be seen, the response is very fast. However, using MatLab, the critical values can be determined. The time to rise to 63.2% of the final value is

$t_r$ = 0.001 seconds

Since $R_m$ and R are already known, Eq.8 can now be solved for $L_a$, this yields

**$L_a$ = 0.00159 henries**

This value can be verified by plotting Eq.8 vs. the actual data using this value of $L_a$. This is shown below in Figure 7.

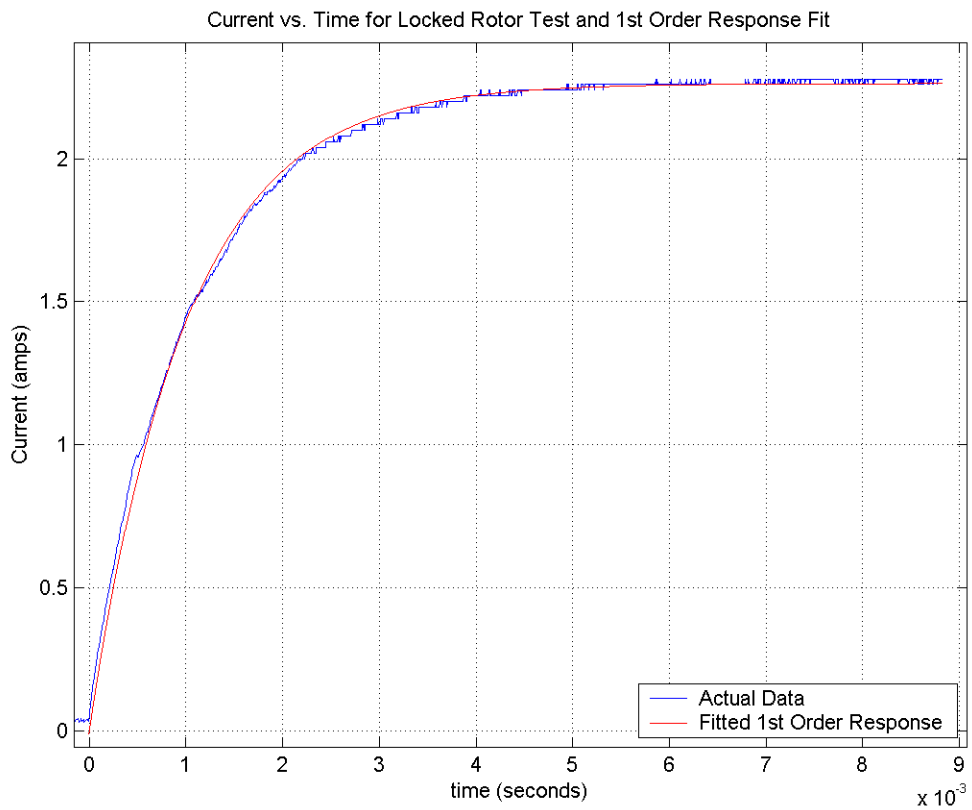


Figure 7: Eq.8 and actual data

As can be seen, this is fairly accurate which implies that the value of $L_a$ is accurate as well.

The value of La can also be determined using the Digital Signal Analyzer (DSA). The theoretical transfer function for this system (Eq.6) can be rewritten in Bode diagram normalized form as

$$G(j\omega) = \frac{1/(R_m + R)}{\left(j\omega \frac{L_a}{R_m + R} + 1\right)} \qquad \textbf{\textit{(Eq.10)}}$$

This is shows that the bode diagram for this transfer function is made up of two factors

1. Constant gain of magnitude $1/(R_m+R)$
2. Pole on the real axis with break frequency $\omega_{break} = (R_m+R)/L_a$

We can use MatLab to graph the theoretical bode plot of the system using the bode function on Eq.10 and the value of La derived using the transient response analysis. This yields Figure 8.

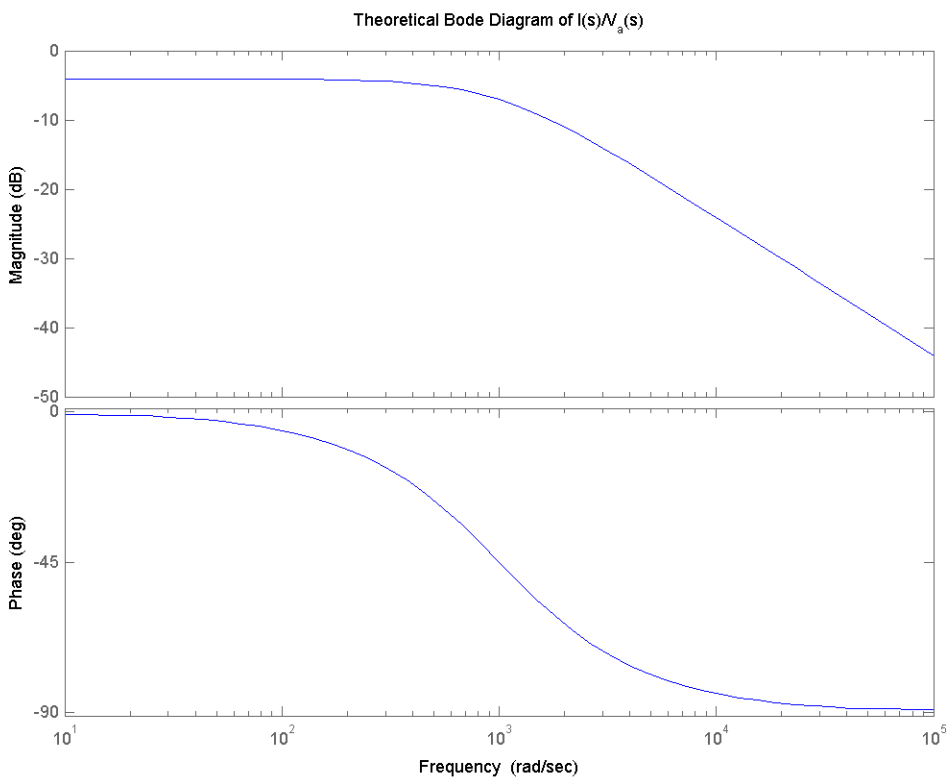


Figure 8: Theoretical bode diagram of Eq.10

The output of the DSA can be analyzed to find the constant gain term and the break frequency. However, there need to be some small changes that affect the magnitude of the bode plot. Recall that in lab, the DSA was hooked up to analyze the output voltage of the measurement resistor ($V_r$) with the input at the voltage going *into* the Kepco amplifier ($V_{in}$). Therefore the bode diagram that the DSA is computing is actually

$$H(s) = \frac{V_r(s)}{V_{in}(s)} \qquad \textbf{(Eq.11)}$$

The DSA output is to be compared to Figure xx, which is the transfer function G(s). H(s) is related to G(s) through two simple relationships. Dividing Eq.11 by (R $K_a$) yields

$$\frac{H(s)}{(K_a R)} = \frac{V_r(s)}{V_{in}(s)(K_a R)} = \left(\frac{V_r(s)}{R}\right)\left(\frac{1}{K_a V_{in}(s)}\right)$$

Recall that $V_r/R = i$ and $K_a V_{in} = V_a$. Using these two substitutions yields

$$\frac{H(s)}{(K_a R)} = \frac{I(s)}{V_a(s)} = G(s) \qquad \textbf{\textit{(Eq.12)}}$$

This shows that the transfer function G(s) can be obtained from the output of the DSA by simply accounting for the factor of $1/(K_a R)$. The value of Ka used in these calculations is described later in the section titled <u>Solve for $J_m$</u>. Keep in mind that this only serves to shift the output of the DSA up or down, it does not affect the shape of the bode diagram since it is only a constant factor. Plotting the modified DSA data for the four different wheel positions with a sweeping sine wave as excitation yields Figure 9.

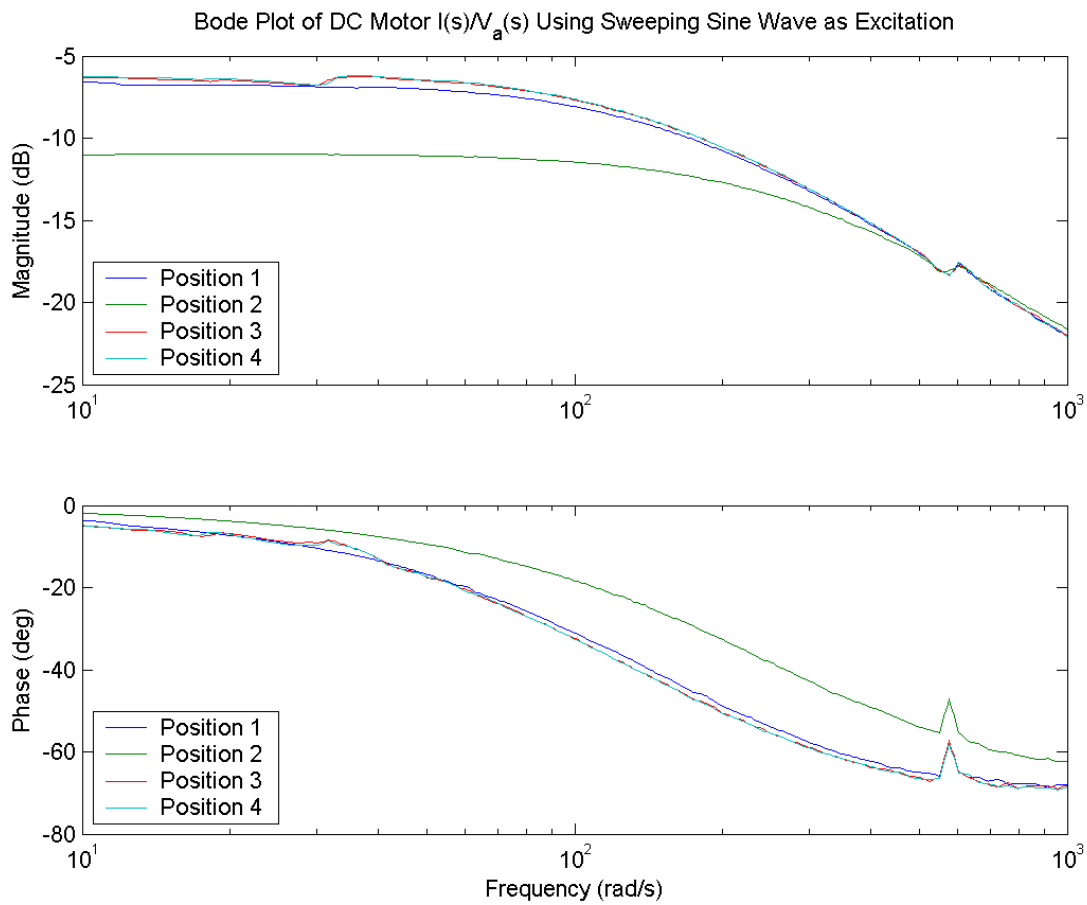


Figure 9: Bode plot of modified DSA output for $I(s)/V_a(s)$ with sweeping sin wave as excitation

As can be seen by comparing Figure xx and Figure xx, the shape and magnitudes appear to be similar. The results agree with each other fairly well with the sole exception of position 2, which appear to be an outlier.

This can be repeated using random noise as the excitation source. When generating the bode plot using the sweeping sine method, a sine wave with a certain frequency is used as input and the output is measured. The frequency is then increased and the process is repeated. This is a slow and methodical process. A faster method is to use random noise as an excitation signal. This noise signal contains many frequencies within it. Fast Fourier Transforms can then be used to analyze the output (which will contain information for many frequencies) and obtain the frequency response. Once again, the magnitude output of the DSA must be corrected by a factor of $(1/K_aR)$. This yields Figure 10.

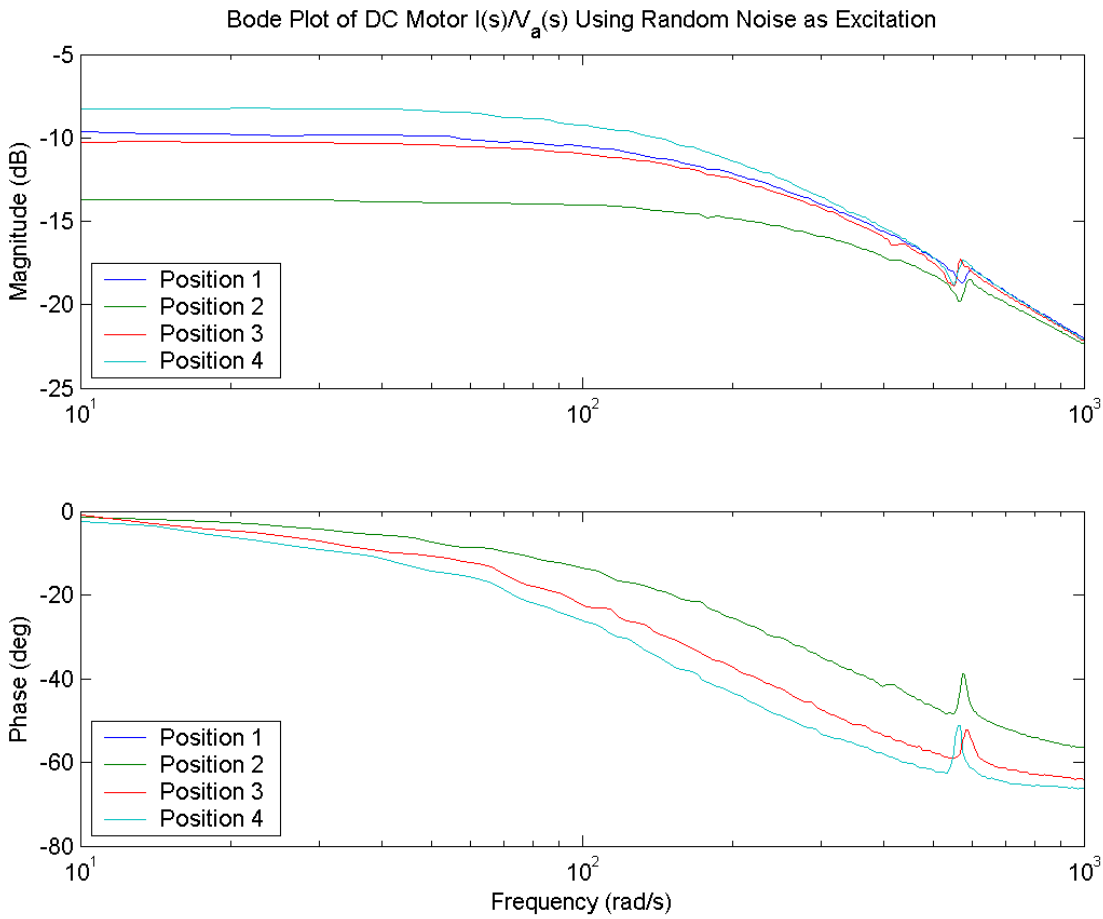


Figure 10: Bode plot of modified DSA output for $I(s)/V_a(s)$ with random noise as excitation

As can be seen by comparing Figure 9 with Figure 10, both the random noise and sweeping sine method produce similar bode plots. Both of these bode plots can be analyzed in order to find the break frequency. The break frequency for this system is the frequency where the magnitude declines 3 dB from the DC gain. By analyzing the plots, the break frequency is determined to be

**$\omega_{break}$ = 154.882 rad/s (from sweeping sin excitation)**
**$\omega_{break}$ = 242.000 rad/s (from random noise excitation)**

As stated above (after deriving Eq.10), these break frequencies should be given by $\omega_{break} = (R_m+R)/L_a$. Solving for $L_a$ yields.

**$L_a$ = 0.01028 henries (from sweeping sin excitation)**
**$L_a$ = 0.00658 henries (from random noise excitation)**

**$L_a$ = 0.00159 henries (from transient analysis)**

As can be seen, the different method produce several different results. They are within an order of magnitude of each other but still very different. Since the bode plots require a somewhat arbitrary choosing of when the break frequency occurs, they are probably less accurate. Also, by observing Figure 7, it seems that the transient analysis can be made to match the experimental results quite well. For this reason, the value of $L_a$ derived using transient analysis will be used for the remainder of the lab.

**Solve for $K_v = K_T$**

In order to solve for the machine constant, the wheel is unlocked an allowed to rotate. With the wheel spinning and at steady state, the electrical equation can be written as

$$V_a(t) - (R_m + R)i(t) = K_v\omega(t) \qquad \textbf{(Eq.13)}$$

As can be seen, at steady state, the current and ω should be a constant. Therefore Eq.13 is the equation of a line with slope $K_v$ and y values of $V_a - (R_m+R)i$. In order to solve for $K_v$, the angular velocity must be known. Therefore the output of the encoder is recorded. The encoder and its operation is described in the Experimental Apparatus section. The output of the encoder is shown below for a counter clockwise rotation.

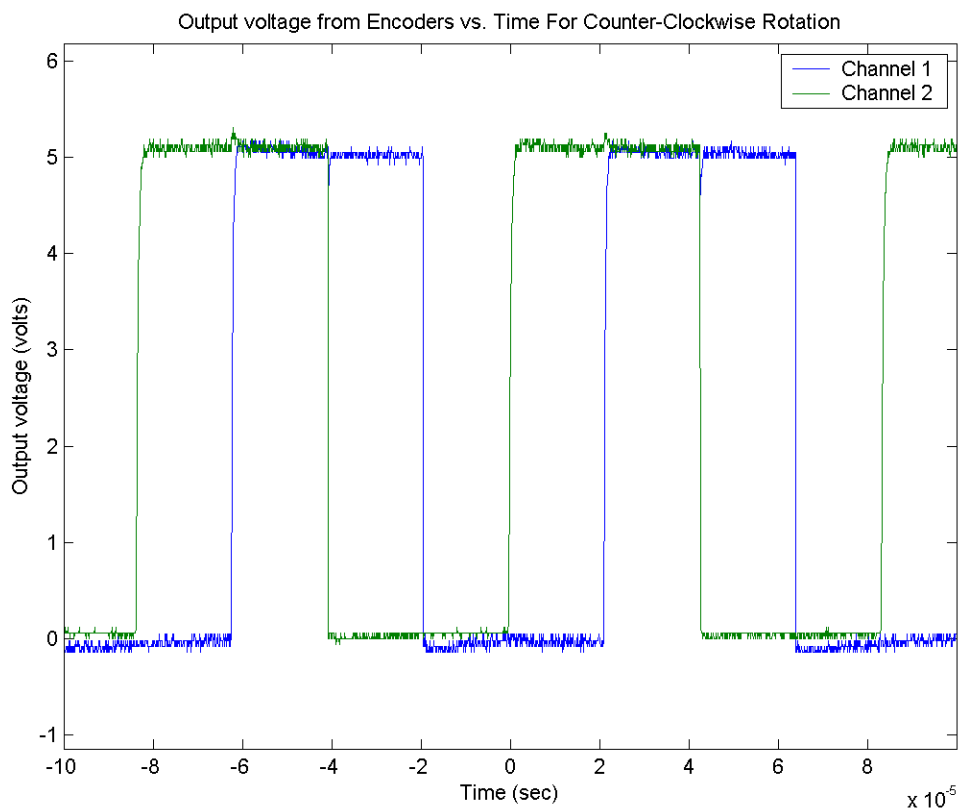


Figure 11: Output voltage from encoder for counter clockwise rotation

As can be seen, there are two pulses read by the encoder. The direction of the encoder can be determined by looking to see what pulse leads which. For example, an opposite rotation is shown below in Figure 12.

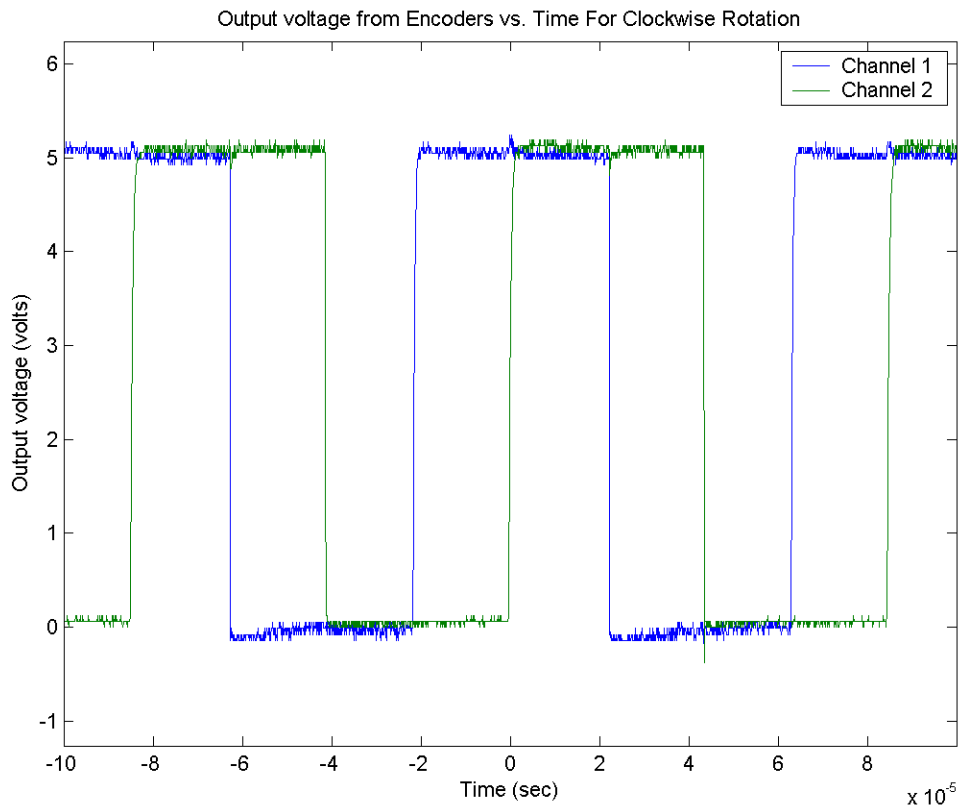


Figure 12: Output voltage from encoder for clockwise rotation

Notice that in this situation, channel 1 leads channel 2 whereas it was opposite for counter clockwise rotation.

Also, the frequency of pulses (f) can be used to determine what that angular velocity is if the number of lines per rotation (N) are known. The workstation used to analyze the data used a 1000 line/rev encoder. The governing equation is

$$\omega(t) = 2\pi \frac{f}{N} \qquad \textbf{\textit{(Eq.14)}}$$

The angular velocity can be varied by changing the armature voltage and the eddy current dampener. The results are shown below in Table 3

Table 3: Armature voltage, voltage across measurement resistor, current, frequency, angular velocity, and brake position used to determine $K_v$

| $V_a$ (volts) | $V_r$ (mV) | i (amps) | f (kHz) | ω (rad/s) | Brake Position (%) |
|---|---|---|---|---|---|
| -7.611 | -128.23 | -2.5646 | -6.02 | -37.82 | 100 |
| -7.706 | -97.21 | -1.9442 | -6.02 | -37.82 | 75 |
| -7.835 | -54 | -1.08 | -9.76 | -61.32 | 50 |
| -7.946 | -17.46 | -0.3492 | -11.75 | -73.83 | 25 |
| -7.955 | -14.15 | -0.283 | -12 | -75.40 | 0 |
| 7.591 | 136.75 | 2.735 | 6.3 | 39.58 | 100 |
| 7.692 | 100.4 | 2.008 | 7.95 | 49.95 | 75 |
| 7.819 | 58.6 | 1.172 | 10 | 62.83 | 50 |
| 7.937 | 19.17 | 0.3834 | 11.9 | 74.77 | 25 |
| 7.945 | 15.95 | 0.319 | 12.121 | 76.16 | 0 |
| 3.837 | 70 | 1.4 | 2.9 | 18.22 | 100 |
| 3.87 | 54 | 1.08 | 3.67 | 23.06 | 75 |
| 3.936 | 32.2 | 0.644 | 4.63 | 29.09 | 50 |
| 3.99 | 14.5 | 0.29 | 5.56 | 34.93 | 25 |
| 3.995 | 12.98 | 0.2596 | 5.68 | 35.69 | 0 |

The quantity $V_a - (R_m+R)i$ can be plotted on the y axis and the corresponding value of ω can be plotted in the x-axis. A linear best fit line can then be fit to the data, the slope of it should be $K_v$. This is shown below in Figure 13.

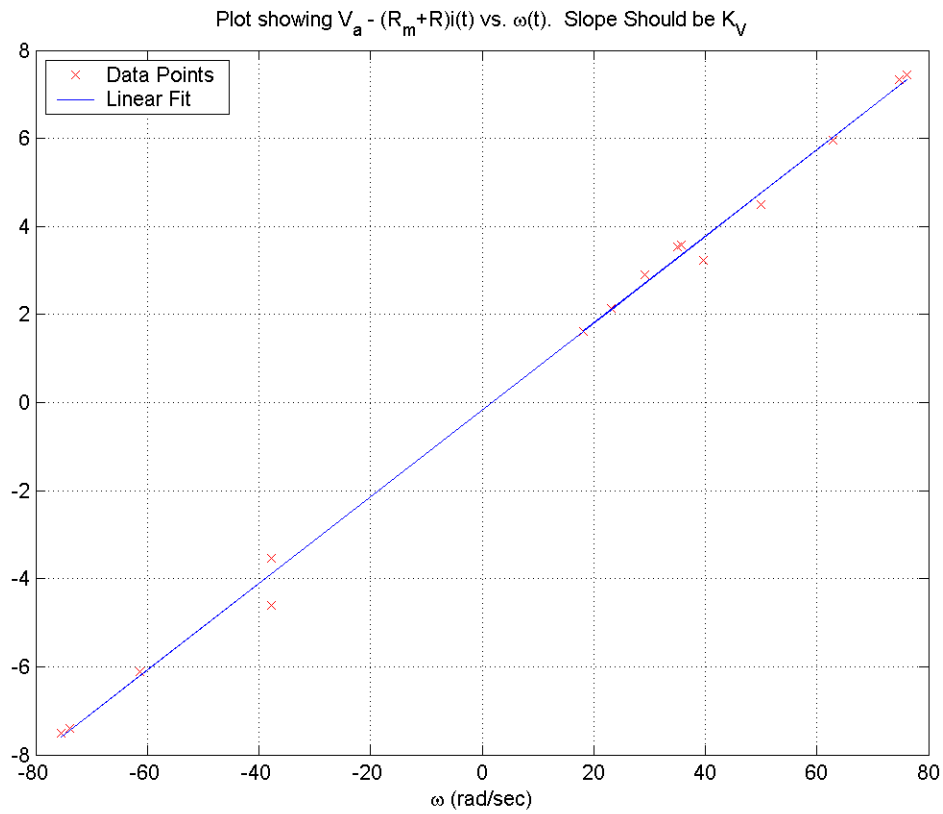


Figure 13: Plot used to determine $K_v$

As can be seen, the data fits the linear line fairly well. However, notice that there is no data for small ω. This makes sense considering the dead zone where the motor will not spin at low values of ω. The slope of the 1st order fit is found to be

**$K_v$ = 0.09854 volt sec**

The value of $K_T$ can also found by balancing of power. The power consumed by the motor must equal the power output from the motor.

$$P(t) = e(t)i(t) = T(t)\omega(t)$$

Recall that the back emf voltage and the torque are given by Eq.1 and Eq.3, respectively.

$$K_v\omega(t)i(t) = K_T i(t)\omega(t)$$

Canceling the current and angular velocity yields

$$K_v = K_T \qquad \textbf{\textit{(Eq.15)}}$$

Eq.15 shows that $K_v$ and $K_T$ are actually the same number if a consistent set of units is used. The value is shown below.

**$K_T$ = 0.09854 Nm/amp**

**Solve for $b_m$**

The torque due to friction can now be determined. Recall that this is a nonlinear function of ω and needs to be experimentally determined. The motor is run at various $V_a$ values with the eddy current brake out and allowed to reach steady state. Here, the friction force must balance the torque produced by the motor in order to ensure that the angular acceleration is zero. In this situation, the mechanical equation (Eq.4) can be written as

$$0 = K_T i(t) - b_m(\omega)$$

$$K_T i(t) = b_m(\omega) \qquad \textbf{\textit{(Eq.16)}}$$

Eq.16 shows that the friction force is a function of angular velocity and is the product of $K_T$ and the current. By varying the current and measuring the angular velocity, a profile of the friction force as a function of ω can be determined. The data collected in lab is shown below in Table 4

Table 4: $V_a$, $V_r$, frequency, angular velocity, current, and torque

| $V_a$ (volts) | $V_r$ (mV) | f (kHz) | ω (rad/s) | i (amps) | $b_m$ (Nm) |
|---|---|---|---|---|---|
| -9.995 | -14.41 | -15.24 | -0.0958 | -0.2882 | -0.0284 |
| -8.018 | -13.5 | -12.05 | -0.0757 | -0.27 | -0.0266 |
| -6 | -13 | -8.85 | -0.0556 | -0.26 | -0.0256 |
| -4.03 | -12.2 | -5.78 | -0.0363 | -0.244 | -0.0240 |
| -2.012 | -12 | -2.50 | -0.0157 | -0.24 | -0.0236 |
| -0.67 | -11 | 0.00 | 0.0000 | -0.22 | -0.0217 |
| -0.489 | -13.2 | 0.00 | 0.0000 | -0.264 | -0.0260 |
| 0.482 | 14.36 | 0.00 | 0.0000 | 0.2872 | 0.0283 |
| 1.02 | 13.5 | 0.00 | 0.0000 | 0.27 | 0.0266 |
| 2.001 | 11.9 | 2.30 | 0.0145 | 0.238 | 0.0235 |
| 4.02 | 12.8 | 5.62 | 0.0353 | 0.256 | 0.0252 |
| 6.015 | 14 | 9.01 | 0.0566 | 0.28 | 0.0276 |
| 8.019 | 15.5 | 12.20 | 0.0767 | 0.31 | 0.0305 |
| 10 | 16.6 | 15.50 | 0.0974 | 0.332 | 0.0327 |

These values can be plotted with their corresponding angular velocities. It is known that the friction force is non linear and has large values at low velocities due to the sticktion forces. However, at $b_m(0) = 0$. This is modeled as Coulomb friction instead of simple viscous friction. However, in the Simulink model, the coulomb friction block does not appear to work well. Instead, a look-up table can be employed. This can be modeled as linear line with a steep slope which passes through (0,0) and then decreases after the friction overcomes the initial sticktion. This is shown below in Figure 14.

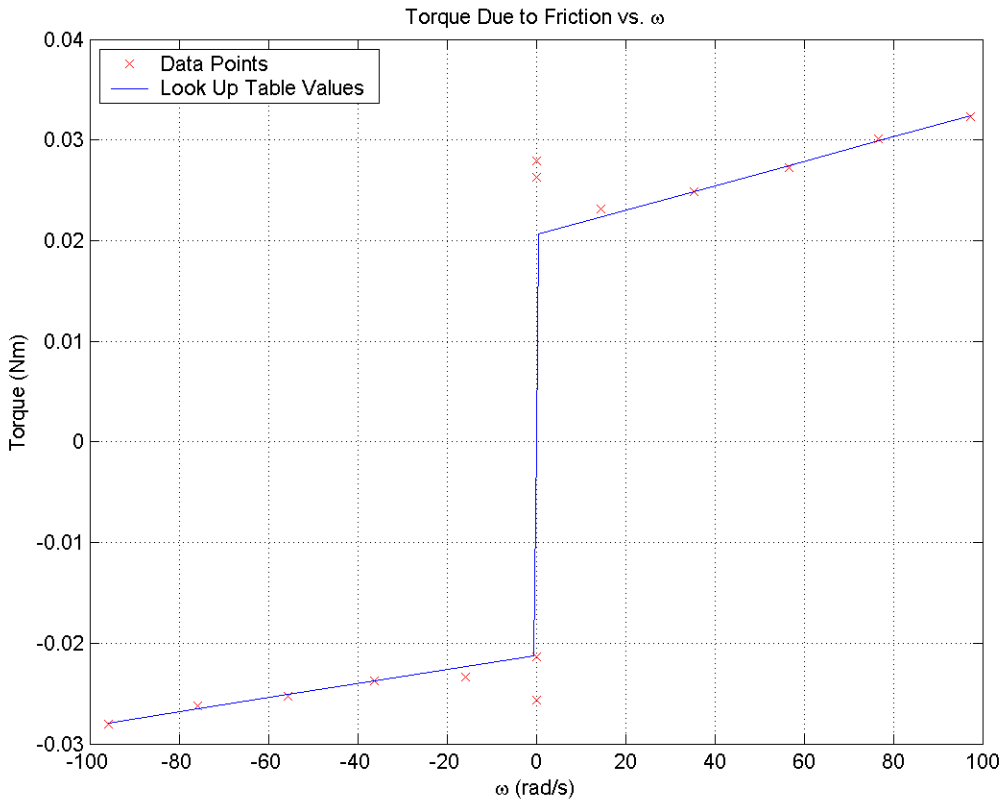


Figure 14: Friction torque vs. ω including linear fit lines

As can be seen, the friction force is indeed non linear. Using the values on the Look Up Table line can be used in the Simulink model to model the behavior of the torque due to friction.

**Solve for $J_m$**

The last parameter that must be determined is the moment of inertia about the axis of rotation ($J_m$). In order to derive this parameter, a pulse input in voltage is sent to the motor. This is achieved using the xpcTarget program as described in the experimental apparatus section. This sends a pulse signal to the amplifier which then sends an amplified signal to the motor. The gain of the amplifier ($K_a$) must be deteremined. The voltage before ($V_{in}$) and after the amplifier is measured ($V_a$)

$V_{in}$ = 0.996 volts
$V_a$ = -3.560 volts

The interesting thing to notice is that the amplifier actually flips the sign of the signal. The gain is simply given by.

$$K_a = \frac{V_a}{V_{in}} \qquad \textbf{\textit{(Eq.17)}}$$

Using Eq.17, the gain of the amplifier is found to be

**$K_a$ = -3.574**

Note that the gain is negative. This is because the amplifier flips the polarity of the signal. However, as long as the sign convention is taken into account, everything will work out fine.

The output of the encoder is then saved in response to this pulse voltage. This is shown below in Figure 15

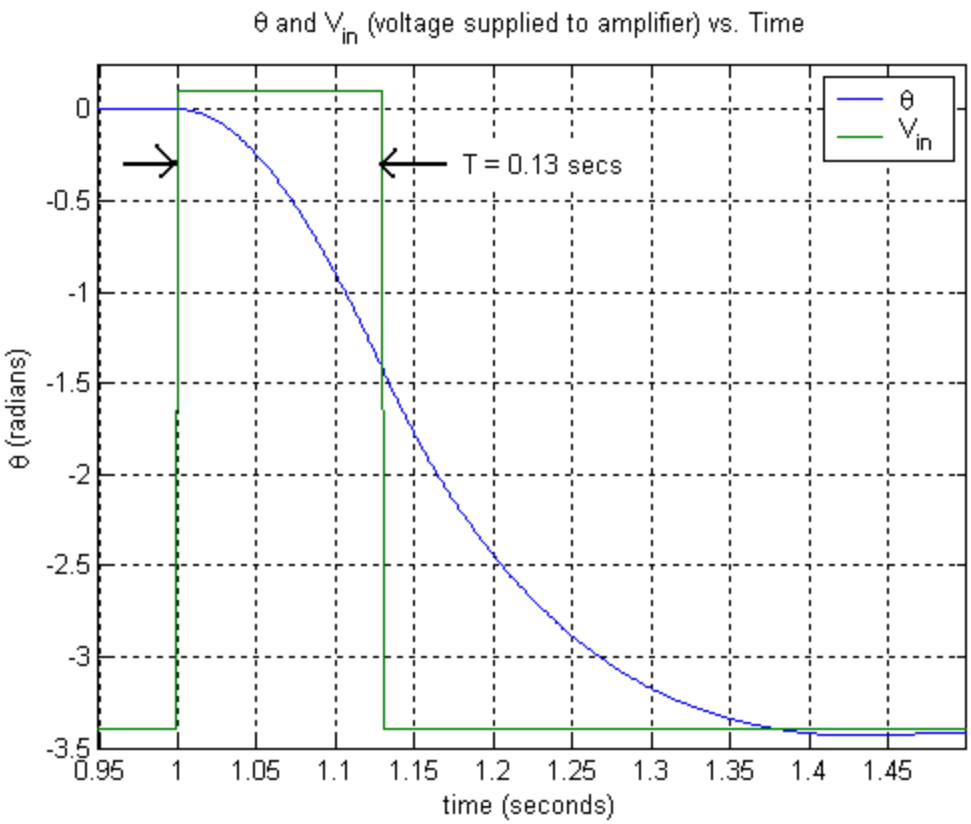


Figure 15: θ vs. time and $V_{in}$ vs. time

As can be seen, when the pulse is applied to the amplifier, the wheel begins to accelerate. Even after the voltage drops back down to zero, the wheel continues to rotate due to the momentum that it has accumulated. Another interesting thing to realize is that even though a positive value of $V_{in}$ was supplied, the wheel rotated in the negative direction. This is due to the fact that the amplifier actually flips the sign of the signal. This is consistent with a negative angle of rotation. According to the sign convention, a positive rotation is clockwise. In lab, the wheel rotated counter clock wise even though $V_{in}$ was positive. Since $K_a$ is negative, the actual voltage across the armature was negative, which results in a counter clockwise rotation.

For the majority of the pulse (most the second half), the velocity appears to be roughly constant. If this is the case, then the torque due to friction should be a constant. Also, since there is no load, $T_L(t) = 0$. This shows that the net torque on the wheel should be the torque produced by the motor minus the constant friction torque. Therefore the net torque on the motor should be a constant. From elementary statics, a constant torque should produce a constant angular acceleration. Therefore, the line θ vs. t should be able to be fitted with a second order equation. Using MatLab's polyfit function yields Figure 16.

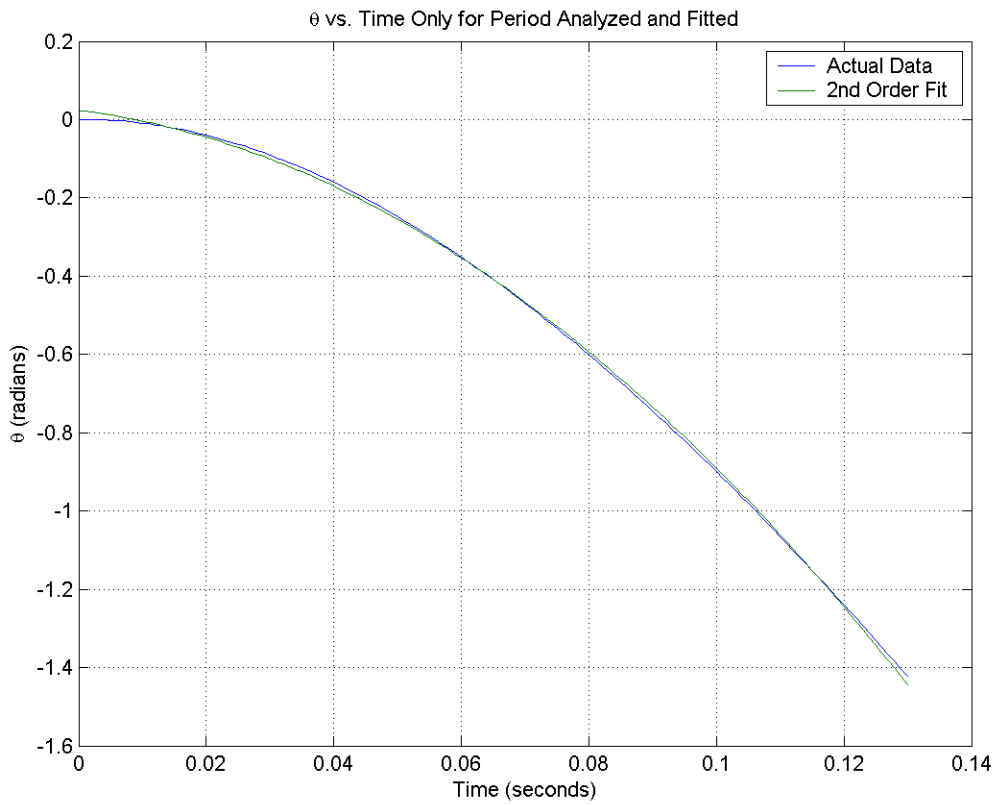


Figure 16: θ vs. time and 2nd order curve fit while pulse is present

As can be seen, the second order curve fit fits the actual data very well. This means that the approximation that the torque is constant is a fairly accurate assumption. The equation of the line is determined to be

$$\theta(t) = -71.55598t^2 - 1.97924t + 0.02302 \qquad \textbf{\textit{(Eq.18)}}$$

The angular acceleration can now be deteremined by taking the second derivative of Eq.18. This yields

$$\ddot{\theta} = -143.1120 \text{ rad/s}^2$$

An expression for $J_m$ can be derived by looking at the mechanical equation (Eq.4). Recall that the wheel has not had a chance to increase velocity much and the angular velocity is assumed to be a constant low value ($b_m(0)$). Also, the eddy current brake is backed out all the way. Therefore the mechanical equation can be written as.

$$J_m \ddot{\theta} = K_T i(t) - b_m(0)$$

$$J_m = \frac{K_T i(t) - b_m(0)}{\ddot{\theta}} \qquad \textbf{(Eq.19)}$$

An expression for i(t) can be derived by looking at the electrical equation. In this situation, Eq.2 can be written as

$$V_a(t) = (R_m + R)i(t) + L_a \frac{di(t)}{dt} \qquad \textbf{(Eq.20)}$$

As can be seen, Eq.20 cannot be solved for a constant value of i(t) since there is a derivative term. The affect of the changing current must be evaluated. In order to do this, the oscilloscope was attached to measure $V_r$ as the pulse was applied. With the trigger set to falling edge, the transient response of the current can be evaluated when the pulse is applied by simply scaling the output $V_r$ vs. t by a factor of R. This yields Figure 17.

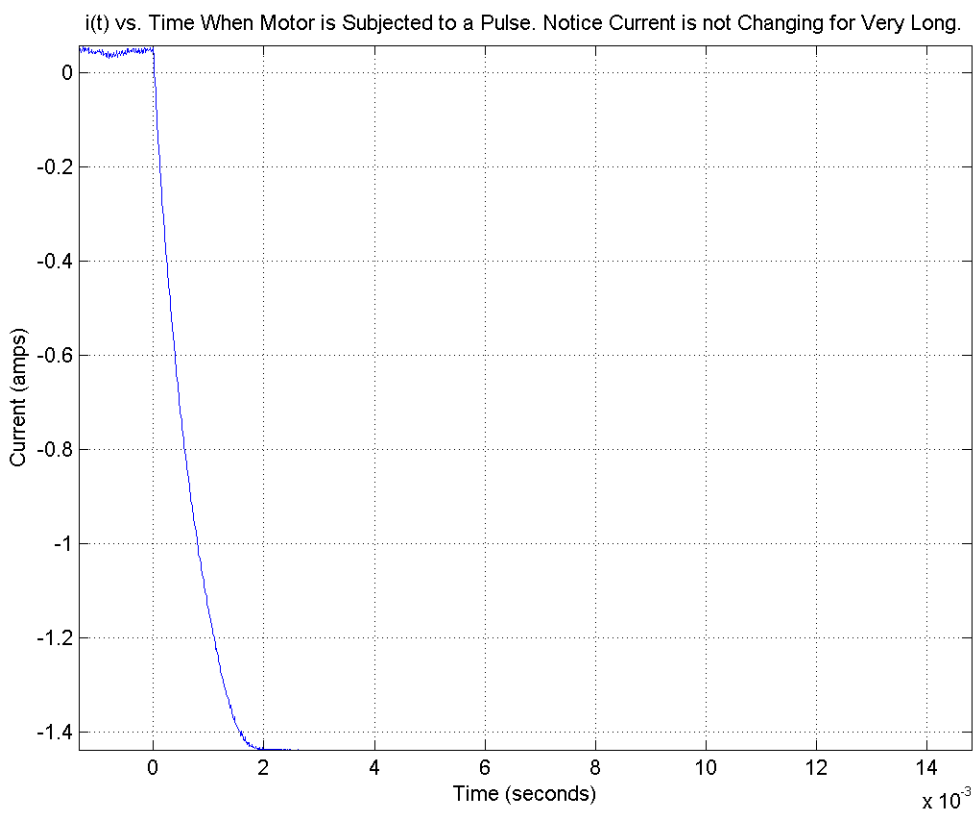


Figure 17: i(t) vs. t when square pulse is applied

As can be seen, the current is most definitely changing and the term di(t)/dt is not zero initally. The thing to notice is the time scale. The term di(t)/dt is only non zero for roughly 0.00199 seconds. However, the pulse was applied for 0.13 seconds. This means that the term di(t)/dt is only non zero for 1.53% of the time analyzed time. Therefore, it is a reasonable approximation to approximate di(t)/dt = 0 in Eq.20. Eq.20 now becomes

$$i(t) = \frac{V_a(t)}{(R_m + R)}$$

Recall that $V_a = K_a V_{in}$

$$i(t) = \frac{K_a V_{in}(t)}{(R_m + R)} \qquad \textbf{\textit{(Eq.21)}}$$

Solving Eq.21 for i(t) yields

**i(t) = -2.2352 amps**

The term $b_m(0)$ is the friction torque at roughly zero velocity. This can be found by referring to Figure 14. Since the wheel is rotating in the negative direction, the value on the negative side is used. This opposes the torque produced so the absolute value must be used.

**$b_m(0)$ = -0.0212 Nm**

Eq.19 now becomes

$$J_m = \frac{(0.0973\text{Nm/amp})(-2.2352 amps) - (-0.0212 Nm)}{-143.112 rad/s^2}$$

**$J_m$ = 0.00137 kg $m^2$**

## V. Control System Design

The controller was developed in two pieces. The first piece was to develop a velocity controller. This controller would be required to track a rotational velocity command input. This is shown later in the Model Validation section.

### Velocity Control System

Development of the velocity controller commenced with a control system as shown in Figure 18.. The *Velocity/Position Control Switch* was placed in the up position to enable the velocity command tracking. Initially, the *Velocity Command Prefilter* block was omitted.

The first portion of the development of the velocity controller required that the integral controller be disabled. To do this, the value of $K_{vi}$ was set to zero. By providing an initial velocity of 2 rad/s for 1 second and then commanding a step change of +2 rev/s (+6.28 rad/s), the effect of the *Pseudo-Derivative* roll off frequency and the proportional gain, $K_{vp}$ was evaluated.

First the effect of the roll off frequency of the pseudo-derivative was evaluated under a constant proportional gain. Table 5 lists the specific system parameters used for each run.

Table 5: Pseudo-derivative roll off frequency variation runs

| Run | $\omega_{break}$ pseudo-derivative | $K_{vp}$ | $K_{vi}$ | Velocity Prefilter Roll Off Freq. |
|---|---|---|---|---|
| 01 | 60 | 1 | 0 | N/A |
| 02 | 50 | 1 | 0 | N/A |
| 03 | 40 | 1 | 0 | N/A |
| 04 | 70 | 1 | 0 | N/A |
| 06 | 80 | 1 | 0 | N/A |

From these results it can be observed that changing the pseudo-derivative roll off frequency has two effects. The most obvious effect is to change the magnitude and frequency of the system's overshoot to the step change. However, this does not appear to be a linear trend. Further reduction or expansion of the roll-off frequency will probably make a minimal change in the response to the step change.

The second effect noted is the more concerning effect of introducing noise into the system. This occurs since the derivative is measuring a position change and dividing it by a very small time. Between two derivative calculations, a variation of one encoder count between two position change measurements yields a shift of 1.6 rad/s in the velocity determination. Since this tends to be a relatively high

frequency effect, turning down the roll-off frequency attenuates the differential noise. This however limits the system's responsiveness to higher frequency commands. The associated phase-lag induced by the lower cut off point would also explain the phase shift in the responses observed.

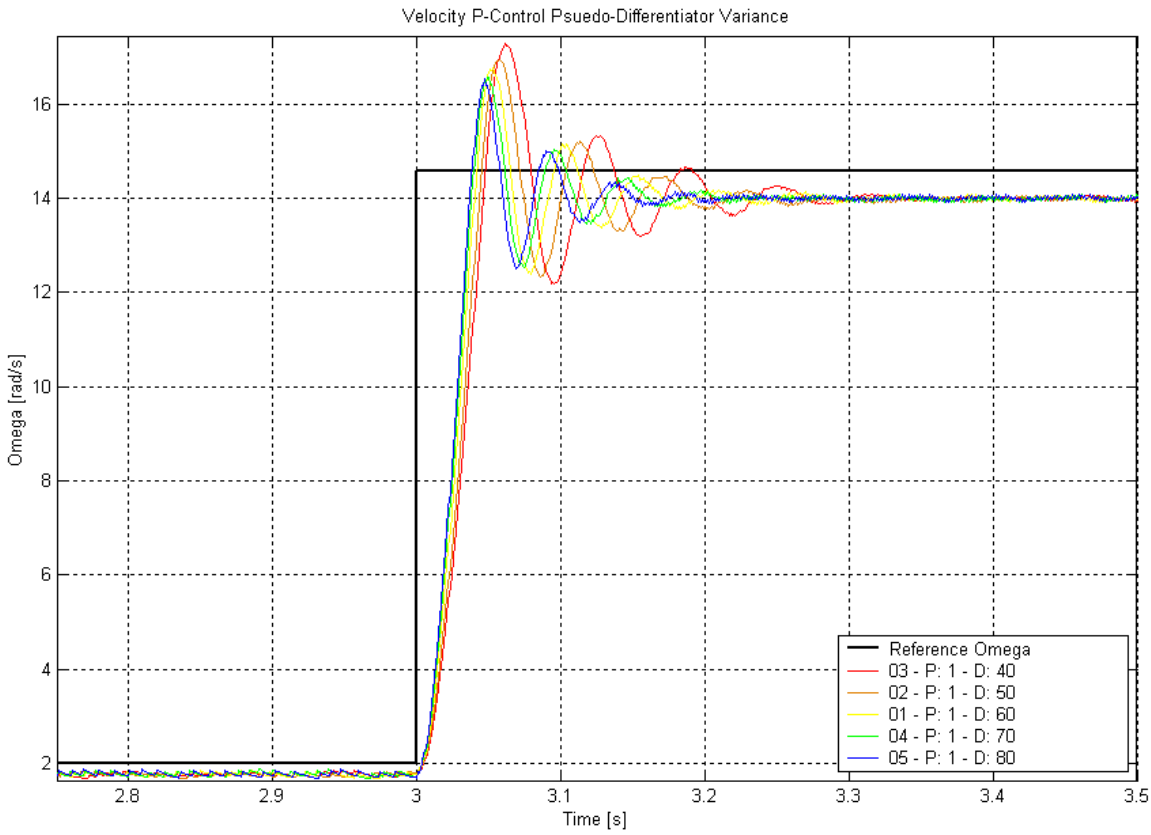


Figure 18: Pseudo-Derivative roll off frequency variational effect.

From these results a pseudo-derivative roll off frequency of 60Hz was chosen to serve as the constant roll off frequency while the proportional gain, $K_{vp}$ was varied. This frequency was chosen as the highest frequency that provided relatively little differentially introduced noise.

Table 2 below lists the parameters used for testing the effect of proportional gain variance. Figure 19 shows the resulting system responses. The approach used for variation was to determine the spectrum of responses from over-damped to marginally-stable. From their the proportional gain was varied until the response provided as close to the desired response without overshooting.

The results of this variation are as expected. As the gain is increased the system heads off towards marginal-stability. The limits imposed on the signal voltage being fed to the motor prevent the system from becoming truly unstable.

Table 6: Proportional gain variation runs

**Table 1 – Proportional gain variation runs.**

| Run | $\omega_{break}$ pseudo-derivative | Proportional Gain | Integral Gain | Velocity Prefilter Roll Off Freq. |
|---|---|---|---|---|
| 01 | 60 | 1 | 0 | N/A |
| 06 | 60 | 0.25 | 0 | N/A |
| 07 | 60 | 0.50 | 0 | N/A |
| 08 | 60 | 0.75 | 0 | N/A |
| 09 | 60 | 3.00 | 0 | N/A |
| 10 | 60 | 0.10 | 0 | N/A |
| 11 | 60 | 0.22 | 0 | N/A |

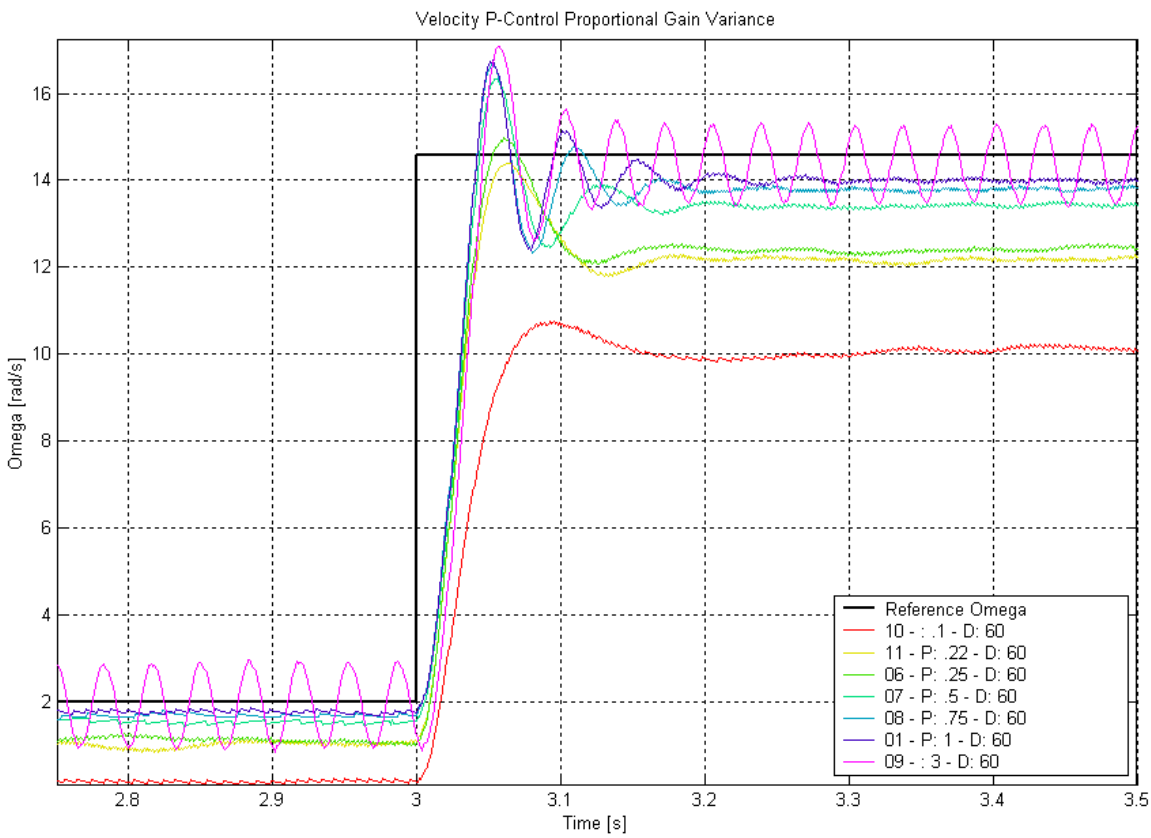


Figure 19: Proportional gain variational effect.

Based upon the results obtained by varying the proportional gain, a gain of 0.22 was chosen. This result is verified by the root-locus design performed using a linearized version of the motor model. The root-locus design yields a proportional gain of 0.219. The resulting Bode plot is shown in Figure 20. When

the linearized model is given a proportional gain of 0.22, the system is calculated to be unstable. This minor discrepancy is likely due to linearization errors.

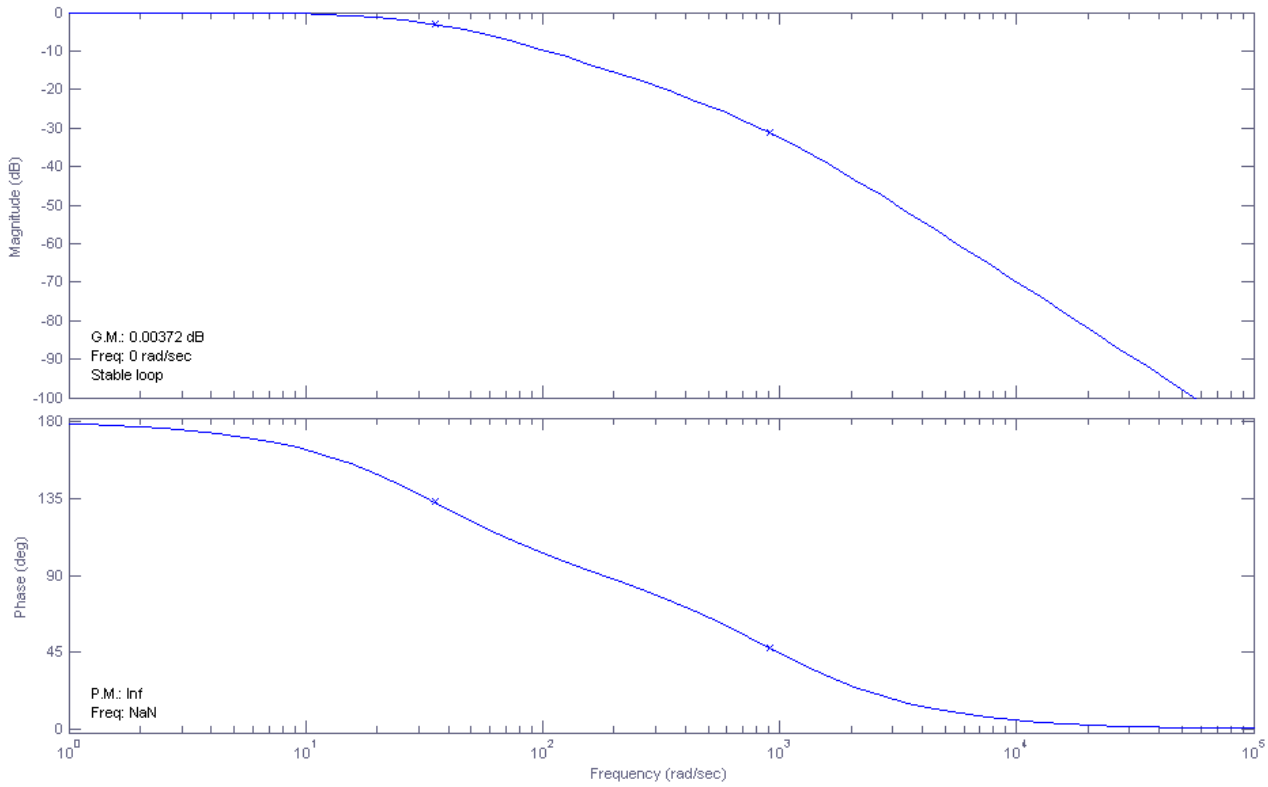


Figure 20: Bode plot of root-locus method proportional gain verification.

The system was then tested under the disturbance of the eddy current damper. This test was performed with the same parameters as used in Run 11. However, the eddy current damper was fully engaged prior to the start of the run. Figure 21 compares the responses with and without the eddy current damper (EC) engaged.

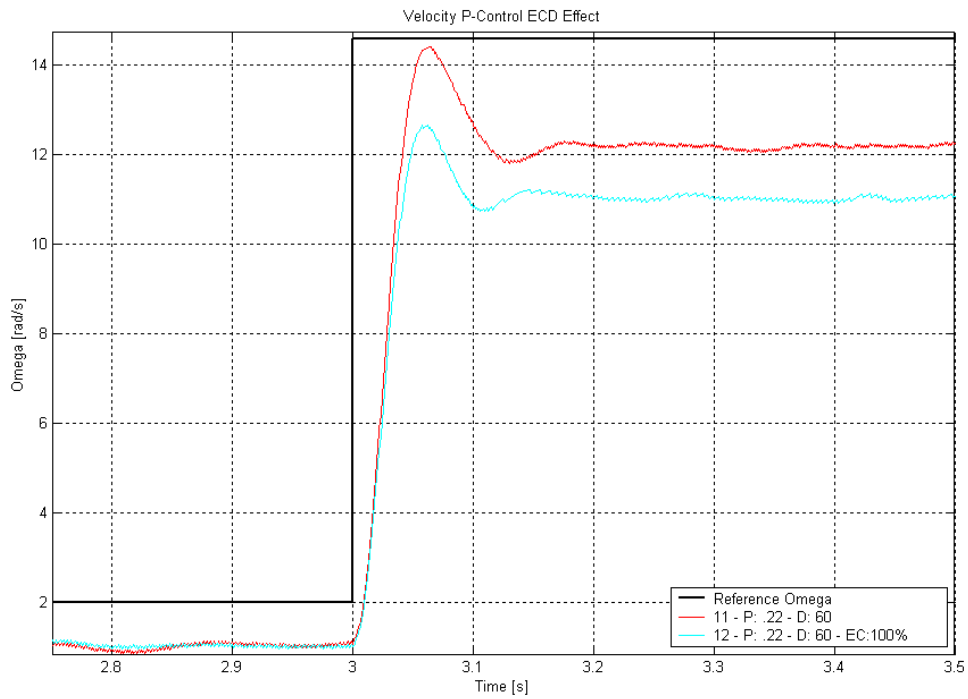


Figure 21: Eddy current damper disturbance effect.

As was expected, the eddy current damper increased the steady state error in the response of the system. Although this type of sensitivity is undesirable, correction cannot be made without the introduction of an integral component to the controller.

With the introduction of the integral component to the velocity controller, the parameters were again varied until a response with less than 20% overshoot and zero steady-state error was obtained. Figure 22 shows the final result obtained by introducing an integral controller. Additionally a *Velocity Command Prefilter* was introduced.

The ideal response was obtained with proportional and integral gains of unity, a pseudo-derivative roll-off frequency of 30Hz, and a pre-filter frequency of 30Hz. This response is compared to Run 11 in Figure 22. These values provided a system response that was relatively quick, yet still met the overshoot and steady-state error requirements. Variation of the pre-filter served to round off the step change commanded of the system. This reduced overshoot, but increased response time.

When the values obtained experimentally are compared to gain values determined through root-locus calculation, the results agree. The root-locus plot shows that attempts to push the integral gain farther would result in a loss of

performance in the system. The Bode plot shown in Figure 23 shows corresponds to the system as defined in Run 13.

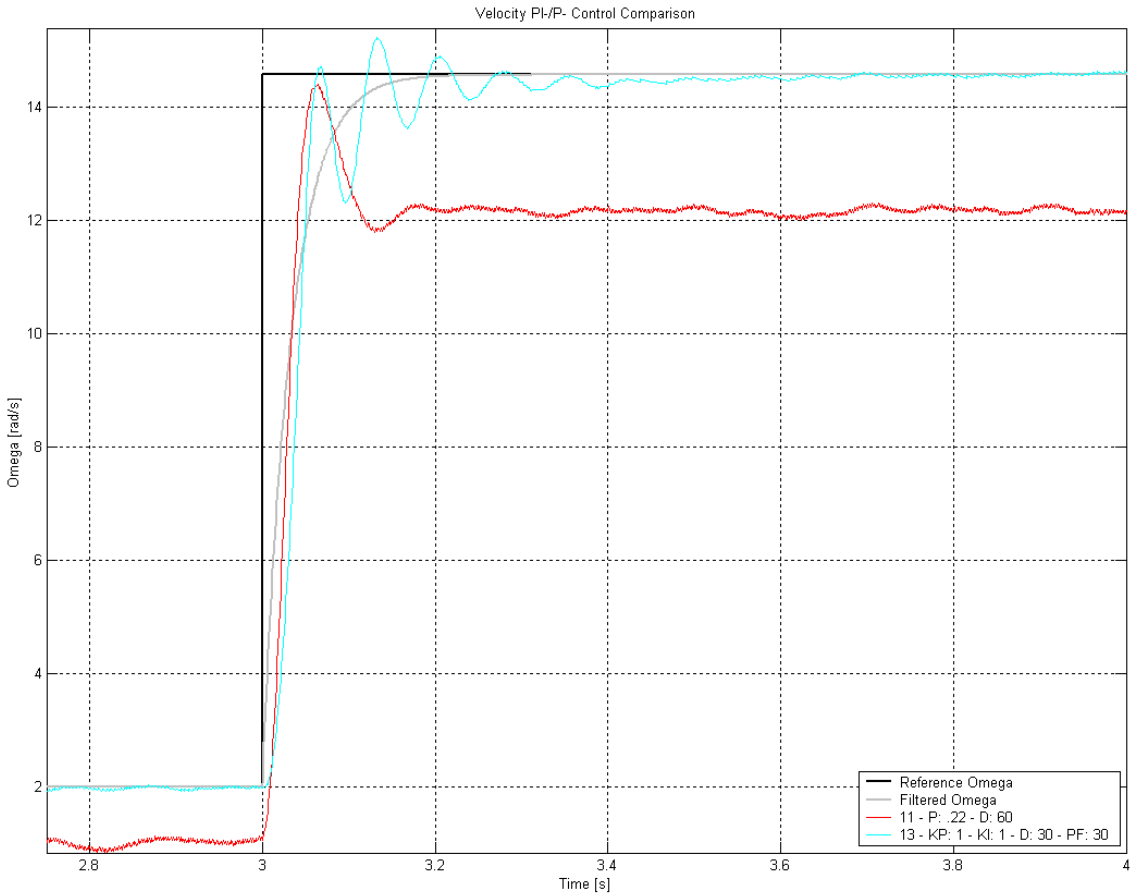


Figure 22 - Integral controller effect

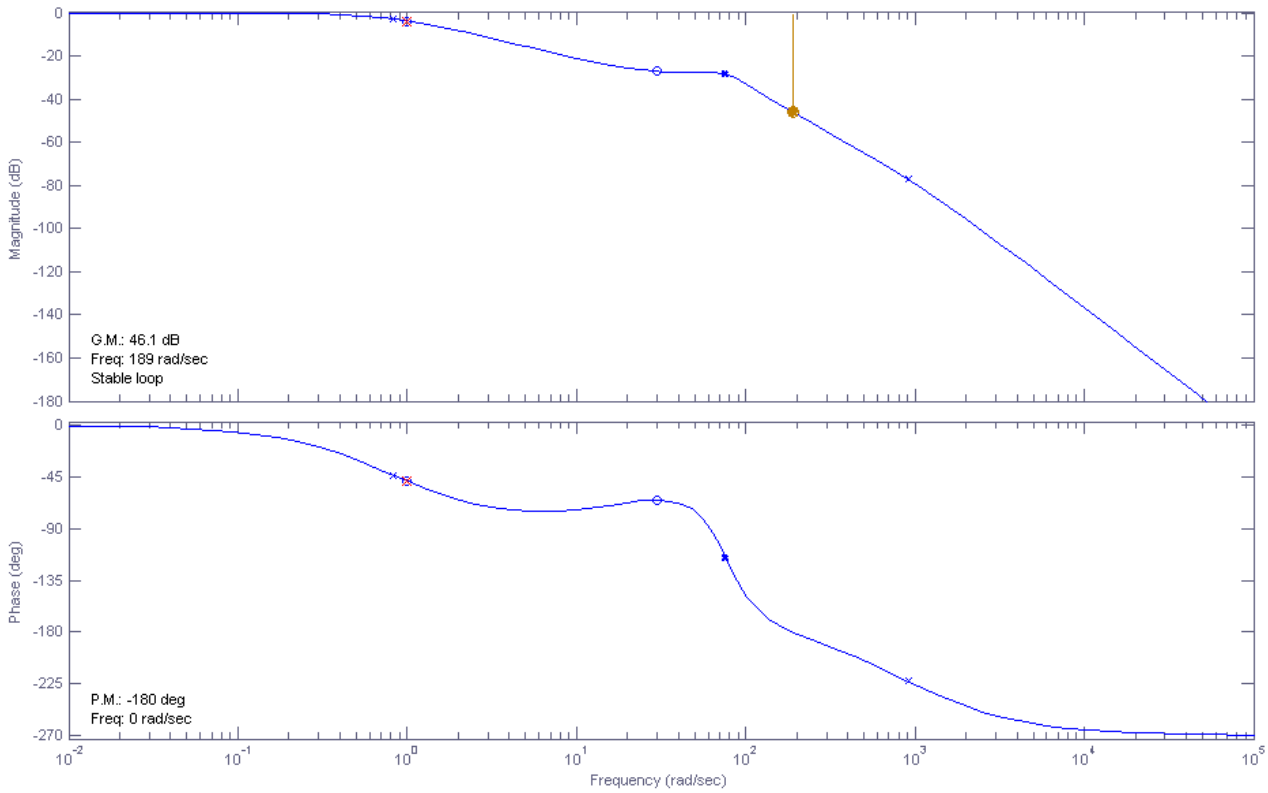


Figure 23: Bode plot for root-locus method designed integral and proportional velocity controller

**Position Control System**

Using the velocity controller parameters used in Run 13, work was begun on developing a position control system. The controller system was first toggled into position control mode by switching the *Velocity/Position Control Switch* into the downward position. Initially the *Position Command Prefiliter* block was omitted from the controller.

With the controller in this configuration a design was sought that would track a position step command of 180° with a response providing less than 10% overshoot and still allow for a quick response time. This was accomplished by varying the proportional gain, *KP*. Table 7 lists the parameters used for each run while Figure 24 shows the system's response to the step command.

Table 7: Proportional gain variance runs

| Run | Position Command Pre-Filter | Proportional Gain | Eddy Current Damper |
|---|---|---|---|
| 14 | N/A | 1 | 0% |
| 15 | N/A | 1 | 100% |
| 16 | N/A | 5 | 100% |
| 17 | N/A | 5 | 0% |

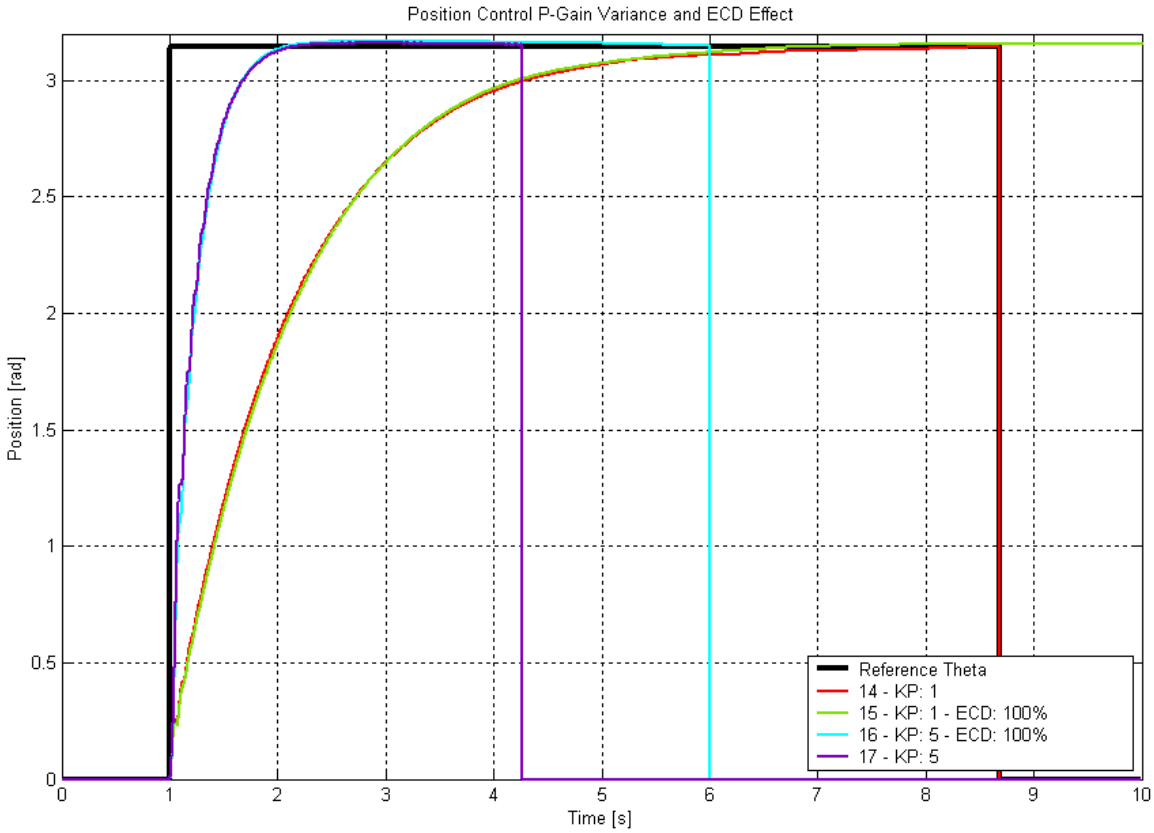


Figure 24: Position control proportion gain variance and ECD effect.

From these results it is clear to see that the system is almost untouched by the addition of the eddy current damper. For both proportional gains used, the eddy current damper provided little or no disturbance to the system. As was expected, the increase in the proportional gain quickened the response. The cost for this increase in speed was an increase in overshoot.

Lastly, a position control pre-filter was added in. The effect of varying the pre-filter frequency in addition to further modification of the proportional gain is shown in Figure 24. Table 8 lists the system parameters specified for each run.

Table 8: Proportional gain variance runs

| Run | Position Command Pre-Filter | Proportional Gain | **Eddy Current Damper** |
|---|---|---|---|
| 18 | 10 | 15 | 0% |
| 19 | 20 | 15 | 0% |
| 20 | 20 | 7 | 0% |

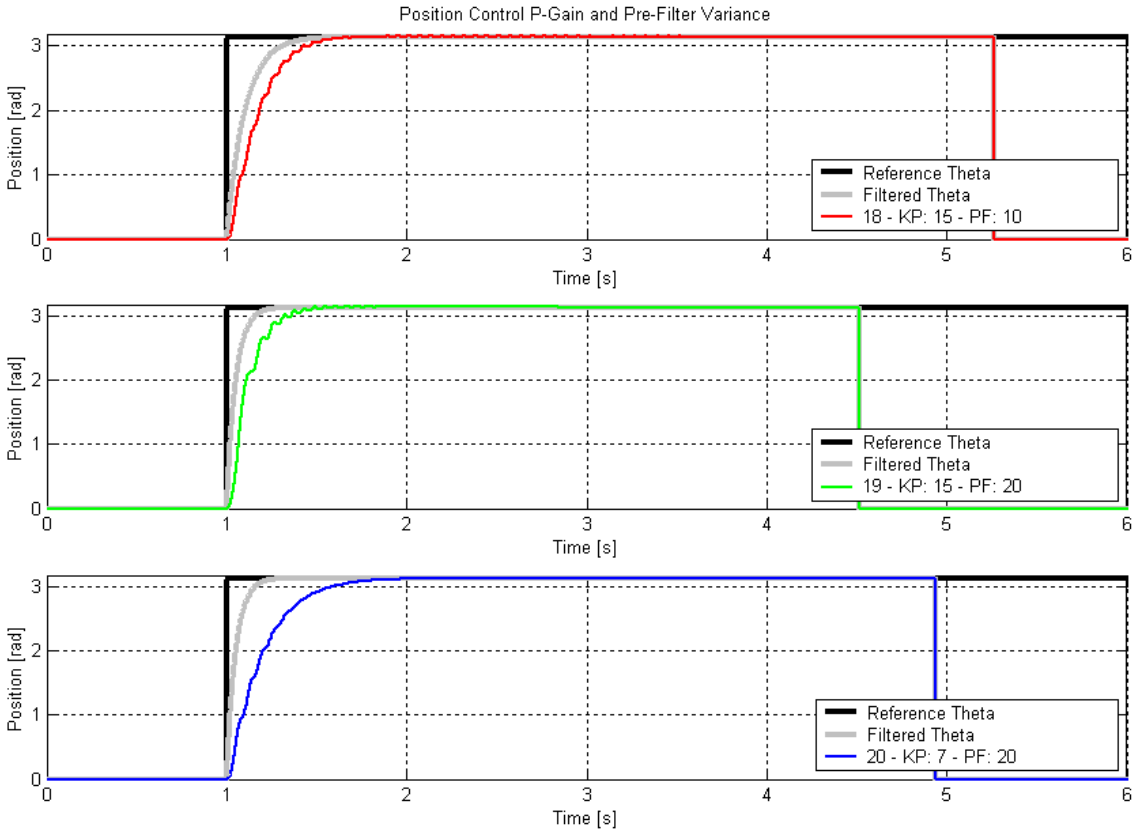


Figure 24: Proportion gain and position pre-filter frequency effects.

Analysis of the results obtained by the addition of the pre-filter indicate that throttling back on the step command allows the system to more easily respond without overshoot. The pre-filter serves to round off the step command and thusly ease the frequency at which the system is commanded to respond.

The drawback to the addition of the pre-filter is that it introduces lag into the system. Since the system is not being required to respond at its maximum rate, the system is slower to attain its final steady state condition. This is as expected since for any given time sample, the proportional error in the position as well as the proportional and integral error of the velocity are going to be less in the presence of a pre-filter. Consequently the system is not driven as hard.

An interesting phenomenon was the introduction of an apparent transitory tracking oscillation. At increased proportional gains, the position tracking would track the filtered command signal in oscillatory manner. This is perhaps generated as the system in a sense overshoots the current command signal and is forced to ease back.

When the motor model is linearized, a root-locus design approach can be taken to calculating the optimal proportional gain. When given a pre-filter frequency of 20 Hz, the root-locus method yields an optimal proportional gain of 41. Figure 25 shows a Bode plot of the root-locus designed optimal system. This is quite large compared to the experimental value of 7. The experimental value appears satisfactory in a root-locus design environment, yet does not yield the fastest response. It is likely that running the real system with a proportional position control gain of 41 would yield suitable results. However, it is highly likely that the oscillatory rise phenomenon described previously would be significantly more prominent.

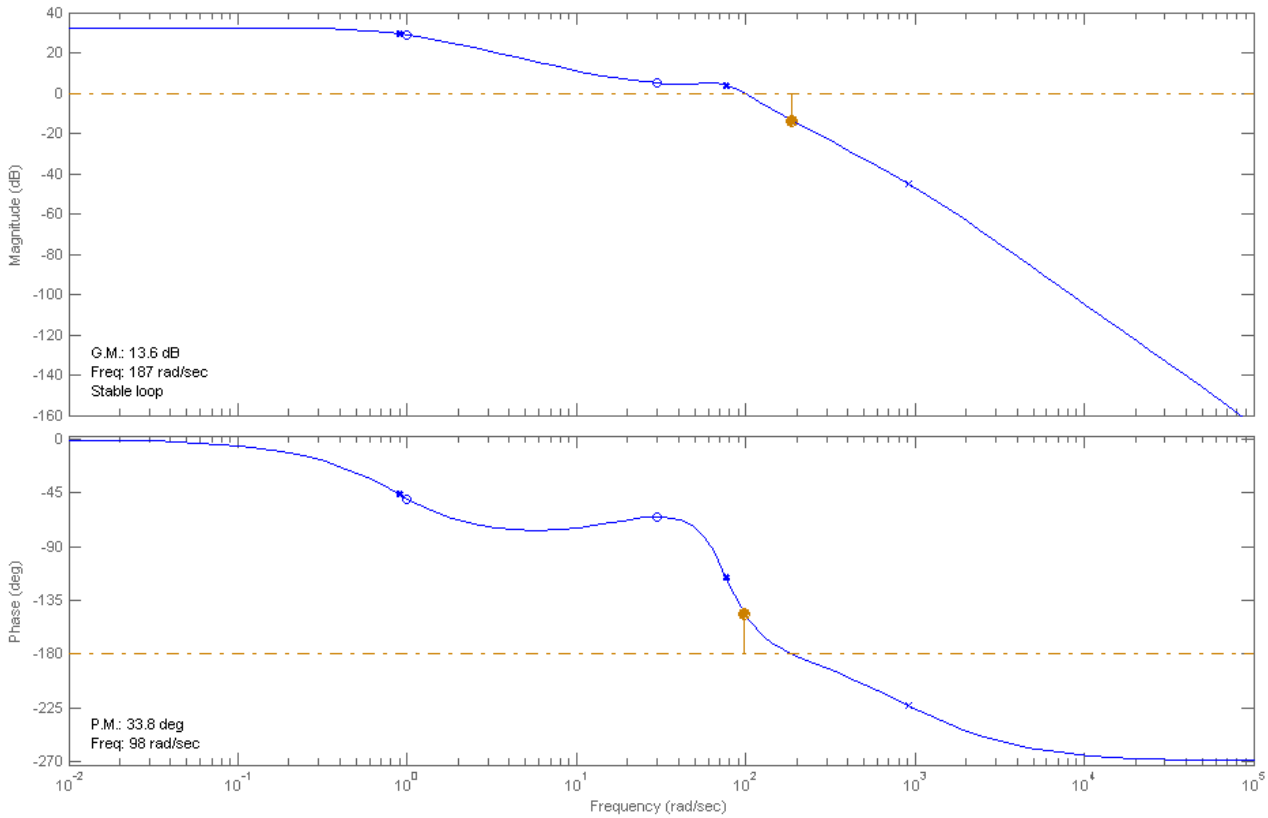


Figure 25: Bode plot for root locus optimized position control.

## VI. Experimental Procedure

This lab consisted of two separate portions. The first was to setup the test apparatus and collect data to determine the physical parameters of the motor. The second was to design velocity and position control for the motor using PID controllers. The first step of the lab was to setup up the apparatus as shown below:

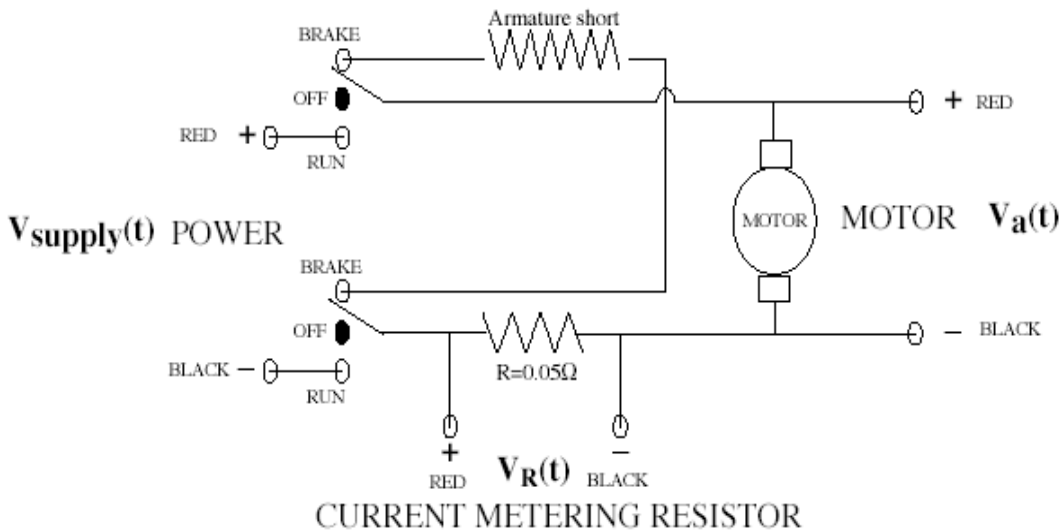


Figure 26: DC motor power connections

Next, the rotor disk was locked in place using the lock key through the holes of the disk, and supplied voltages was varied from –4 to 4 V. The motor switch was toggled to run and data was recorded for Va, Vr, and rotor position. The disk position was then changed and data was again recorded. This portion of the data was used to determine the value of Rm.

Next, to determine La, two different methods were used. First, the transient response of i(t) was captured using the oscilloscope, and second, the frequency response was recorded using the DSA. For the transient response the motor disk was again locked in place and channel 1 of the oscilloscope was connected across the current metering resistor. The oscilloscopes scales were adjusted to 50 mV/div vertically and 5 msec/div horizontally in order to see the response more accurately. Also, the rising edge triggering mode was selected on the oscilloscope. In order to capture the first order response, the oscilloscope was turned on, and immediately after the motor was toggled to run. Once captured, the response was smoothed using the Waveform->Acquire->Averaging buttons and setting the number of avgs to 10. To save this data you select Utilities->Print Configuration->Format->CSV Data->Print to: Disk->File->Quick Print. For the

DSA analysis, the motor disk was locked and the source of the DSA was connected to Channel 1 of the DSA and the Voltage Programming Input of the Kepco Power Supply. The first method using the DSA was the swept-sine method where the DSA provides a swept sine input signal to the system over a defined range of frequencies. In our case this was performed between 10Hz and 10kHz. The detailed setup for this method can be seen in the figure below:

| Hard Key | Soft Key1 | Soft Key2 |
|---|---|---|
| **Preset** | **Do preset** | **2 Channel** |
| | | **Swept-sine** |
| **Disp Format** | **Bode Diagram** | |
| **Meas data** | **Frequency Response** | |
| **Trace coor** | **dB Magnitude** | |
| | **X-axis** | Toggles **Log** |
| **Active trace** | | |
| **Trace coor** | **phase** | |
| | **X-axis** | Toggles **Log** |
| **Freq** | **start** | ***10* Hz** |
| | **stop** | ***1* kHz** |
| | **Sweep** | Toggles **Log** |
| **Source** | **Level** | ***0.2* Vpk** |
| **Start** | | |

Figure 27: Detailed description of DSA Swept-Sine method setup. Hard key refers to the buttons on the control panel and soft key refers to the screen selection buttons.

To calculate the frequency responses and Bode plot of the transfer function, the voltage across the Current Metering Resistor was input into Channel 2 of the DSA. The Bode plot results were saved using the Save/Recall command in the System section of the control panel. This test was repeated at two different rotor positions. The second portion of the DSA analysis was performed using the Random Noise mode. The detailed setup for this method can be seen in the figure below:

| Hard Key | Soft Key1 | Soft Key2 |
|---|---|---|
| **Preset** | **Do preset** | **2 Channel** |
| | | **FFT-Analysis** |
| **Disp format** | **Bode Diagram** | |
| **Meas data** | **Frequency Response** | |
| **Freq** | **start** | ***10* Hz** |
| | **stop** | ***1* kHz** |
| **Avg** | **Average** | Toggles **ON** |
| | **Number of averages** | ***10*** |
| **Source** | **Level** | ***0.2* Vpk** |
| | **Random Noise** | |
| | **Source** | Toggles **ON** |
| **Start** | | |

Figure 28: Detailed description of DSA Random Noise method setup.

After the DSA was setup for the Random Noise method, the same procedures were followed to obtain Bode plots.

The next portion of the lab was using varying load testing to determine K, the machine constant. First the oscilloscope was connected to both Channels A and B of the encoder. The motor was toggled to run using both positive and negative values of $V_a$ in order to observe the phase relationships between Channels A and B. These plots were also saved. Next, the encoder frequency was determined using the table below:

| Encoder Type | Frequency | Motor Speed | | |
|---|---|---|---|---|
| (lines) | (kHz) | (RPS) | (RPM) | (RAD/SEC) |
| 1000 | 1.0 | 1.0 | 60 | 6.2832 |
| " | 10.0 | 10.0 | 600 | 62.832 |
| " | 20.0 | 20.0 | 1,200 | 628.32 |
| 500 | 1.0 | 2.0 | 120 | 6.2832 |
| " | 10.0 | 20.0 | 1,200 | 62.832 |
| " | 20.0 | 40.0 | 2,400 | 628.32 |

Figure 29: Relationship between encoder signal frequency and motor speed.

Next, the supply voltage was set to 8V and the Eddy Current damper was fully engaged. The motor was toggled to run, and when steady state was reached, we measured $V_r$ across the current metering resistor and the frequency of rotation. The eddy current damper was then decreased in ~1/4 increments and values for $V_a$, load fraction (eddy damper engagement), $V_r$, and frequency we recorded.

This was repeated for 3 different supply voltages, one of which was opposite polarity.

The last physical parameter of the motor to find was the motor torque function, $b_m(\omega)$. To do this, the eddy current damper was fully disengaged and the armature voltage, $V_a$, was set to 10V and $V_a$, $V_r$, and $f_{rotation}$ were recorded. The supply voltage was adjusted in order to achieve this. This measurement was repeated for various values of $V_a$ ranging from –10V to +10V. Also recorded was the range of voltages where the rotor did not spin, or the dead zone.

During the second week of lab, the only procedure was to determine $J_m$, the motor drive inertia. To do this, the eddy current damper was fully disengaged and the setup was wired as shown in the following figure:

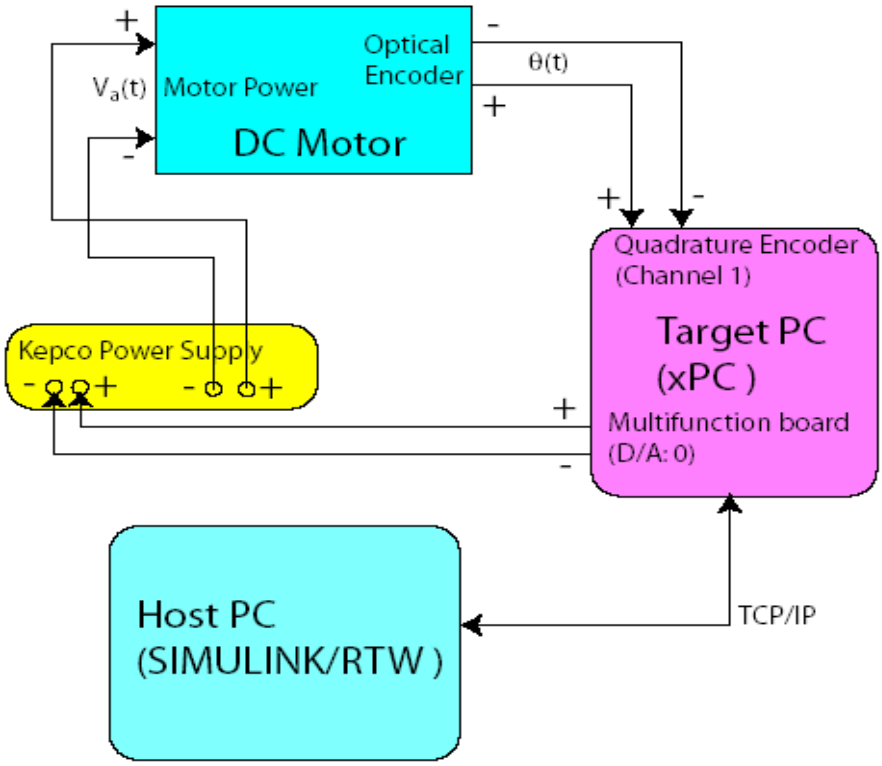


Figure 30: Motor and computer setup to determine $J_m$

The Kepco proportional amplifier gain, Ka, was first determined by supplying a constant voltage to the amplifier, and measuring the armature voltage out of the amplifier. The Simulink model was modified so that it interfaced with the Quadrature encoder board, CIO-QUAD04, and the analog output of the multifunction board, CI0-DAS1602 16. The Simulink model used to interface with the xpcTarget is shown later in the Model Validation section.

The analog output was also connected to the proportional amplifier input of the kepco power supply. Then, using 2 step input sources applied at 2 different start times, a pulse was sent into the armature voltage to rotate the rotor approximately $\pi$ radians, 180 degrees. The transient responses of the encoder were then saved.

During week 3 we were to design a feedback control system to track both velocity and position commands. This was split into 2 parts, a proportional control and a proportional integral control. Once again, the Simulink model used is shown in the Model Validation Section.

The PI controller was then inserted into the model for velocity control. First, the integral gain, Kvi, was set to 0 and the proportional gain, Kvp was set to a value which provided good tracking responses to a step command in velocity of $\omega ref$=2 rev/sec. The design objectives were: No overshoots, high control bandwidth, and good tracking to command and disturbance rejection. The eddy current damper was used to introduce a disturbance into the system so the steady state error of the commanded responses could be observed. This procedure was repeated using both Kvi and Kvp gains. The design objectives were: Less than 20% overshoot, high control bandwidth, reasonably fast rise and settling times, and good tracking to command and disturbance rejection. Again the eddy current damper was used to observe steady state error of the commanded responses due to disturbances.

The last portion of the lab was to design a feedback control system to track position commands consisting of a Proportional control and a prefilter. The Simulink model used was the same as the velocity control model except for the manual switch was toggled down to include the outer loop for position control. First the proportional feedback gain, Kp, was designed. The design objective was to achieve good tracking responses with less than 10% overshoot to a step command of $\pi$ radians. The steady state error responses were again observed as a constant force was applied to the rotor disk. A prefilter was then added to the model to examine whether or not it would improve the tracking responses of the system. Different bandwidths for the prefilter were used, and the data was saved.

## VII. Modeling Validation

### Open Loop

The parameters derived in the System Modeling section can be applied building a Simulink model in order to validate their accuracy. The output of the Simulink model can then be compared to the output of the actual motor. In order to control the motor, the following Simulink model was built as shown below in Figure 31.

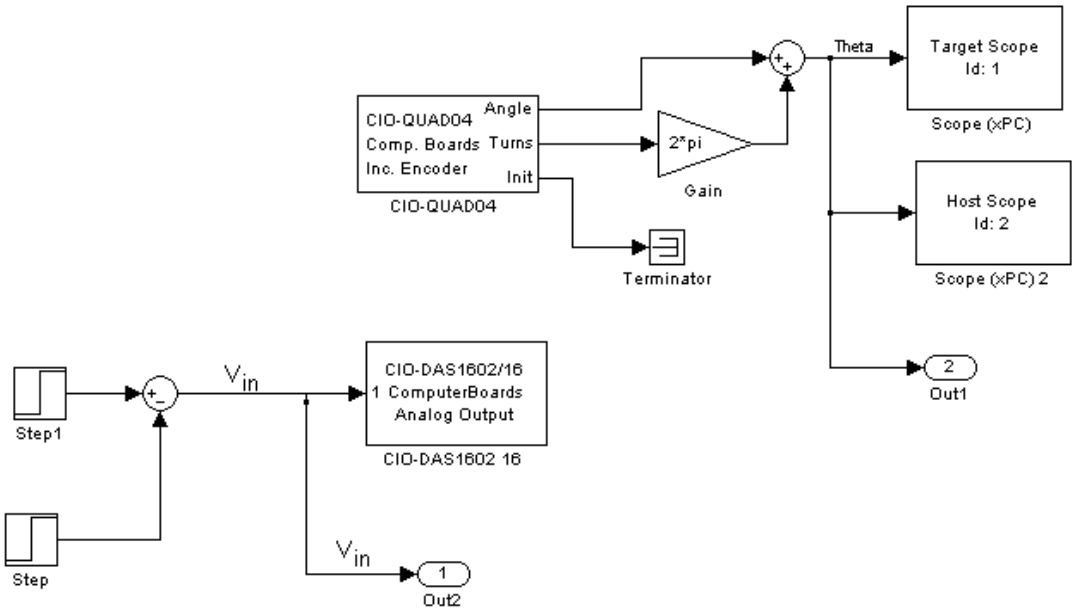


Figure 31: Simulink model used to interface with xpcTarget to produce a step in voltage

The Simulink model used to simulate the actual motor is shown below in Figure 32.

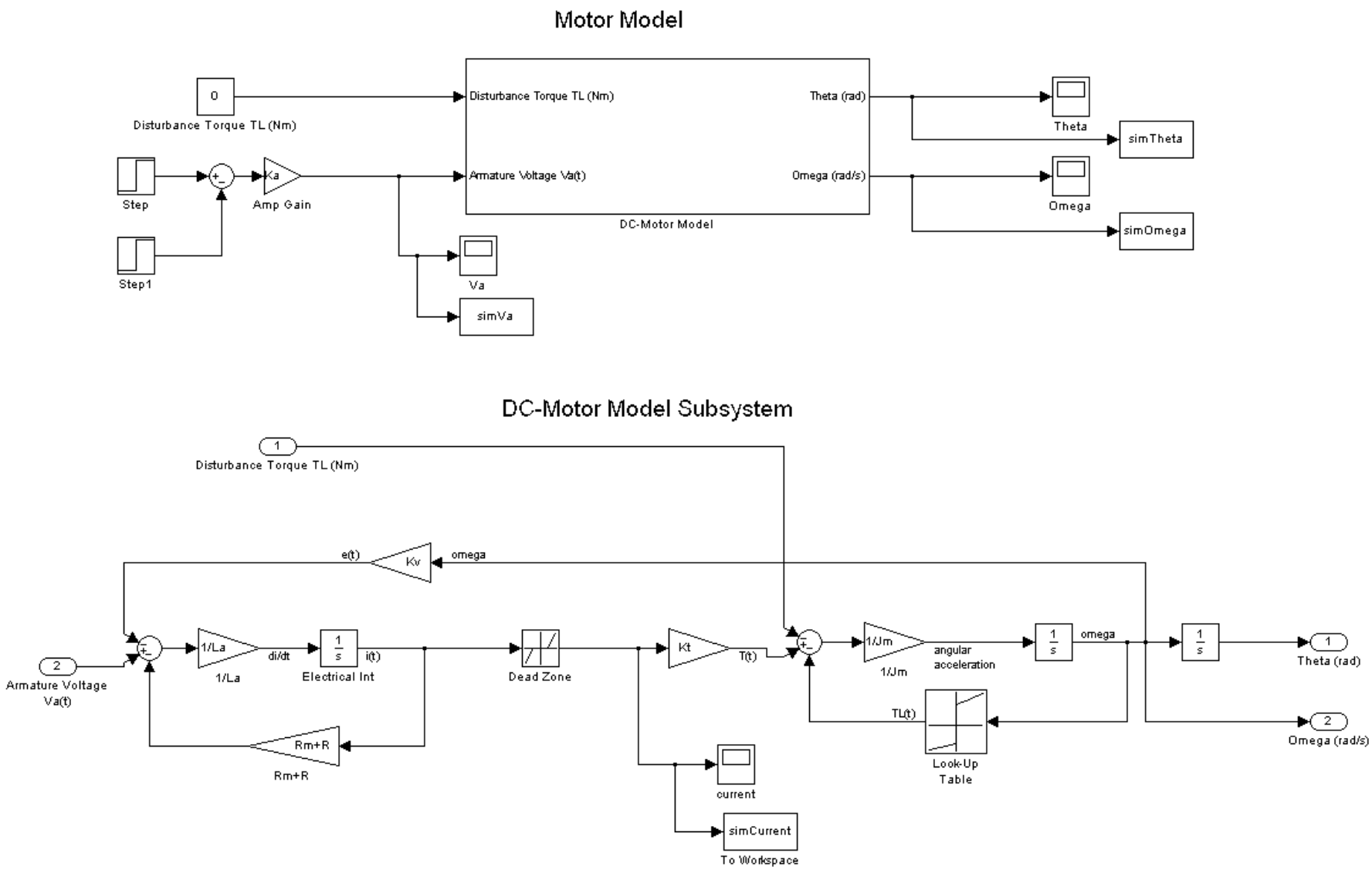


Figure 32: Simulink model used to approximate open loop pulse voltage signal

This Simulink model creates a step of magnitude 1 for 0.13 seconds which is sent to the input of the amplifier. The actual pulse sent to the amplifier and the simulated pulse sent to the amp are shown below in Figure 33.

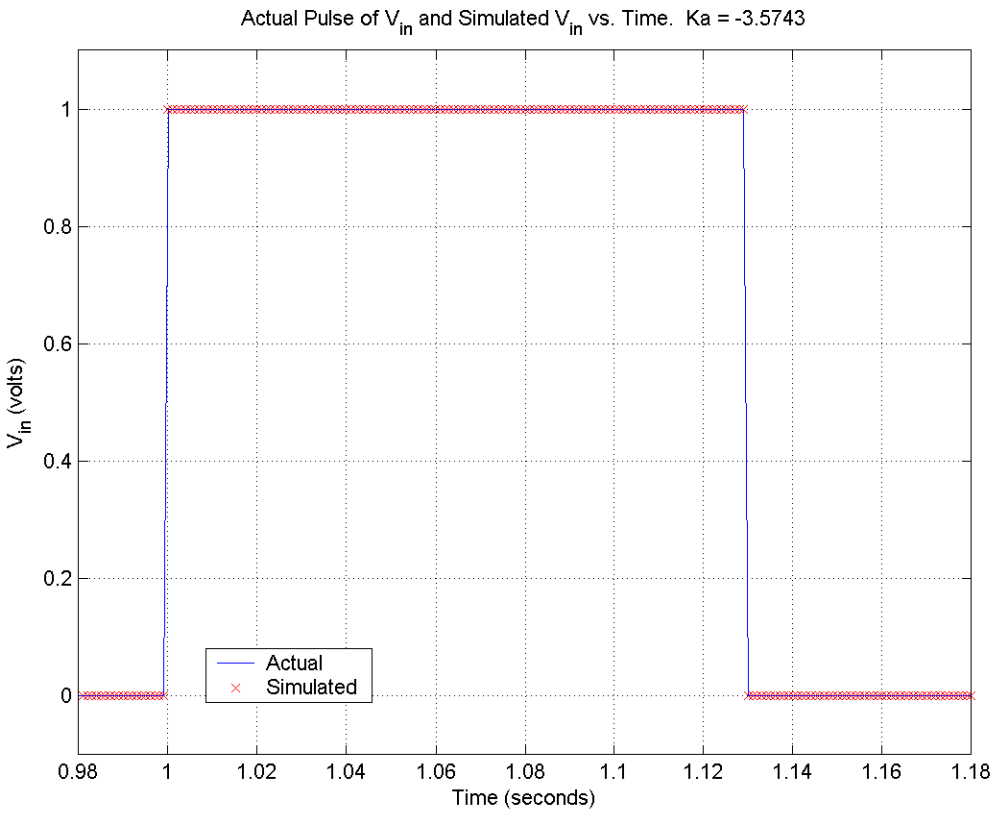


Figure 33: Simulated and actual $V_{in}$ used to turn wheel approximately 180 degrees

As can be seen, the pulse commanded in lab and the pulse generated in the simulation are virtually identical. Keep in mind that these are values of $V_{in}$, the voltage before the amplifier.

The wheel rotation is measured using the encoder. The actual and simulated angular displacements are shown below in Figure 34.

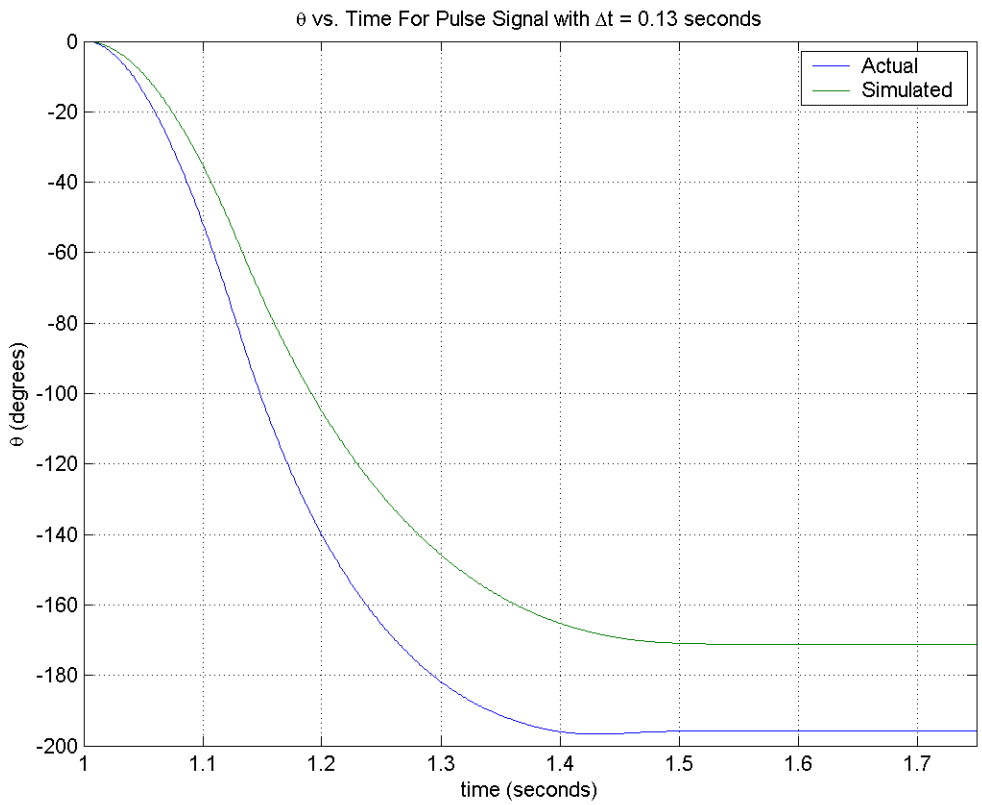


Figure 34: Actual and simulated θ vs. time in response to a pulse $V_{in}$

As can be seen, the model is fairly accurate. The general shape follows the lab experiment fairly well. However, there is a small steady state error or roughly 25 degrees. There are several reasons for this. First of all only $V_{in}$ not $V_a$ was measured in the lab. The gain of the amplifier was not measured during the application of the pulse. If the gain was slightly different or if the amp takes some time to adjust the gain, it is possible that $V_a$ in lab is different from $V_a$ used in simulation. If the gain were to shift by only 10% during the 0.13 seconds where the step is applied, Figure 35 is obtained.

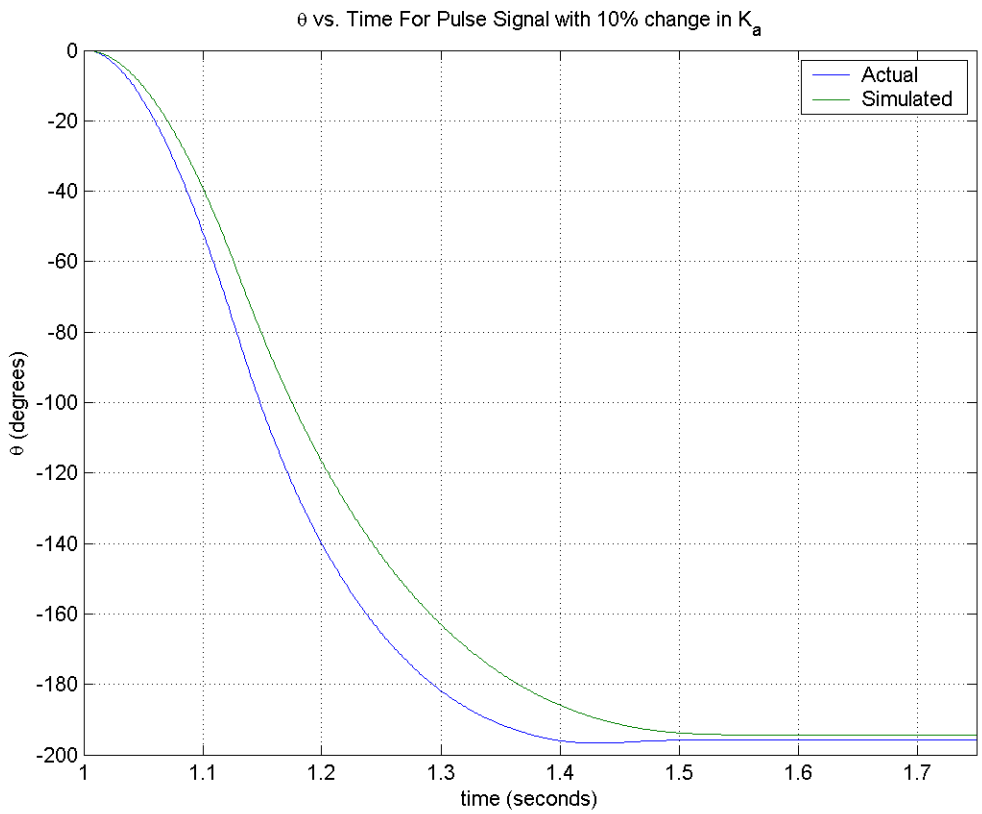


Figure 35: Actual and simulated θ vs. time in response to a pulse $V_{in}$ with a 10% change in $K_a$

As can be seen, this small shift in $K_a$ yields a very accurate model. Another possible place for error is the fact that the Coulomb friction is modeled as a look up table shown previously in Figure 14. This could cause differences in the values of θ at large t.

**Velocity Controller**

The next level of validation is to evaluate the performance of the closed loop system. The inner loop consists of the a velocity controller which maintains a set velocity. The Simulink model used in lab to obtain data is shown below in Figure 36.

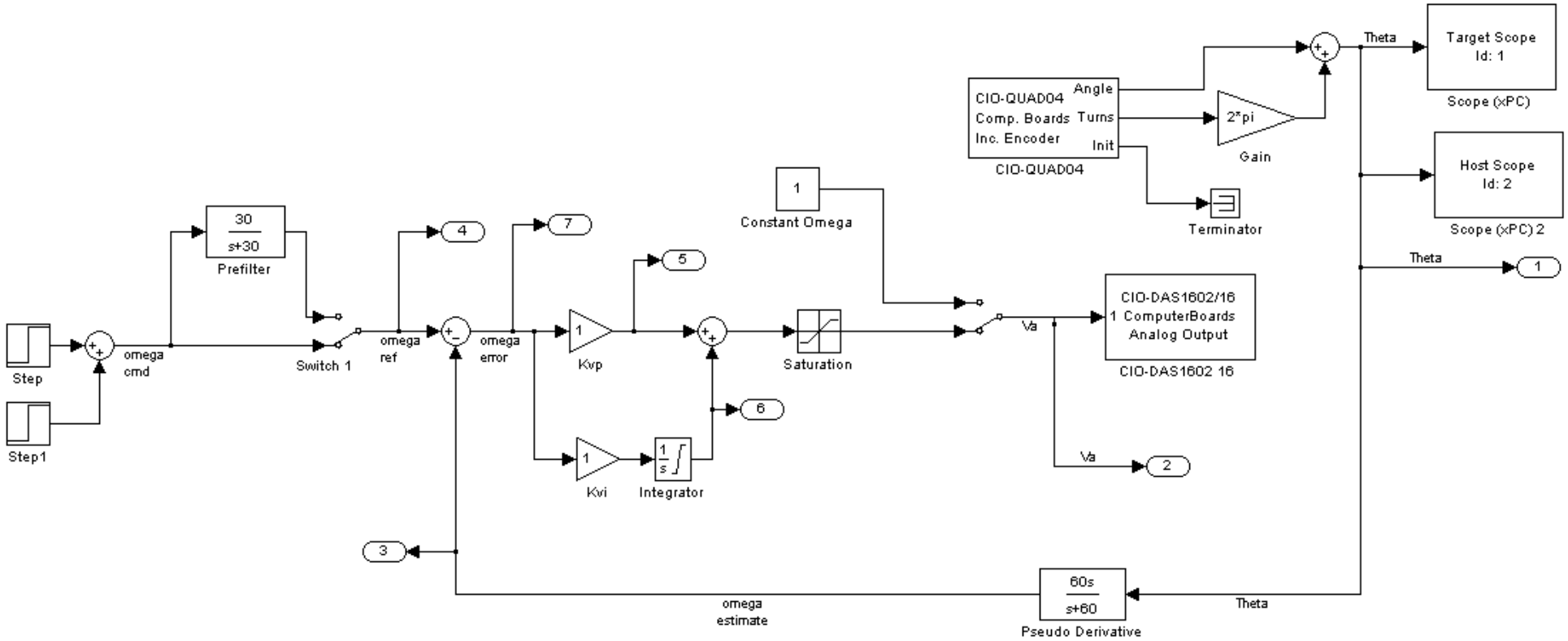


Figure 36: Simulink model used to interface with xpcTarget to track a velocity step of 12.5664 rad/s starting at 2 rad/s

This model is used to create a step in reference velocity. In order to avoid the stiction problems, the initial velocity is set for 2 rad/s. The motor is set to spin at 2 rad/s for two seconds then a step of magnitude of 12.5664 rad/s is introduced (2 rev/s). Initially in lab, the gain $K_{vi}$ was set to zero and $K_{vp}$ was varied until zero percent overshoot was achieved. The prefilter was not used and the pseudo derivative block is as shown in Figure 36. The parameters needed to achieve this were found experimentally to be

$$\mathbf{K_{vp} = 0.22}$$

The Simulink model used to simulate the lab environment is shown below in Figure 37.

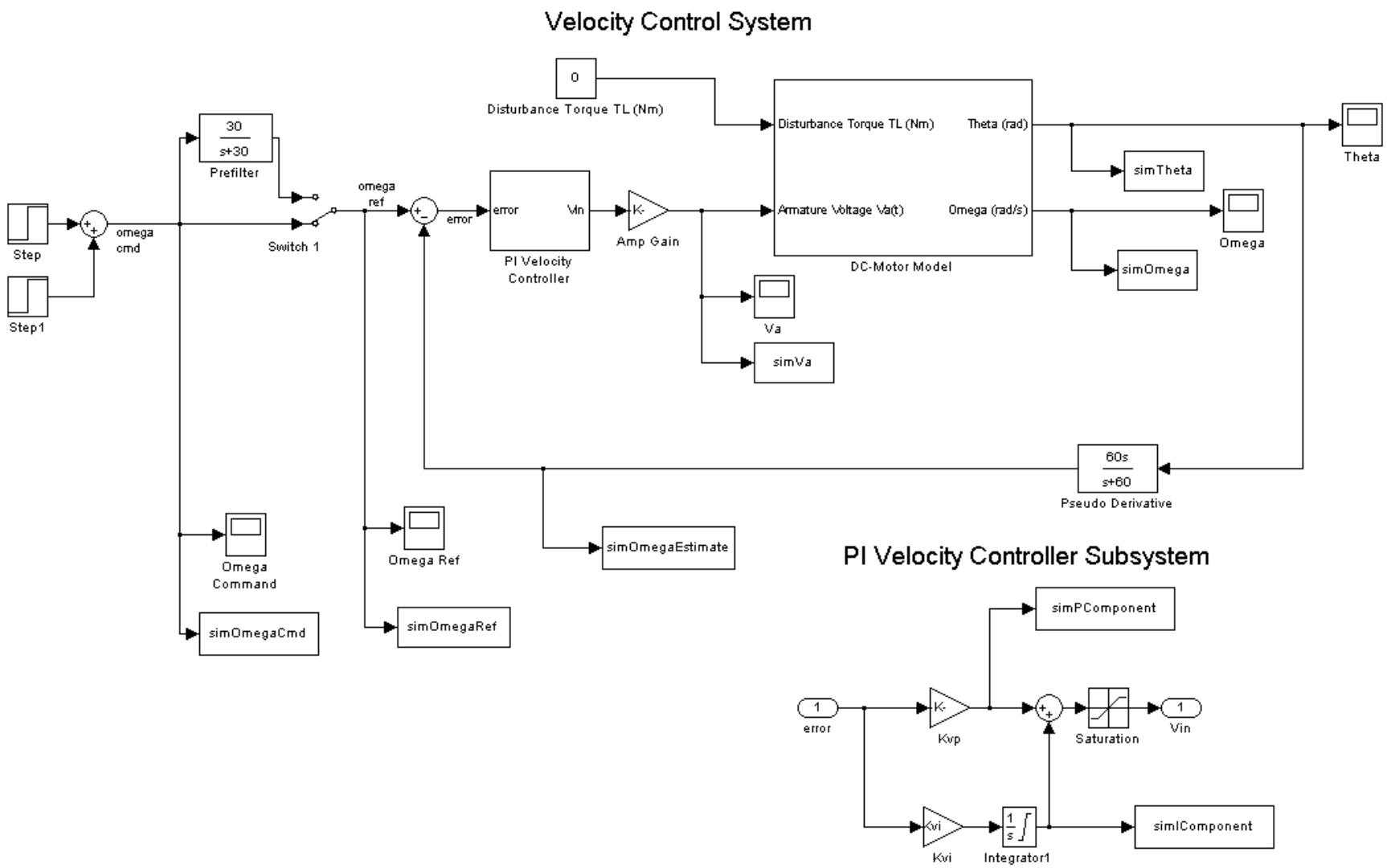


Figure 37: Simulink model used to simulate DC motor with velocity step of 12.5664 rad/s starting at 2 rad

The simulation is then run in order to compare the results. The lab data and simulation output is shown below in Figure 38 and Figure 39.

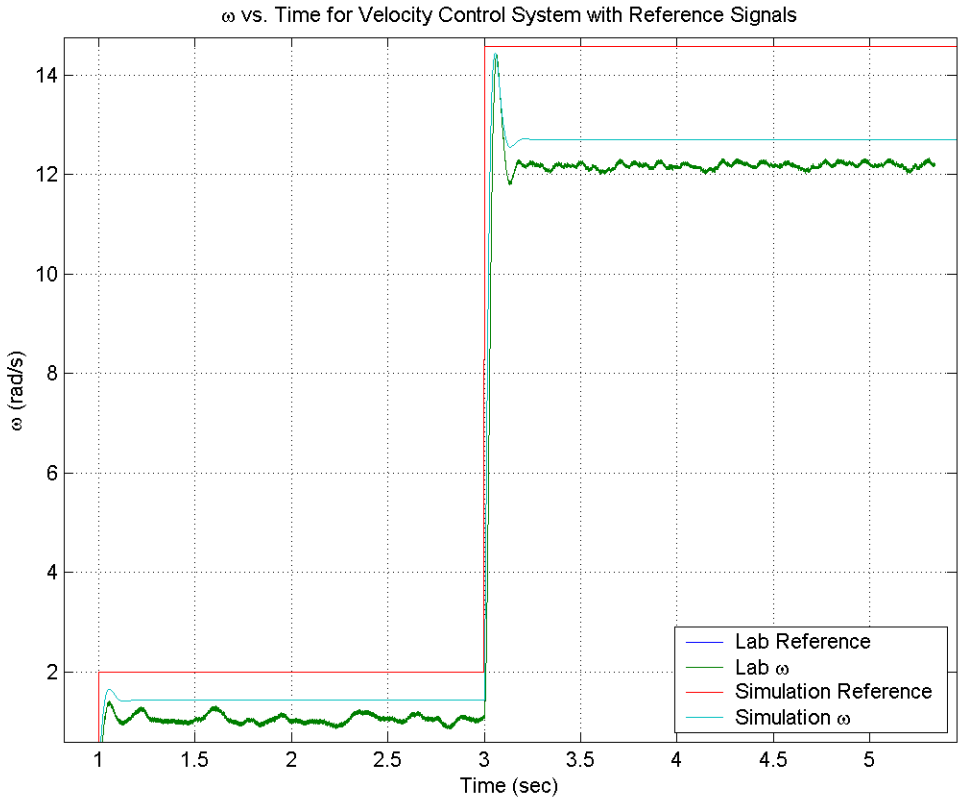


Figure 38: ω vs. time for step in velocity reference with $K_{vp} = 0.22$

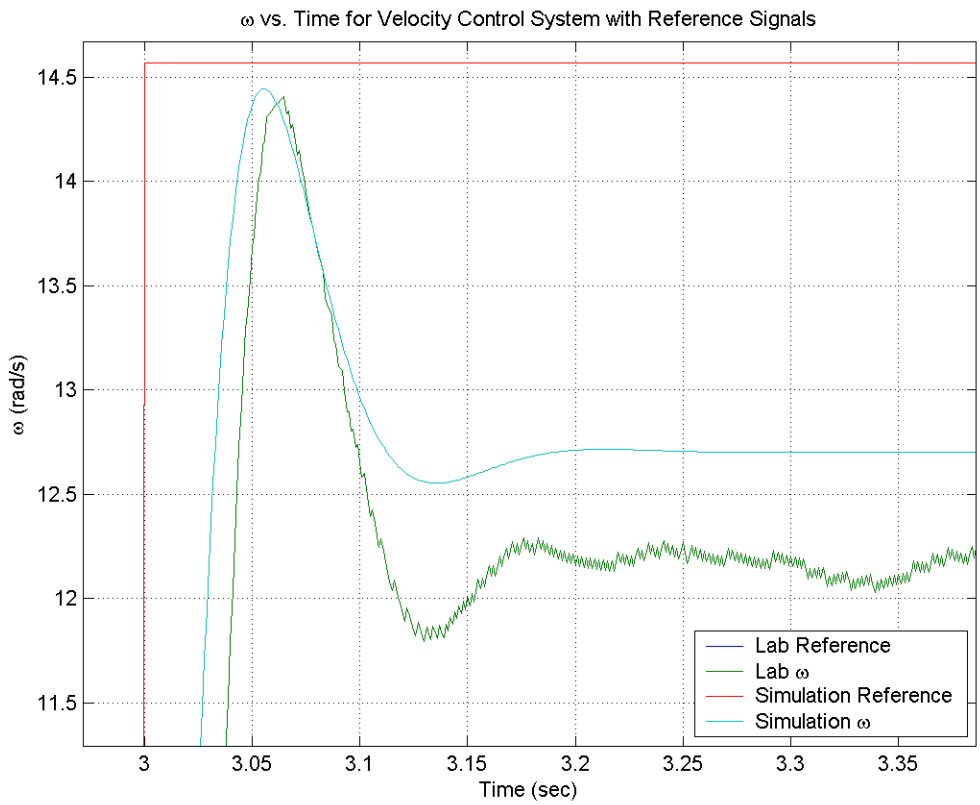


Figure 39: Closeup of overshoot ω vs. time for step in velocity reference with $K_{vp} = 0.22$

As can be seen, the simulation matches the lab data very well. Both the lab data and the simulation have roughly the same percent overshoot and the same shape. It appears that the simulation has slightly less steady state error than what was observed in lab, but considering the number of approximations made in deriving the simulation parameters, the simulation appears to be remarkably accurate.

The simulation must also be able to model a disturbance torque as well. In lab, this is provided in the form of the eddy current dampener. In the simulation, the disturbance torque is simply modeled as a constant. Since there is no way to determine the relationship between ω and $T_L$, the constant value in the simulation is varied until the simulation matches the lab data. The response is shown below in Figure 40 with a constant disturbance torque of $T_L = 0.083$ Nm.

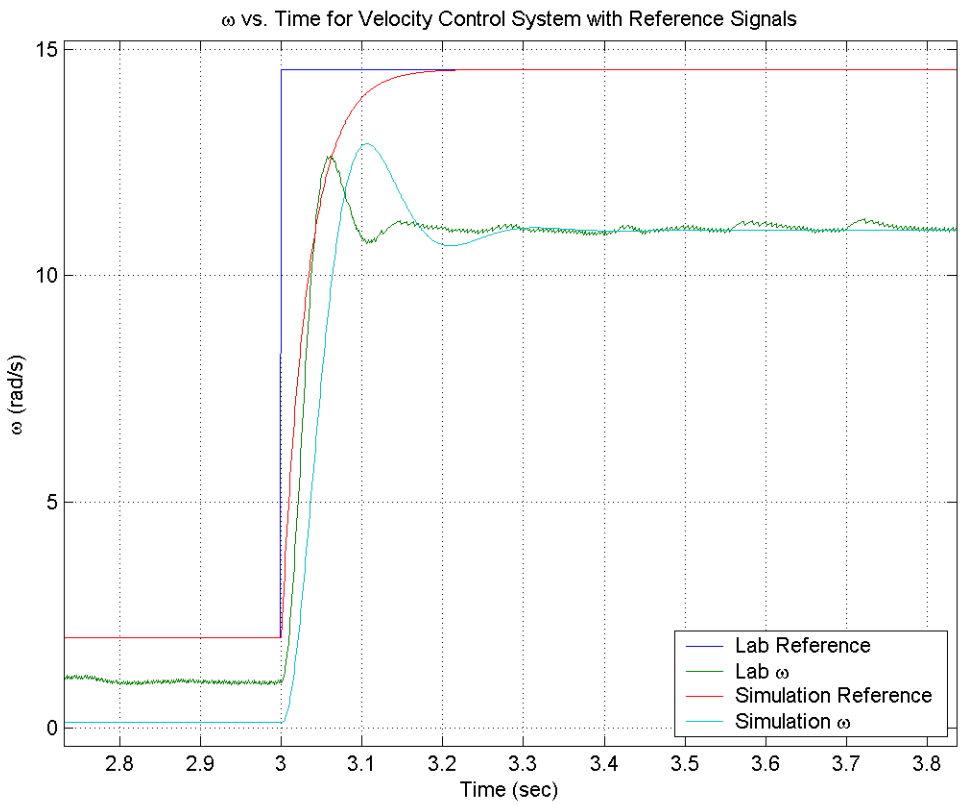


Figure 40: ω vs. time with constant disturbance torque $T_L = 0.083$ Nm

As can be seen, it appears that at steady state, the eddy current brake when fully in provides roughly 0.083 Nm or resistance torque. The interesting thing to notice is that at small omega, it appears that the simulation is not consistent with the lab results. This is because in the simulation, the disturbance torque is modeled as a constant. However the eddy current brake system provides a torque that is proportional to ω. Therefore, it would be more accurate to model the disturbance torque as a function of ω.

Next, the integral gain, $K_{vi}$ is varied to achieve the design objectives of less that 20% overshoot, high control bandwidth, reasonably fast rise and settling times, and good tracking to command and disturbance rejection. In order to achieve this, it was found that the gain $K_{vp}$ and the break frequency of the pseudo-derivative block need to be changed. In addition, a prefilter was also added. The parameters necessary to obtain the design objectives are shown below.

**$K_{vi} = 1$**
**$K_{vp} = 1$**
**$\omega_{break} = 30$ rad/s (pseudo derivative)**
**$\omega_{break} = 30$ rad/s (prefilter)**

The comparison between the lab data and the simulation is shown below in Figure 41.

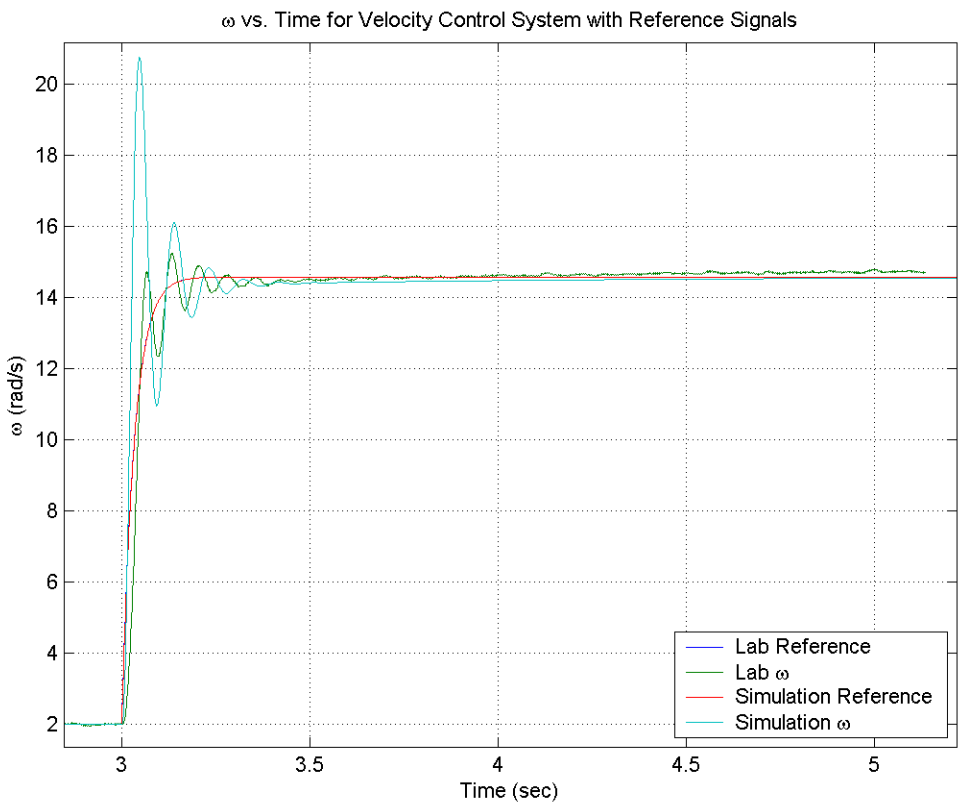


Figure 41: ω vs. time for step in velocity reference with $K_{vp} = K_{vi} = 1$

As can be seen, the model is fairly accurate in this situation. However, it is evident that the percent overshoot in the simulation is significantly higher than in the percent overshoot observed in lab. Zooming in on the area of maximum activity yields Figure 42.

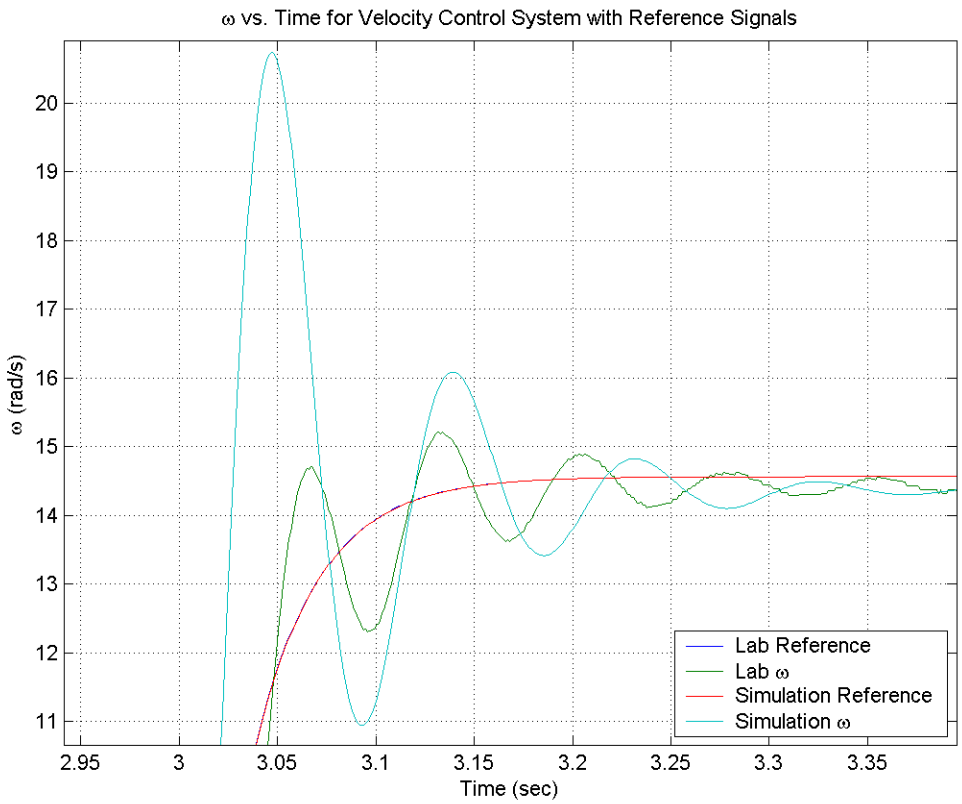


Figure 42: Closeup of overshoot ω vs. time for step in velocity reference with $K_{vp} = K_{vi} = 1$

As can be seen, the shape of the simulation output and the actual lab data are similar. The frequency of oscillations are even similar. However, it is observed that the percent overshoot in the simulation is significantly higher than the lab results. This could be caused for several reasons. One of the reasons is that the value of $J_m$ used is low. Recall that this was one of the parameters where the most approximations were made (see System Modeling section). If this value were too low, the motor would be able to spin up very fast which may lead to a higher percent overshoot.

Another interesting observation is to look at the steady state error. This is shown below in Figure 43.

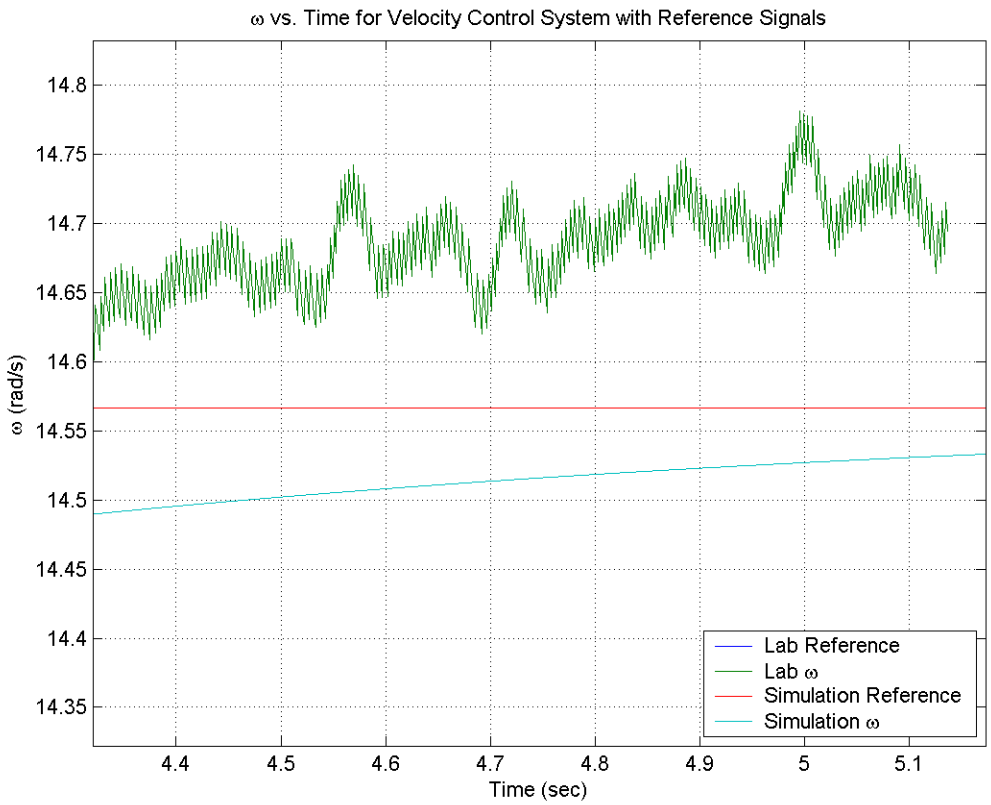


Figure 43: ω vs. time showing steady state error

An interesting observation to make here is to look at the simulated ω. In this situation, it appears that the integrator is doing its job and is reducing the steady state error to zero. However, by looking at the measured ω, it looks like the error is steady and possibly even increasing. Obviously, there is something wrong in this situation. The presence of the integrator should ensure that the steady state error goes to zero. It appears that there is a problem with the way that MatLab displays the error in the lab because when the oscilloscope was attached to the encoder, it was found that the wheel was actually on condition.

In addition to tracking a reference signal, the model is made to be robust in the face of a disturbance. In the lab, a disturbance in the form of the eddy current brake was applied.

**Position Control**

Now that the inner loop of the simulation (velocity control) is validated. A position control simulation can be implemented. This is effectively an outer loop that is wrapped around the inner velocity loop. The Simulink model used to control position in the lab is shown below in Figure 44.

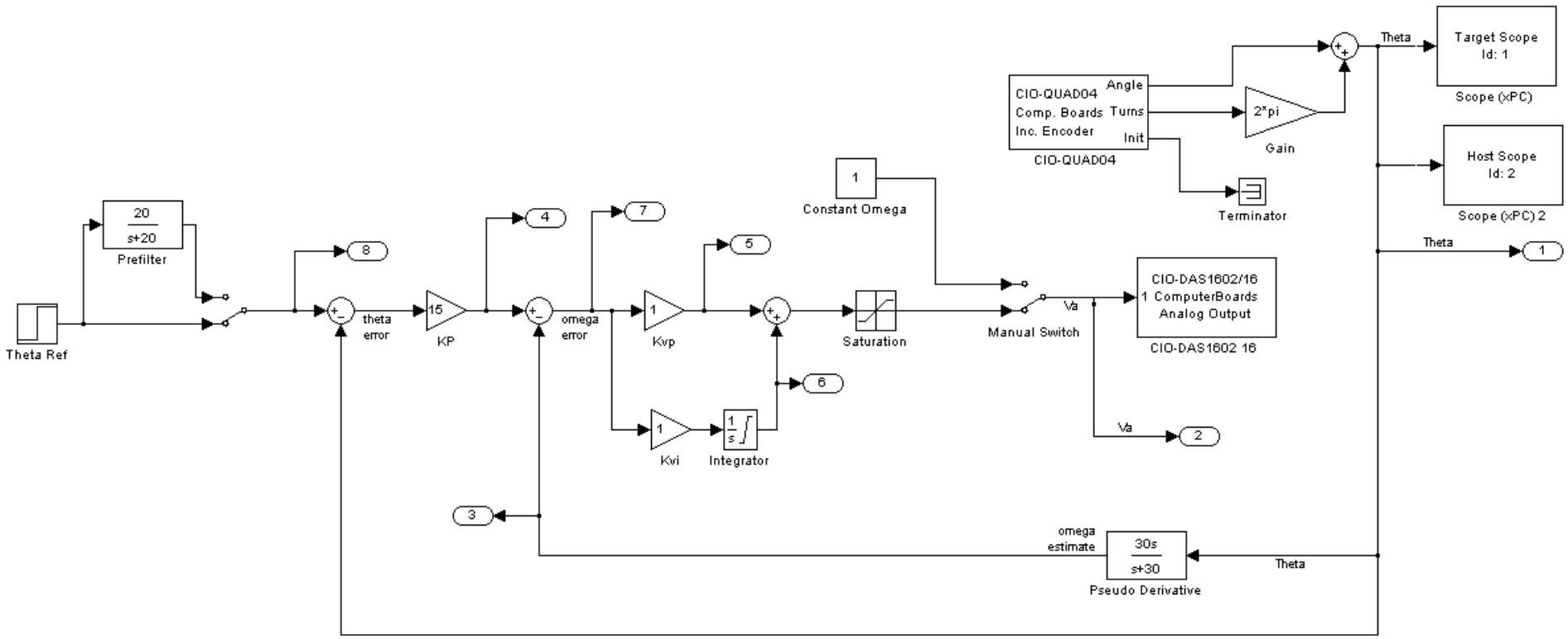


Figure 44: Simulink model used to interface with xpcTarget to track a angular displacement (θ) step of 180 degrees.

This model is used to send a command to the motor which will cause it to move 180 degrees. Initially, the step is commanded without a prefilter. The design goals for this section are less that 10 percent overshoot. Initially, the prefilter was not used (as shown above in Figure 44). Furthermore, it was discovered experimentally that $K_p = 1$ worked well and yielded no overshoot. Using these settings, data for a position command was acquired. In order to validate the accuracy of the model, the following Simulink model was constructed to simulate the position control system.

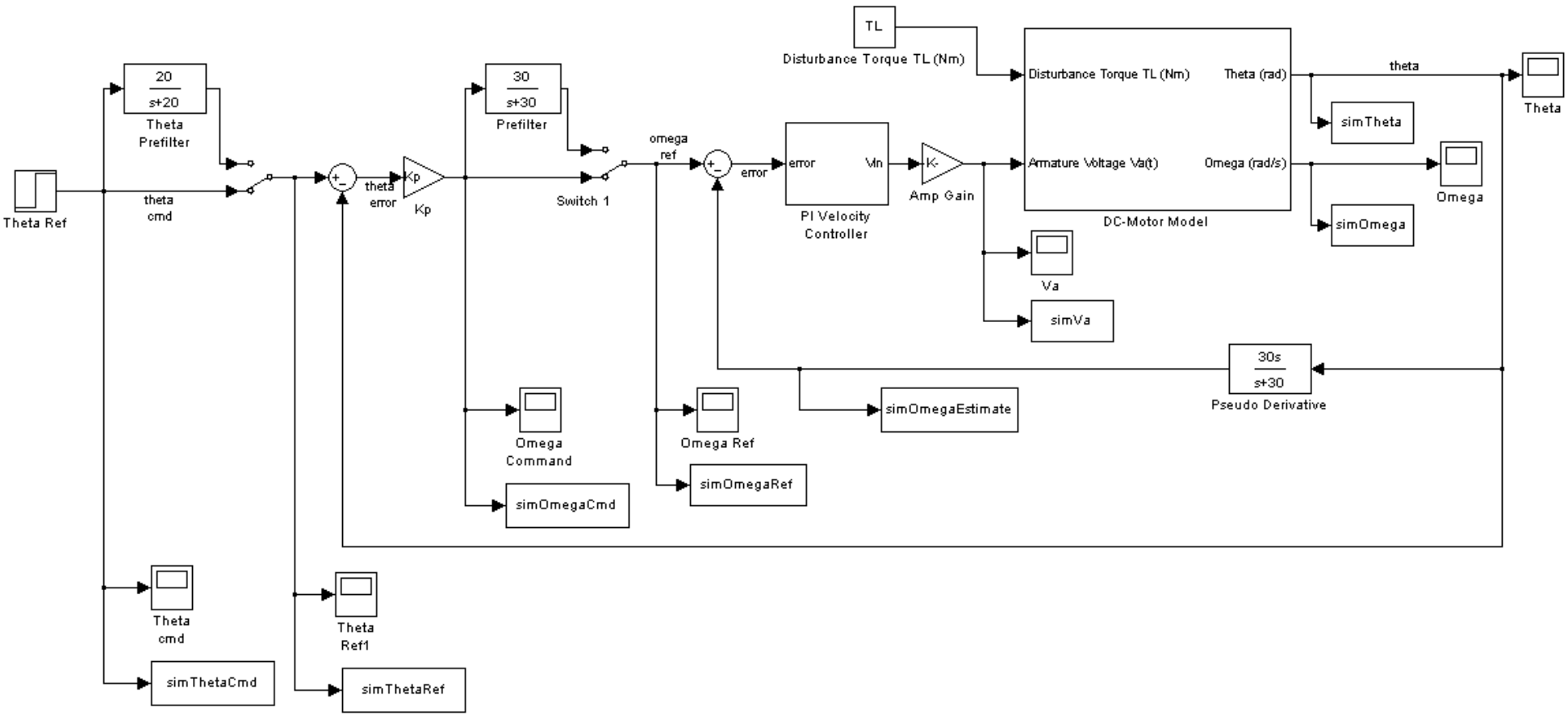


Figure 45: Simulink model used to simulate DC motor with angular displacement step of 180 degree

The simulated response of the system to a step input of θ = 180 degrees can now be compared with the data collected in lab. This is shown below in Figure 46.

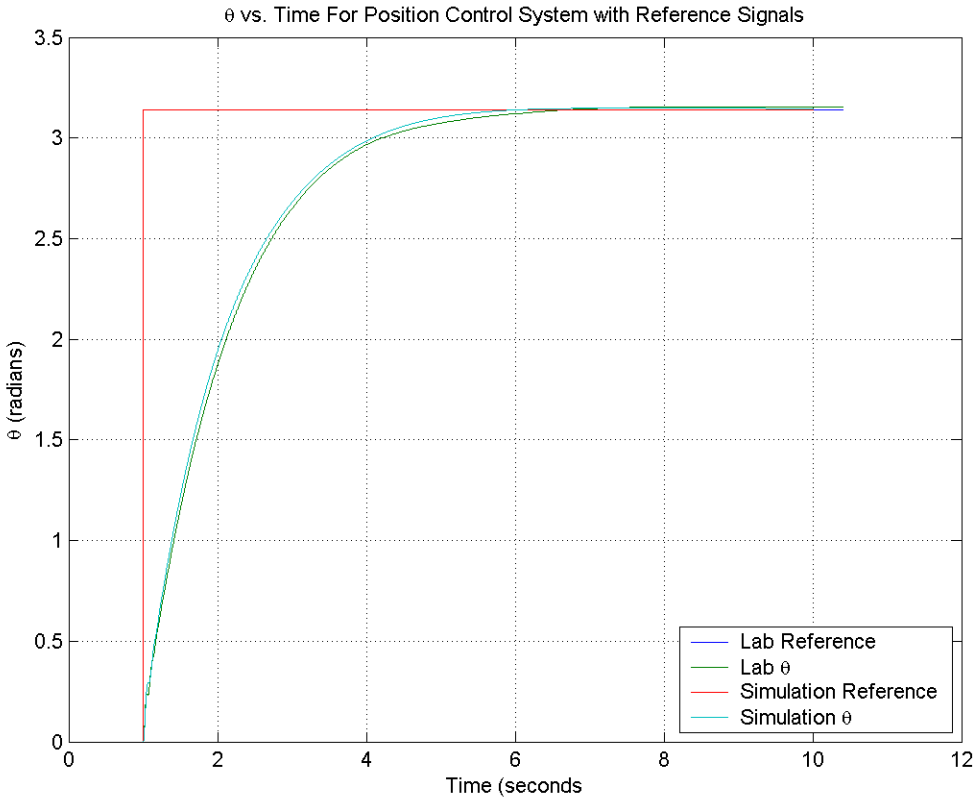


Figure 46: Simulated and actual θ vs. time for $K_p = 1$ and no prefilter

As can be seen from the above figure, the simulation is remarkable accurate as it matches the output of the lab extremely well. Both the shape and the magnitude as well as the steady state offset match the experiment. However, as shown above, the time to achieve this rotation is not ideal. In an attempt to decrease the settling time, $K_p$ is increased and a prefilter is added to lessen the percent overshoot. The following parameters are used to decrease the settling time

**$K_p = 7$**
**$\omega_{break} = 20$ rad/s (for θ prefilter)**

The actual response and the simulated response is shown below in Figure 47.

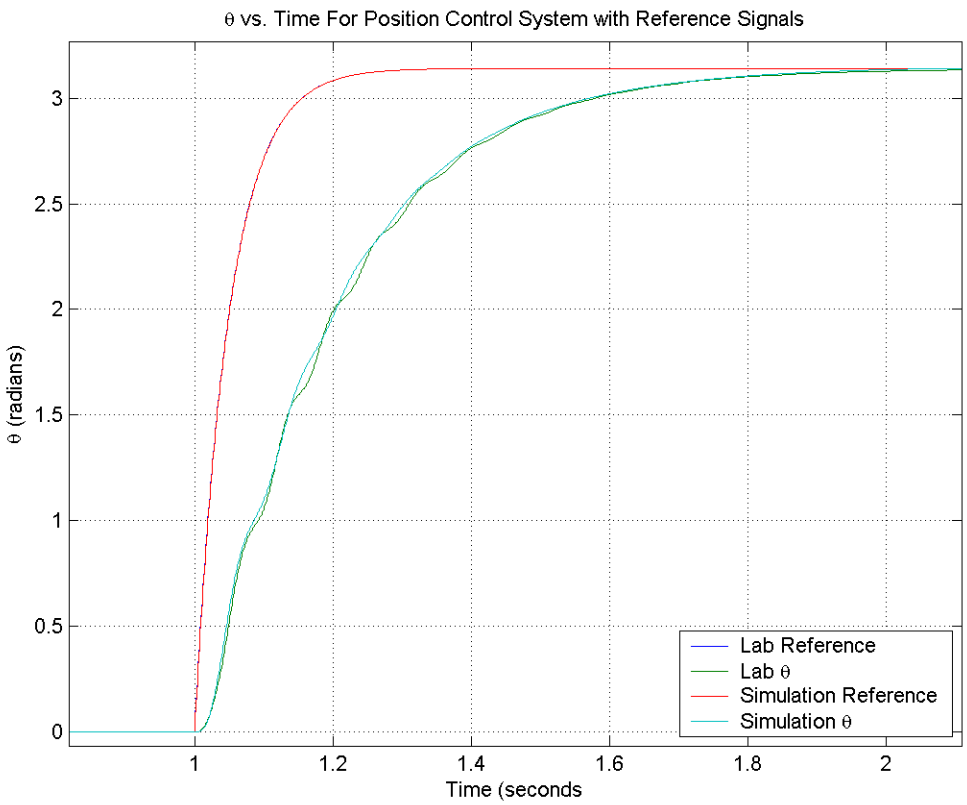


Figure 47: θ vs. time for position control system with reference signals

As can be seen, the increase in $K_p$ definitely decreases the settling time (previously 5.25 seconds to less than 1 second). Also, notice how the prefilter serves to smooth out the command from an infinitely sharp step to a more gentle reference signal. Also notice how closely the simulation fits the actual lab data. This shows that the model is indeed accurate and the parameters derived in the System Modeling section are very accurate.

This shows that a prefilter is very helpful. However, the prefilter does affect closed loop stability and gain and phase margin. Since the prefilter is essentially a pole at the origin. The bode diagram of the phase is simply a line that starts at zero then goes to -90 degrees as shown below.

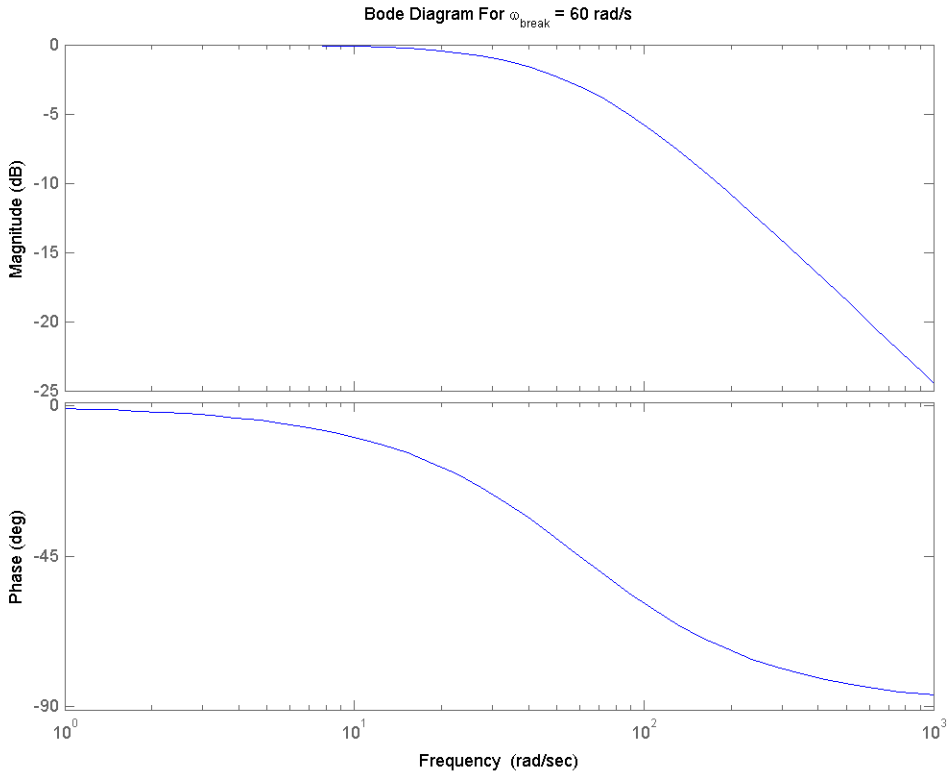


Figure 48: Bode plot of simple pole on real axis

As can be seen, not much phase lag is introduce at low frequencies. However, if the break frequency is lowered, the phase plot will begin to drop sooner, which essentially shifts the plot to the left. Therefore at the same frequency of operation, the system now has more phase lag. This cuts into the phase and gain margin. If the break frequency is lowered too far, the phase margin and gain margin could become negative, which causes instability. Therefore, the addition of a prefilter does in fact affect closed loop stability and gain and phase margin. The bandwidth of the of the prefilter should not be less than the response of the system. If the bandwidth is lower, then the system will actually be affected by the prefilter and information will be lost.

## VIII. Control Design Performance

Both the velocity and position control systems' performance is evaluated here. The first section of the velocity control design was to design a proportional feedback control gain which included a pseudo-derivative term with a frequency of 60Hz. The $K_{vp}$ best found to yield no overshoots (as stated in the lab) was a value of 0.22, and gave the results shown in Figure 49 below.

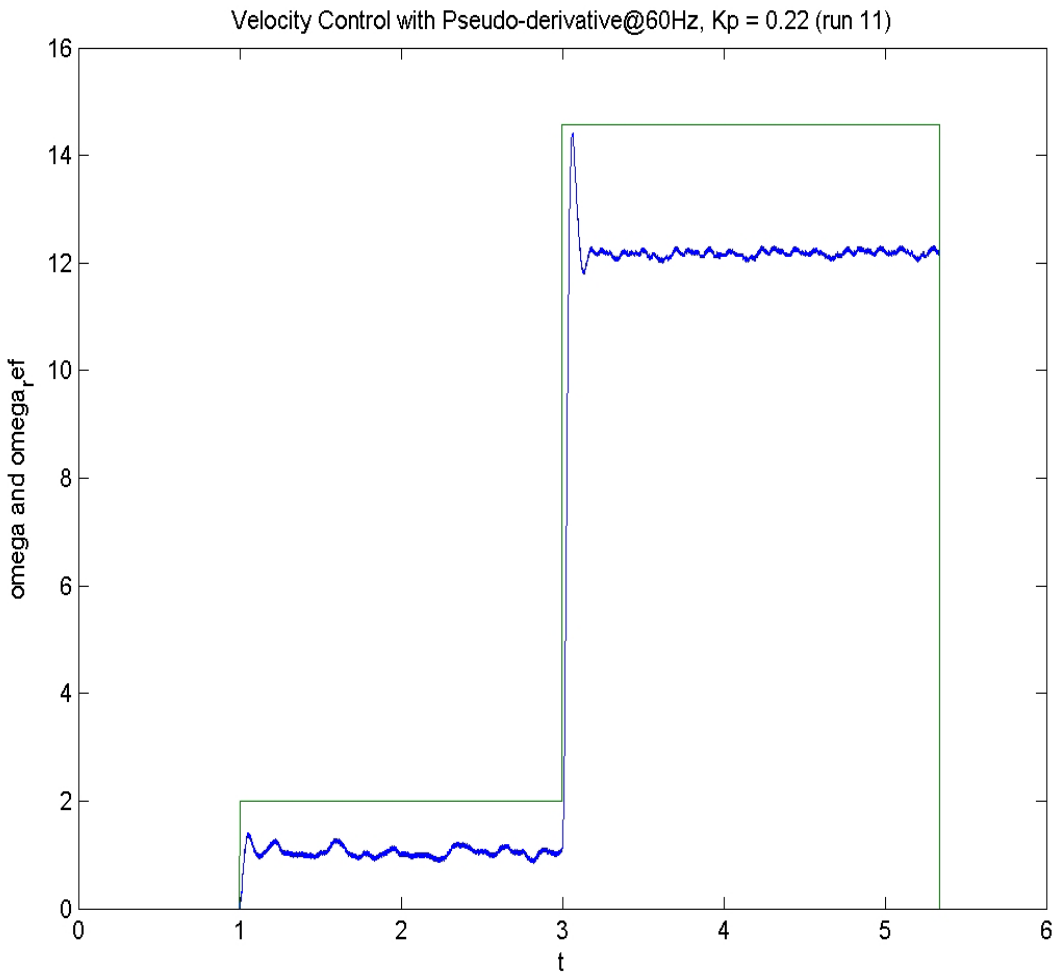


Figure 49: Velocity and reference velocity with P control.

Looking at the velocity data obtained it can be seen that using just a proportional gain does not give very good performance with respect to the reference step velocity. So, the next step was to add an integral gain and design a PI controlled position control system. The results for this are shown below in Figure 50. A proportional gain of $K_{vp} = 1$ and an integral gain of $K_{vi} = 1$ were used with a prefilter using a frequency of 30Hz and a pseudo-derivative term with a frequency of 30Hz to give the velocity data shown.

Using Matlab, code was developed to give performance data including percent overshoot, rise time, settling time, and steady state error. These calculations were all in reference to the set value given by the step change (either velocity or position). The steady state error was calculated by taking the mean of the value data in the area it showed settling behavior and then comparing that steady state mean value to the reference value. Each performance parameter is shown on the graphs below. The peak overshoot is marked by a red '*', the rise time is marked by a red '.', the settling time is marked by a red '+', and the mean steady state value is marked on the graph by a red line. The peak overshoot was related to the reference value to yield the percent overshoot. The rise and settling times were read from the time data stored in the data arrays for the appropriate points in the data. The steady state error again was computed by comparison of the mean steady state value to the reference value.

For the PI controlled velocity control system, a percent overshoot of 0.045% was calculated with a rise time of 3.06sec, a settling time of 3.254sec, and a steady state error of –0.04rad/sec. The graph shows how much better these results are than those found using only a P control gain (seen in the figure above).

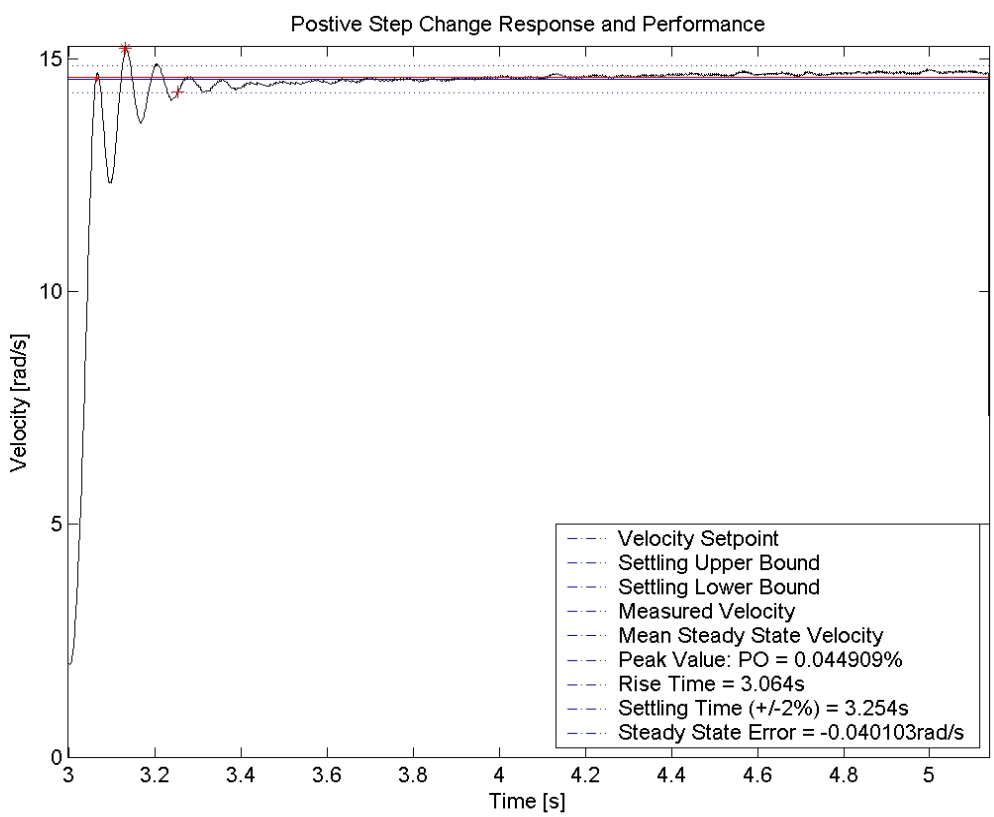


Figure 50: Performance results for velocity with PI control.

Particularly, here there is much lower steady state error. The PI control system models the reference velocity much better than the simply P control system first designed.

The next part of the lab was to design a position control system with just a proportional controller. This was first done without a prefilter, and then with a prefilter with a frequency of 20Hz. The results for the proportional controller without the prefilter are shown in Figure 51 below. During this portion of the position control system development, a KP of 5 was used.

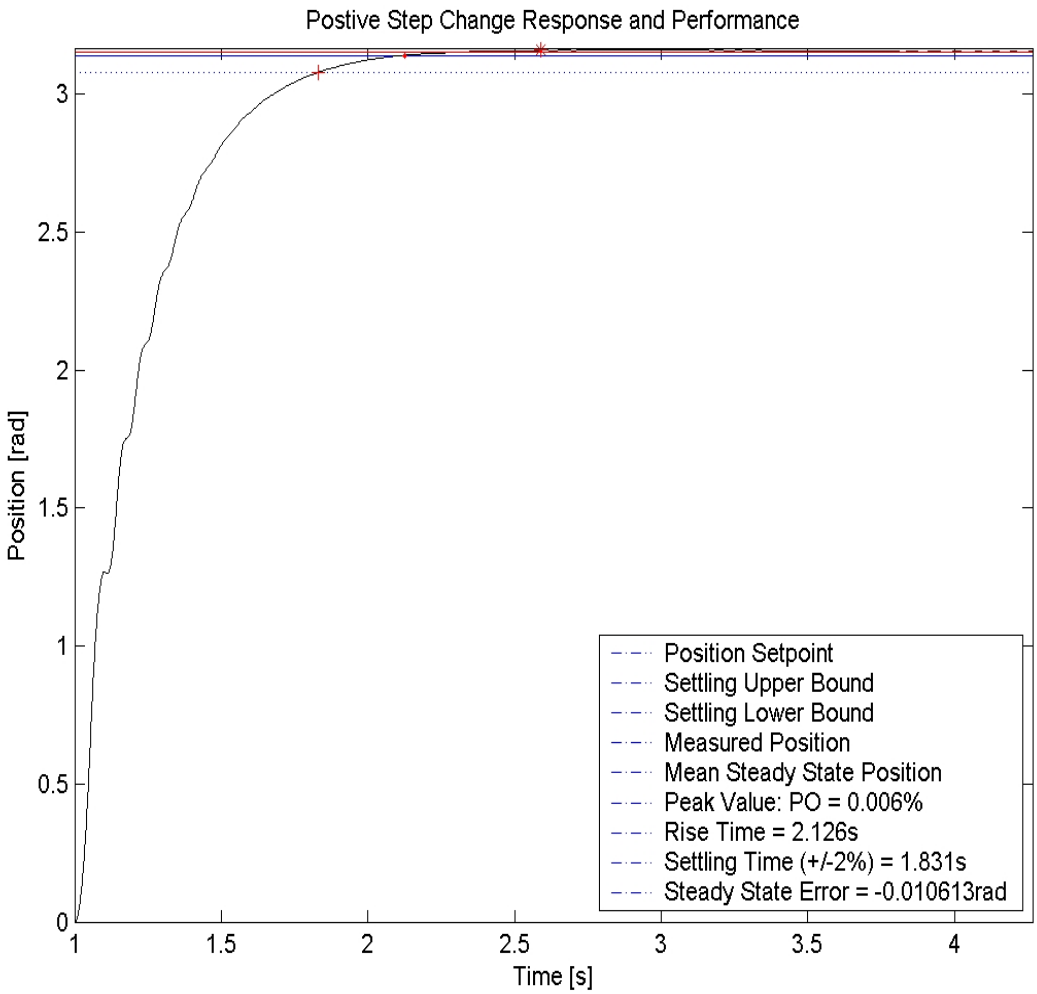


Figure 50: Performance results for position with P control.

Performance Matlab code applied to this data gave a percent overshoot of 0.006%, a rise time of 2.126sec, a settling time of 1.831sec, and a steady state error of –0.0106rad.

With a prefilter then added, the proportional gain was adjusted to KP = 7 with the prefilter frequency set at 20Hz. The results are shown in Figure 51 below.

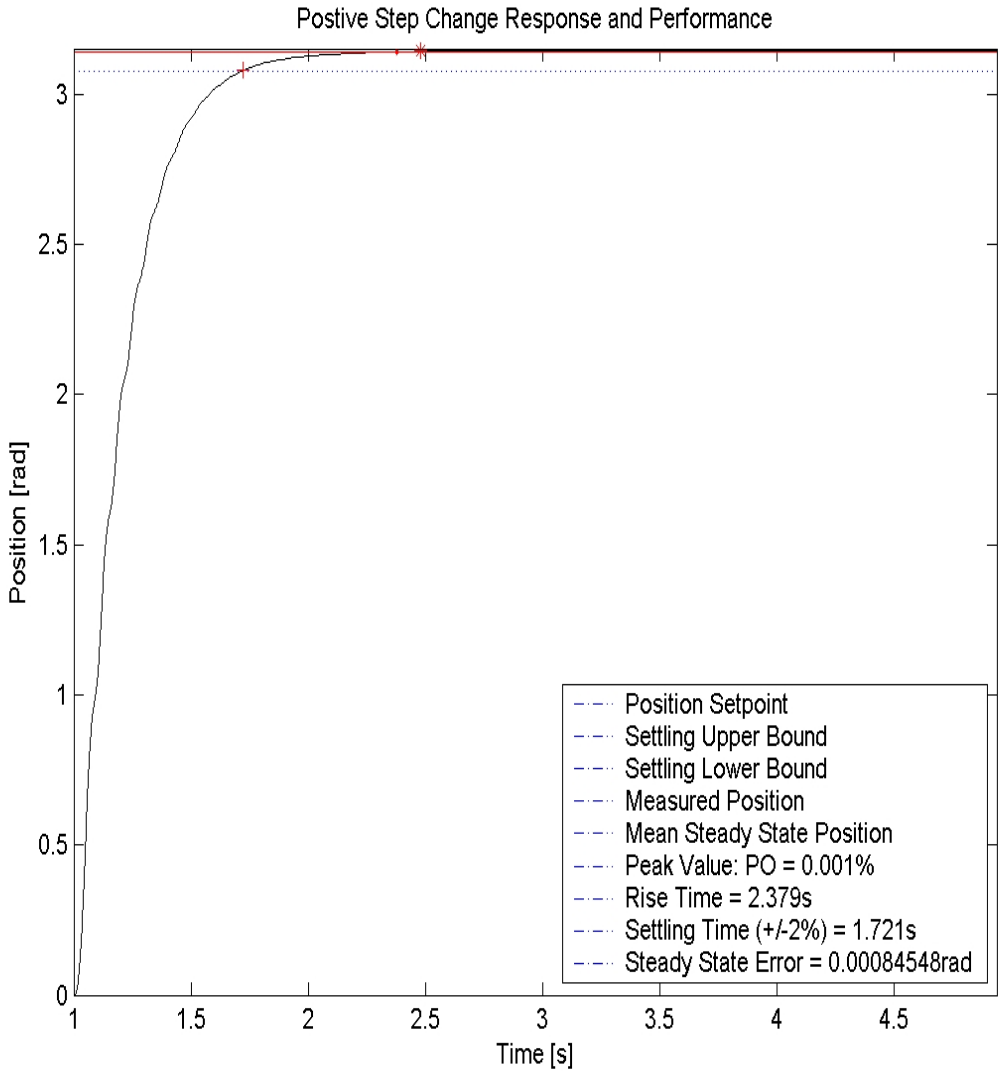


Figure 51: Performance results for position with prefilter.

The performance analysis run on these results gave a percent overshoot of 0.001%, a rise time of 2.379sec, a settling time of 1.721sec, and a steady state error of 0.0008rad. Comparison of the performance parameters for the proportional controller with and without a prefilter shows the controller using the prefilter as having better performance data. Specifically, there is a smaller percent overshoot, a faster settling time, and a smaller steady state error. The rise time was about 0.2sec slower, but the other performance parameters all are much better for the position controller using the KP of 7 with the prefilter.

### *Laboratory Discussion Items*

1. The Simulink models used to evaluate the velocity and position control system have already been constructed as shown in the Model Validation Section.

2. The open loop response of the Simulink model to a simple step command in velocity of $\omega_{ref} = 2$ rev/s has already been presented and analyzed in the Model Validation Section.

3. The step responses to a position command of $\pi$ radians in lab and simulation was already discussed in the Model Validation section. Likewise, the benefits of the prefilter were already covered.

4. The robustness of the control design can be quantified by measuring the gain and phase margin of each system. In order to do this, a bode plot must be generated of the system. This is difficult because the friction torque is nonlinear. Therefore, any linearized models (such as those obtained with LTI view, SISOtool, or linmod) will be approximations. Therefore, it becomes necessary to write custom MatLab code which will in effect, act like a digital signal analyzer.[2] This will input a sin wave of known frequency and amplitude and measure the output sin wave to obtain the attenuation and phase lag. The Simulink model used for obtaining the bode plot of the non-linear velocity control system is shown below in Figure 54.

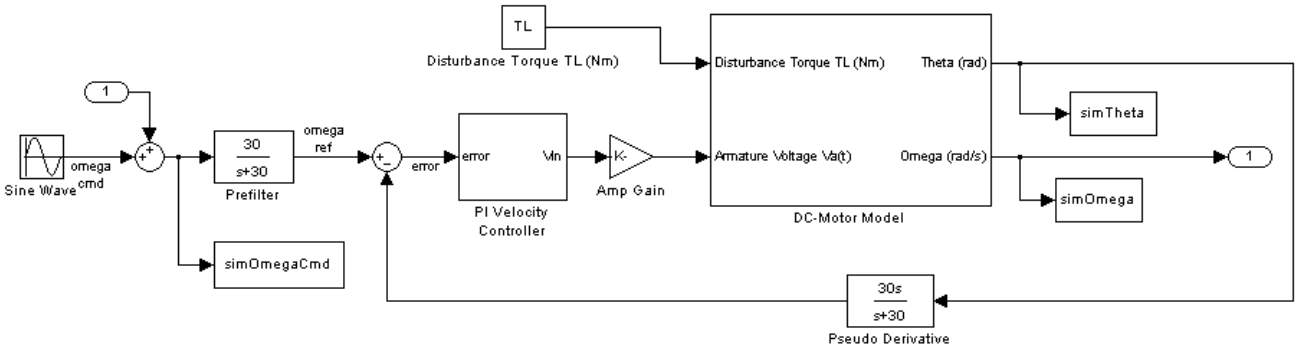


Figure 54: Simulink model used to simulate DSA and non-linear stiction torque

A known sin wave is then used as input and the output is measured. The code the identifies the maximum or minimum of the input and then finds the corresponding maximum or minimum of the output wave. The output is shown below in Figure 55

[2] See Appendix A for MatLab Code

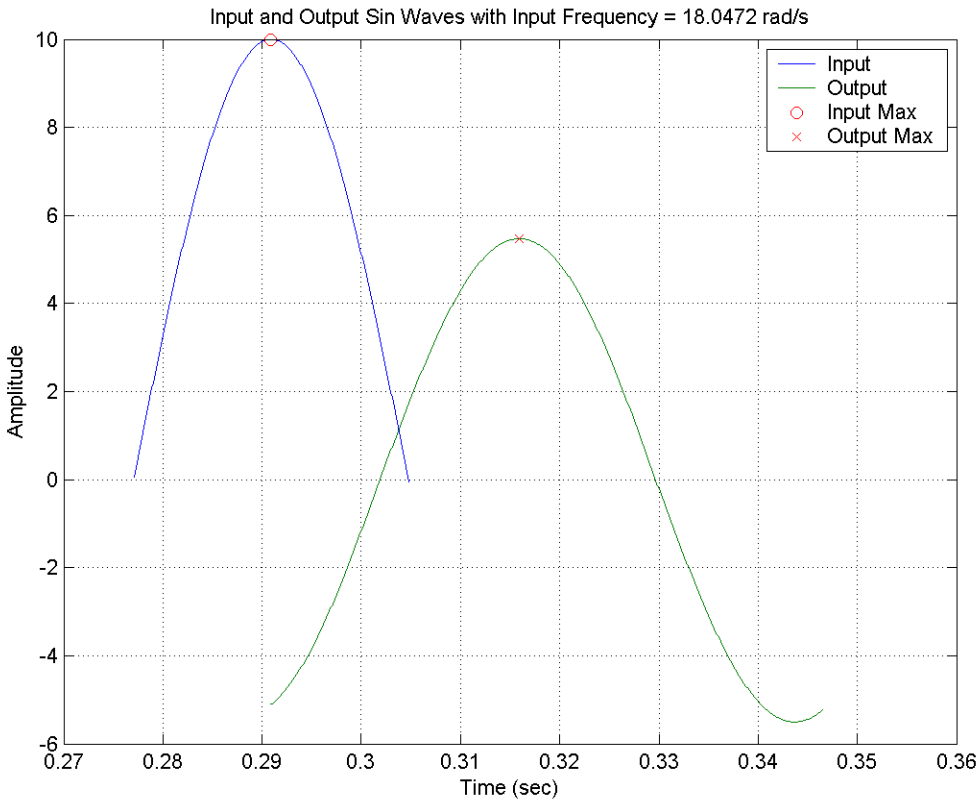


Figure 55: Output of computer code used to manually calculate bode plots

In Figure 55, it can be seen that there is a significant phase lag and that the output is slightly attenuated. As can be seen, the code correctly identifies the peaks of the input and the output. The input frequency is simply increased and the phase and amplitude are recorded. The resulting bode plot is shown below in Figure 56.

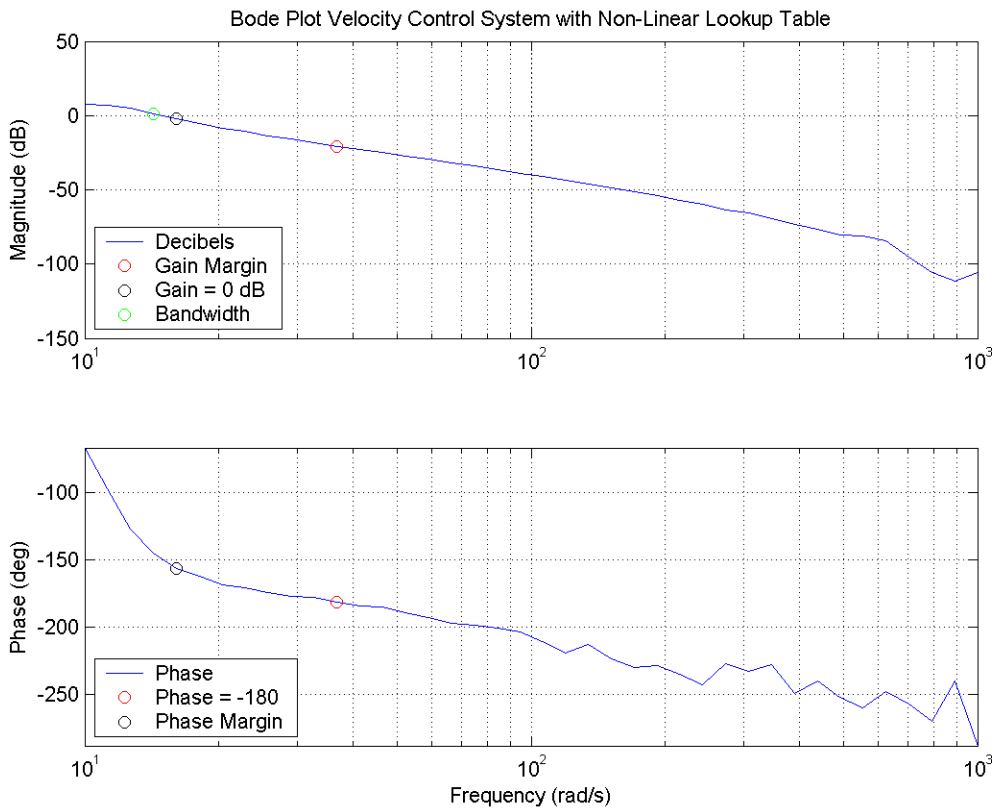


Figure 56: Bode plot which includes effects of nonlinear friction force

As can be seen above, the response is reasonable. The phase margin and gain margin can be calculated from the bode plot. Also, the bandwidth can be calculated. The actual values will be shown below.

In a similar fashion, the entire system can be linearized using the same model. Using the linmod function in MatLab, the linearized model can be obtained.

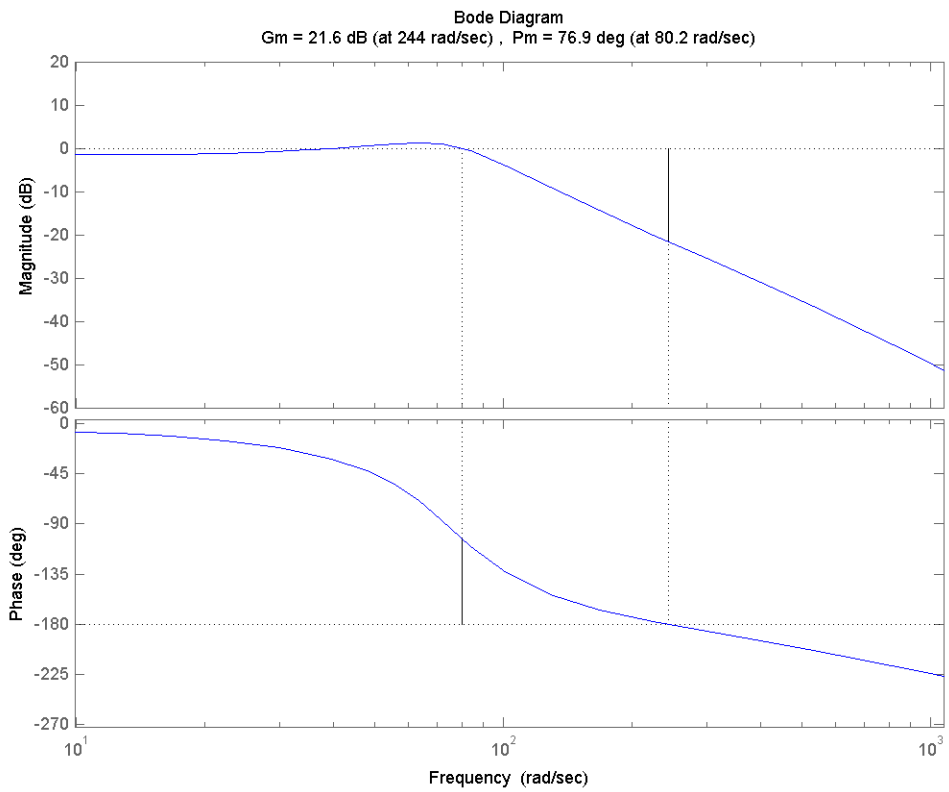


Figure 57: Bode plot of linearized velocity control system

As can be seen by comparing Figure 56 and Figure 57, both of them are fairly similar. The main difference between the two of them is that one takes into account the non-linear stiction torque and the other does not. However, the code that generated Figure 56 is still relatively untested. Therefore its accuracy may be in question. The critical parameters of the velocity control system are shown below in Table 9.

Table 9: Phase margin, gain margin, and bandwidth of velocity control system

| **System** | **Gain Margin (dB)** | **Phase Margin (degrees)** | **bandwidth (rad/s)** |
|---|---|---|---|
| Non-Linear | 20.6 | 23.6 | 14.3 |
| Linearized | 21.6 | 76.9 | 96.8 |

Both of the analysis yields stable models. It appears that the linearized model appear to be much more robust. Once again, this could be in error because in reality, the model is not linear. However, since the Non-linear code is not tested, it is more likely that the gain and phase margin and bandwidth obtained using linmod is more accurate.

In a similar fashion, the following Simulink model can be used to analyze the position control system.

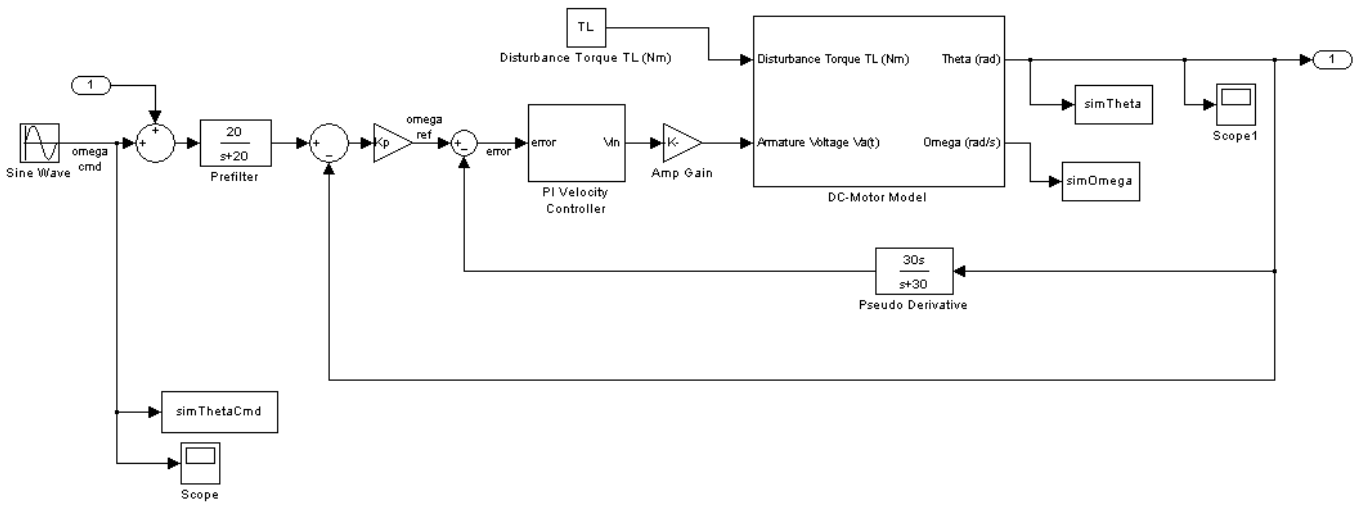


Figure 58: Simulink model used to simulate DSA and non-linear stiction torque

The bode diagram outputted by the code is shown below in Figure 59

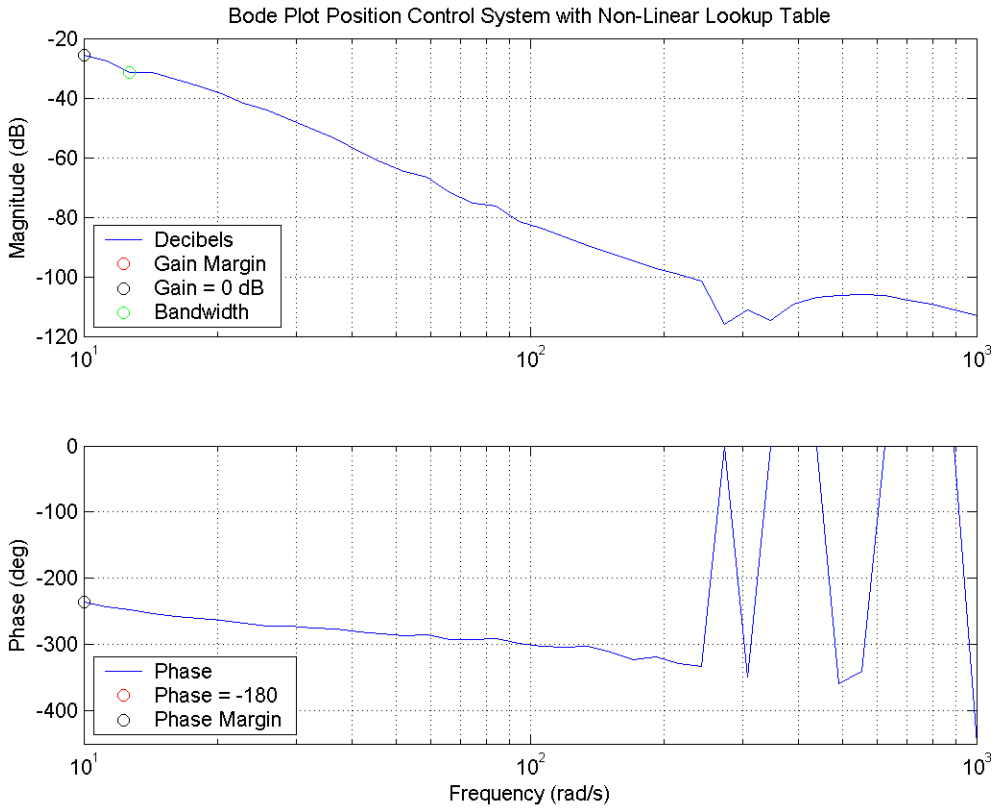


Figure 59: Bode plot which includes effects of nonlinear friction force

As can be seen from Figure 59 above, the code is unable to handle high frequency signals. The reason why is shown below in Figure 60.

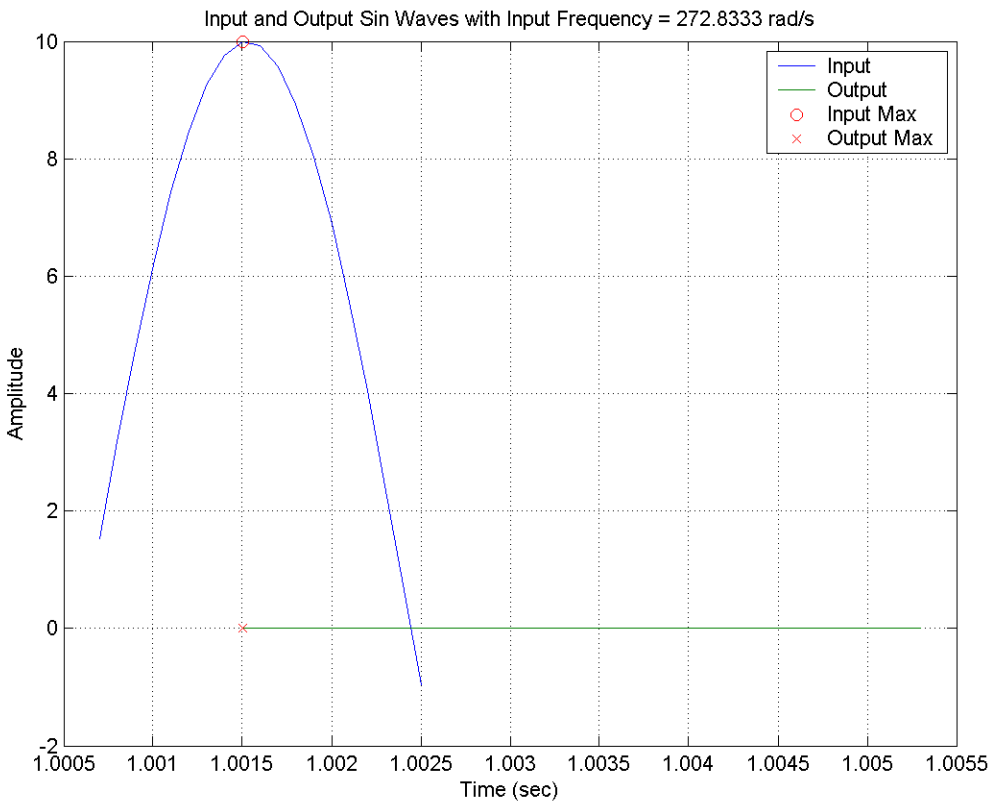


Figure 60: Output of computer code used to manually calculate bode plots showing miscalculation

The reason why high frequency oscillations do not result in accurate phase lag calculations is because the output is attenuated so much that it becomes a flat line. In this situation, it is impossible for the code to find the maximum peak and the max or min function returns the first point on the output curve. This results in a zero phase lag.

Since this solution is unreliable, the linmod function is used. The bode plot from the linearized model is shown below.

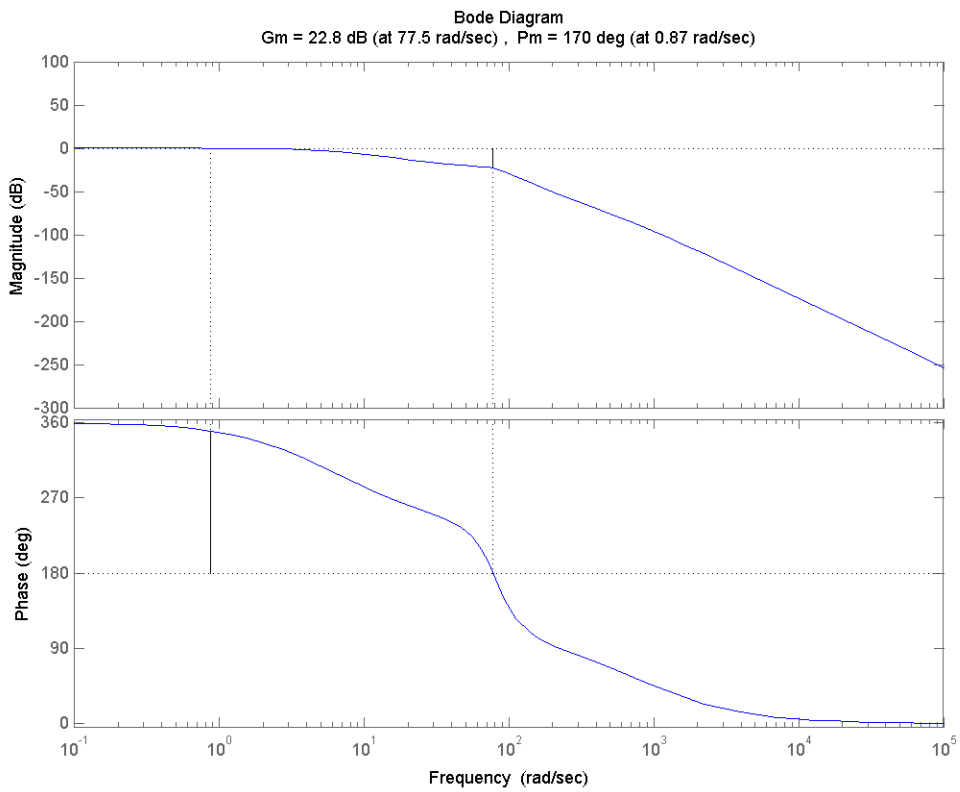


Figure 61: Bode plot of linearized position control system

The values from the linearized and non-linear model are shown below in Table 10.

Table 10: Phase margin, gain margin, and bandwidth of position control system

| System | Gain Margin (dB) | Phase Margin (degrees) | bandwidth (rad/s) |
|---|---|---|---|
| Non-Linear | 25.5 | -55.4 | 12.66 |
| Linearized | 22.8 | 170.0 | 5.09 |

As can be seen, the linearized model is similar to the non-linear model in terms of gain margin and bandwidth. However, the phase margin calculations of the custom MatLab code should not be trusted due to the phenomenon shown in Figure 60.

The other interesting thing to notice here is that the bandwidth decreased in both situations. This makes sense considering that the position control is the outer loop, which has inherently more phase lag than the inner loop (velocity control).

5. The gains $K_{vp}$, $K_{vi}$, and $K_p$ have already been derived using the root locus method have already been discussed in the Control System Design Section. These gains that were derived using simulation proved to be slightly higher than the actual gains used in lab. This is possibly due to unmodeled dynamics. The most significant set of unmodeled dynamics is non-linear friction torque. In order to implement the root locus method. The system needed to be linearized. This creates a loss of information and will affect the gain values.

6. The closed-loop command bandwidth of the velocity tracking has already been covered in Laboratory Discussion Question 4.

7. The closed-loop command bandwidth of the position tracking has already been covered in Laboratory Discussion Question 4.

8. The effect of A/D resolution can be significant for tracking applications. This is because the entire system is actually discrete. The encoder has a finite resolution. Obviously, if a count is missed, then the accuracy of the tracking system will decrease. The xpcTarget also has a finite A/D accuracy, or resolution. If this is less than the resolution of the encoder, then it is possible that information can be lost. This can lead to errors in the tracking signals.

## IX. Conclusions

In general, this was a very successful lab. There were many parameters to derive and many opportunities to go astray. However, all of the parameters were derived accurately and the model was determined to fit the data extremely well.

The exhaustive modeling of the system gave us insight and feel for the challenge of accurately modeling even a simple system. The presence of any non-linearities increases the difficulty exponentially.

Considering how common DC motors are, this lab allows for practical experience with these systems.

Given more time, further effort would have been put into the custom MatLab code which would simulate the DSA with a non-linear Simulink model. Currently, the code works for lower frequencies where the output signal is not attenuated much. However, as this frequencies increase, the robustness of the code decreases.

# X. Appendix

## *Appendix A: MatLab Code*

### lab2_DR.m

```
%Programmed by Christopher Lum
%9923916
%AA448 Ly
%Feb.4, 2003

%---------LAB 2 - MODELING AND DESIGN OF A
DC MOTOR---------------------
tic

clear
clc
close all

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:  Lock rotor test to
determine Rm--------------
disp('----------------------------
Determining Rm------------------------------
-')

%We need to first read in the data from the
tab deliniated file of the form.
%It is required that the data for each
Vsupply is in groups of 4
%
%     1st Column        2nd Column    3rd
Column      4th Column
%
%   Vsupply (volts)     Va (volts)      Vr
(mV)     Wheel Position
%
load A_5.txt

%Extract the supply voltage (Vsupply),
voltage across the motor (Va), and
%voltage across the measurement resistor
(Vr) and the position
Vsupply = A_5(:,1);
%Extract Vsupply (volts)
Va = A_5(:,2);
%Extract Va (volts)
Vr = A_5(:,3)/1000;
%Extract Vr (volts)
Position = A_5(:,4);
%Extract wheel position

%We can calculate the current in the loop
since we know the resistant of
%the measurement resistor.
R = 0.05;
%Ohms

Current = Vr/R;
%Amps

%The voltage across the motor varies with
the rotor position.  Therefore,
%we should take an average of the readings.
We know that there are 4
%datapoints for each supply voltage. We want
the code to start looking at
%the data at indices of 1-4, 5-8, 9-11, 12-
15, etc
window = 1;

%How many different "blocks" are there in
the data set?
NumBlocks = length(Vsupply)/4;

for counter = 1:NumBlocks
    %Extract the values of Va and current
only in the window 1-4, 5-8, etc.
    VaSet = Va(window:(window+3));
    CurrentSet = Current(window:(window+3));

    %calculate the average
    VaAve(counter) =
sum(VaSet)/length(VaSet);
    CurrentAve(counter) =
sum(CurrentSet)/length(CurrentSet);

    %increment the window by 4
    window = window + 4;
end

%Va and current (i) should be related by
%
%   Va = (Rm+R)*i
%
%Therefore we should try a linear fit
through the data.
coefficients = polyfit(CurrentAve,VaAve,1);

%The slope should be (Rm+R)
Rm = coefficients(1)-R;
offset = coefficients(2);

sprintf('Rm = %3.3f ohms (resistance of
measurement resistor)',Rm)

if graphselection==1
    figure

plot(CurrentAve,VaAve,'rx',CurrentAve,((Curr
entAve*Rm)+offset))
    title('Voltage Across  Motor (V_a) vs.
Current Through Loop (i).  Slope should be
R_m+R.')
    xlabel('Current i (amps)')
    ylabel('V_a (volts)')
    legend('Averaged Data Points','Linear
Fit',2)
    grid
else
end


%----------------Part B:  Lock rotor test to
determine La--------------
disp('----------------------------
Determining La------------------------------
-')
%We need to first load the data of the
dynamic response of voltage vs. time
%with the rotor locked.
%
%  1st Column     2nd Column
%
%   time (sec)     Vr (mV)
%
%the first try was garbage, so we will use
the second and third attempts
load B_1_4_second_try.txt
load B_1_4_third_try.txt
load B_1_4_fifth_try.txt
load B_1_4_sixth_try.txt

%Extract the time (recorded in sec) and
voltage across the measurement
%resistor (measured in mV)
timeB2 = B_1_4_second_try(:,1);
%Extract time (sec)
VrB2 = B_1_4_second_try(:,2)/1000;
%Extract Vr (volts)

timeB3 = B_1_4_third_try(:,1);
%Extract time (sec)
VrB3 = B_1_4_third_try(:,2);
%Extract Vr (volts)

timeB5 = B_1_4_fifth_try(:,1);
%Extract time (sec)
VrB5 = B_1_4_fifth_try(:,2);
%Extract Vr (volts)

timeB6 = B_1_4_sixth_try(:,1);
%Extract time (sec)
VrB6 = B_1_4_sixth_try(:,2);
%Extract Vr (volts)

%Calculate the current
CurrentB2 = VrB2/R;
%amps
CurrentB3 = VrB3/R;
%amps
CurrentB5 = VrB5/R;
%amps
CurrentB6 = VrB6/R;
%amps

%Its appears that the fifth try has good
results.  We can find delta_i_ss
%(change in current due to step) easily
delta_i_ss = max(CurrentB5)-min(CurrentB5);

%We can now find out when delta_i_ss*(1-e^-
1) occurs
critical_value_i = delta_i_ss*(1-exp(-1)) +
min(CurrentB5);
```

```
for counter=1:length(CurrentB5)
    if CurrentB5(counter)>critical_value_i
        break
    else
    end
end

critical_time_i = timeB5(counter);

%The rise time is the difference between
this value and the time when the
%signal started rising.  From inspection of
the graph, the current starts
%rising around t = -0.5e-5 seconds
start_rise_time = (-0.5*10^-5);
rise_time = critical_time_i -
start_rise_time;

%We can now solve for La
La_calculated = rise_time*(Rm+R);

sprintf('La = %3.5f henries (inductance of
motor derived from data collected on 5th try
(see figure))',La_calculated)

if graphselection==1
    figure

plot(timeB5,CurrentB5,critical_time_i,Curren
tB5(counter),'rx',[critical_time_i
critical_time_i],[min(CurrentB5)
critical_value_i],'k-',[min(timeB5)
critical_time_i],[critical_value_i
critical_value_i],'k-')
    title('Current vs. Time for Locked Rotor
Test Using 1 Average on 5th Try')
    xlabel('time (seconds)')
    ylabel('Current (amps)')
    legend('Actual Data','Critical Point',2)
    grid
else
end

%We can also calculate the first order
response using this value of La to
%see how well it fits the data.  In order to
do this, we need to create a
%time array that only starts when the
current begins to rise.
%From inspection of the graph, the current
starts
%rising around t = -0.5e-5 seconds.  We need
to cut of any time values
%before this.
bad_indices = find(timeB5<start_rise_time);

%find the last bad index
last_bad_index = max(bad_indices);

%now create a time array that only has value
of time where the current is
%rising
modified_time =
timeB5(last_bad_index:length(timeB5));

uo = 3.448;                          %volts
partA=delta_i_ss;
partB=(((Rm+R)/La_calculated)*modified_time)
;
fitted_current = partA*(1-exp(-partB));

if graphselection==1
    figure

plot(timeB5,CurrentB5,modified_time,fitted_c
urrent,'r-')
    title('Current vs. Time for Locked Rotor
Test and 1st Order Response Fit')
    xlabel('time (seconds)')
    ylabel('Current (amps)')
    legend('Actual Data','Fitted 1st Order
Response',4)
    grid
else
end


%We obtain a variety of rise times.  We will
average them together until
%a better method is introduced
differentTr = [1.2 1.108 1.13 .998 1.004];

Tr =
sum(differentTr)/length(differentTr)/1000;
%seconds

%We can now calculate La since the response
is assumed to be first order
La = Tr*(Rm+R);

sprintf('La = %3.5f henries (inductance of
motor derived by averaging different rise
times measured off oscilloscope)',La)

%----------Part C: Varying load test to
determine machine constant----------
disp('---------------------------
Determining Kv = Kt------------------------
----')
%------------------------------C.3---------
---------------------------------
%We can load the data saved from the encoder
for the counter-clockwise
%rotation.  The file is of the form
%
%     1st Column        2nd Column      3rd
Column
%
%     time (sec)        Ch. 1 (volts)
Ch.2
%
load C_3_CCW.txt

%Extract time (sec), and voltage from the
encoder channels (volts)
timeC_CCW = C_3_CCW(:,1);
%Extract time (sec)
Ch1C_CCW = C_3_CCW(:,2);
%Extract channel 1 voltage (volts)
Ch2C_CCW = C_3_CCW(:,3);
%Extract channel 2 voltage (volts)

%Plot them to observe the phase relationship

if graphselection==1
    figure

plot(timeC_CCW,Ch1C_CCW,timeC_CCW,Ch2C_CCW)
    title('Output voltage from Encoders vs.
Time For Counter-Clockwise Rotation')
    xlabel('Time (sec)')
    ylabel('Output voltage (volts)')
    legend('Channel 1','Channel 2')
    axis([min(timeC_CCW) max(timeC_CCW)
min(Ch1C_CCW)-1 max(Ch1C_CCW)+1])
else
end



%We can perform a similar operation for when
the polarity was reversed and
%the wheel was rotating clockwise.  The file
is of the form
%
%     1st Column        2nd Column      3rd
Column
%
%     time (sec)        Ch. 1 (volts)
Ch.2
%
load C_3_CW.txt

%Extract time (sec), and voltage from the
encoder channels (volts)
timeC_CW = C_3_CW(:,1);
%Extract time (sec)
Ch1C_CW = C_3_CW(:,2);
%Extract channel 1 voltage (volts)
Ch2C_CW = C_3_CW(:,3);
%Extract channel 2 voltage (volts)

%Plot them to observe the phase relationship
if graphselection==1
    figure
    plot(timeC_CW,Ch1C_CW,timeC_CW,Ch2C_CW)
    title('Output voltage from Encoders vs.
Time For Clockwise Rotation')
    xlabel('Time (sec)')
    ylabel('Output voltage (volts)')
    legend('Channel 1','Channel 2')
    axis([min(timeC_CW) max(timeC_CW)
min(Ch1C_CW)-1 max(Ch1C_CW)+1])
else
end


%------------------------------C.7---------
---------------------------------
%We now can now vary the torque load on the
motor by changing the location
%of the eddy current brake.  The file is of
the form
%
%     1st Column        2nd Column      3rd
Column          4th Column
%
%     Va (volts)        Vr (mV)
frequency (kHz)     Brake Position (%)
%
%The brake position denotes where it is
located.  At 100% the brake is
%fully in and and 0%, it is fully out.
load C_9.txt

%Extract the data
VaC = C_9(:,1);
%Extract Va (volts)
VrC = C_9(:,2)/1000;
%Extract Vr (volts)
```

```
frequencyC = C_9(:,3)*1000;
%Extract frequency (Hz) positive values
denotes positive Va
BrakePositionC = C_9(:,4);
%Extract brake position (%)

%We can calculate the current through the
loop at each speed easily by
%dividing Vr by R
currentC = VrC/R;

%We need to convert the frequncy to radians
per second.  Since we have a
%thousand line encoder, we divide the
frequency (pulses per second) by
%1000 pulses per revolution to obtain
revolutions per second.  We then
%simply multiply by 2pi to convert to
rads/seconds.
omegaC = (frequencyC/1000)*2*pi;
%rad/s

%We can now calculate Kv
KvC = (VaC - ((Rm+R)*currentC))./omegaC;

%lets find the average
Kv = (sum(KvC)/length(KvC));

sprintf('Kv = %1.5f volt seconds (machine
constant derived by solving equation and
taking average)',Kv)


%We can also derive this graphically.  The
electrical equation can be
%rewritten as
%
%   Va - ((Rm + R)*i(t)) = Kv*omeage(t)
%
%Therefore, we can plot the RHS vs. the
omega and the slope should be Kv.
%We can then apply a linear fit whose slope
should be Kv
coefficientsC = polyfit(omegaC,VaC -
((Rm+R)*currentC),1);

KvGraphical = coefficientsC(1);
offsetGraphical = coefficientsC(2);

%Kt is simply equal to Kv since we are using
consisten units
Kt = Kv;

sprintf('Kv = %1.5f volt seconds (machine
constant derived graphically)',KvGraphical)

if graphselection==1
    figure
    plot(omegaC,VaC -
((Rm+R)*currentC),'rx',omegaC,(omegaC*KvGrap
hical)+offsetGraphical)
    title('Plot showing V_a - (R_m+R)i(t)
vs. \omega(t).  Slope Should be K_V')
    xlabel('\omega (rad/sec)')
    legend('Data Points','Linear Fit',2)
    grid
else
end

sprintf('Kv derived by solving eqaution and
taking average will be used for further
calculations')

%----Part D: Varying input voltage to
determine motor torque function bm---
disp('----------------------------
Determining bm-------------------------------
-')
%We can now vary the input voltage in order
to see the effects of friction
%and sticktion.  The file is of the form and
is required to be sorted from
%negative at beginning and positive values
at end
%
%     1st Column        2nd Column      3rd
Column
%
%     Va (volts)        Vr (mV)
frequency (kHz)
load D_4.txt

%Extract the data
VaD = D_4(:,1);
%Extract Va (volts)
VrD = D_4(:,2)/1000;
%Extract Vr (volts)
frequencyD = D_4(:,3)*1000;
%Extract frequency (Hz) positive values
denotes positive Va

%calculate omega in rad/sec
omegaD = (frequencyD/1000)*2*pi;
%rad/s

%calculate the current
currentD = VrD/R;
%amps

%We know that the mechanical equation at
steady state is given by
%
%   T(t) = bm*omega(t)          (Eq.D)
%
%We also know that the torque produced is
the product of the Kv and the
%current
%
%   T(t) = Kv*i(t)
%
TorqueD = Kv*currentD;

%We know that if we plot Eq.D, the graph
will be nonlinear.  We instead
%want to approximate this as two linear
functions.  We would like a linear
%function for when the rotation is positive
and for when the rotation is
%negative.  From the graph, it appears that
we should try and fit the datapoints
%if the angular frequency is greater than 20
rad/sec and less than -20 rad/sec.
%We'll create two sets of datapoints
positiveCutoffRate = 20;
%rad/s
negativeCutoffRate = -20;
%rad/s

sprintf('Only using data where omega is
greater than %2.2f rad/s and less than %2.2f
rad/s',positiveCutoffRate,negativeCutoffRate
)

for positiveOmegaCounter=1:length(omegaD)
    %We know that the datafile starts from
negative to positive, so the first value
    %it comes across should be the first
time omega is greater than the cutoff rate
    if omegaD(positiveOmegaCounter) >
positiveCutoffRate
        break
    else
    end
end

%We can now create an array of on the values
of torque and omega that are greater
%than the cutoff rate
positiveOmegaD =
omegaD(positiveOmegaCounter:length(omegaD));
positiveTorqueD =
TorqueD(positiveOmegaCounter:length(TorqueD)
);

%Likewise, we can do this for the negative
side.  Since the data is arranged form
%negative to positive, we can write
for negativeOmegaCounter=1:length(omegaD)
    if omegaD(negativeOmegaCounter) >
negativeCutoffRate
        break
    else
    end
end

negativeOmegaD =
omegaD(1:negativeOmegaCounter-1);
negativeTorqueD =
TorqueD(1:negativeOmegaCounter-1);

%We can now find a 1st order polyfit for the
data
coefficientsPositive =
polyfit(positiveOmegaD,positiveTorqueD,1);
coefficientsNegative =
polyfit(negativeOmegaD,negativeTorqueD,1);

%The slope should be the value of alpha
since in the linear region
%
%   bm = alpha*omega
alphaPositive = coefficientsPositive(1);
alphaNegative = coefficientsNegative(1);

sprintf('When motor is spinning in positive
direction (CCW)\n\n  alpha = %1.9f\n\nWhen
motor is spinning in negative direction
(CW)\n\n  alpha =
%1.9f',alphaPositive,alphaNegative)

offsetPositive = coefficientsPositive(2);
offsetNegative = coefficientsNegative(2);

%Plot the linear fit line along with the
data assuming that this value
%of bm is linear from zero.  This requires
that we create an array with values
%of the linear fit starting from zero.  This
requires concatenating a value of 0 in
%the array
omegaDFitPositive = [0;positiveOmegaD];
omegaDFitNegative = [negativeOmegaD;0];

TorqueDFitPositive =
(omegaDFitPositive*alphaPositive)+offsetPosi
tive;
```

```
TorqueDFitNegative =
(omegaDFitNegative*alphaNegative)+offsetNega
tive;

%Since it is difficult to implement this in
Simlink, we will use a lookup
%table function.  We need to know the
coordinates of the points to use as
%look up points.  Keep in mind that at zero
velocity, the friction torque
%should be zero.  We would like to add some
datapoints near zero also.
negative_near_zero = -0.5;            %rad/s
positive_near_zero = 0.5;             %rad/s

xnegative =
[omegaDFitNegative(1:length(omegaDFitNegativ
e)-1);negative_near_zero];
xpositive =
[positive_near_zero;omegaDFitPositive(2:leng
th(omegaDFitPositive))];

omegaLookUpTable = [xnegative;0;xpositive];

ynegative =
(xnegative*alphaNegative)+offsetNegative;
ypositive =
(xpositive*alphaPositive)+offsetPositive;

TorqueLookUpTable = [ynegative;0;ypositive];

%plot the data and the linear fit, and the
lookup table output.
if graphselection==1
    figure

plot(omegaD,TorqueD,'rx',omegaLookUpTable,To
rqueLookUpTable)
    title('Torque Due to Friction vs.
\omega')
    xlabel('\omega (rad/s)')
    ylabel('Torque (Nm)')
    legend('Data Points','Look Up Table
Values',2)
    grid
else
end

%We also know when the motor stopped
rotating.  Therefore we can calculate the
%dead zone
negativeDeadZone = -13.2/1000/Rm;
positiveDeadZone = 14.36/1000/Rm;
```

```
%--------------------------------WEEK 2-----
-------------------------------------
disp('---------------------------
Determining Jm------------------------------
-')
%
%--------------Part A:  Determination of
Motor Drive Inertia Jm-----------------
%
%
%We would like to find Jm.  In order to do
this, we need to load in the data taken
%from the XpcTarget machine.  This outputs
to the workspace under a variable named
%tg.

%load the acquired data.  This loads the
following three variables
%
%   -time
%   -theta
%   -pulseVoltage
%
load JmData

%however, since positive theta is defined as
CCW and the wheel rotated CW,
%then a negative sign must be applied the
the angular position
theta = theta*-1;

if graphselection==1
    figure
    plot(time,theta,time,(3.5*pulseVoltage-
3.4))
    title('\theta and V_i_n (voltage
supplied to amplifier) vs. Time')
    xlabel('time (seconds)')
    ylabel('\theta (radians)')
    legend('\theta','V_i_n')
    grid
    axis([0.95 1.5 -3.5 0.25])
else
end
```

```
%Lets find out when the pulse occurs.
initialVoltage = pulseVoltage(1);

for counter=1:length(pulseVoltage)
    currentPulseVoltage =
pulseVoltage(counter);

    if currentPulseVoltage > initialVoltage
        break
    else
    end
end

startIndex = counter;

%We would only like to find out when the
pulse ends
for counter=startIndex:length(pulseVoltage)
    currentPulseVoltage =
pulseVoltage(counter);
    if currentPulseVoltage <
pulseVoltage(startIndex);
        break
    else
    end
end

endIndex = counter;

%We could try and fit the entire curve of
theta vs. time but it may be more
%accurate to only look at the beginning
section since this is truly when
%angular velocity is small and the friction
can be ignored.
criticalPercent = 1;                  %how
much of pulse should be analyzed? ie 0.6 =
60% of pulse should be analyzed.

%lets calculate how long the pulse was
applied for
pulseTime = time(endIndex)-time(startIndex);

sprintf('The pulse was present for %1.4f
seconds.  However, only %2.2f percent of the
pulse was analyzed and
fitted.',pulseTime,criticalPercent*100)

%lets find the ending index that corresponds
to the critical percentage of
%time when we are analyzing
CriticalEndIndex = round((endIndex-
startIndex)*criticalPercent)+startIndex;

%We can now create an array of variables
that only look at the critical time when
%the pulse was occuring and set this to time
zero.
criticalTime =
time(startIndex:CriticalEndIndex) -
time(startIndex);
criticalTheta =
theta(startIndex:CriticalEndIndex);
criticalPulseVoltage =
pulseVoltage(startIndex:CriticalEndIndex);

%what is the window of time that we are
analyzing?
timeAnalyzed =
criticalTime(length(criticalTime));

%The response should be a second order
function, so we can polyfit it
coefficients =
polyfit(criticalTime,criticalTheta,2);

sprintf('Coefficients of fitted line (a*x^2
+ b*x + c)\n\n  a = %2.5f\n  b = %2.5f\n  c
=
%2.5f',coefficients(1),coefficients(2),coeff
icients(3))

fittedTheta =
(coefficients(1)*criticalTime.^2)+(coefficie
nts(2)*criticalTime)+(coefficients(3));

%Plot the response
if graphselection==1
    figure

plot(criticalTime,criticalTheta,criticalTime
,fittedTheta)
    title('\theta vs. Time Only for Period
Analyzed and Fitted')
    xlabel('Time (seconds)')
    ylabel('\theta (radians)')
    legend('Actual Data','2nd Order Fit')
    grid
else
end


%We can now solve for angular acceleration
by differentiating the fitted expression for
%theta twice.  This will simply yield
2*coefficients(1)
thetadoubledot = coefficients(1)*2;
```

```
sprintf('Angular Acceleration = %2.4f
rad/sec^2 (derived from curve fitting theta
vs. time)',thetadoubledot)

%We are now in a position to determine Jm.
According to the mechanical
%equation, the only torques on the system
are the Torque produced by the
%motor and a friction force that is equal to
the low omega value (considered
%a constant)
%
%   Jm*thetadoubledot = T(t) -
T_friction_small_w = Kt*i(t) +
T_friction_small_w    (no external loads)
%
%We can find an expression for i(t) from the
electrical side
%
%   Va = La*(di/dt) + (Rm+R)i(t)
(small omega)
%
%In order to solve for i(t) We should see if
the current changes as a
%function of time.  To do this, I hooked up
the scope to the measuring
%resistor and measured the voltage using the
scope as the motor started to
%turn.
%
%We can load the data saved from the scope.
The file is of the form
%
%     1st Column        2nd Column
%
%     time (sec)        Ch. 1 (volts)
%
load wk2_A_6.txt

%Extract time (sec), and voltage from the
scope channel (volts)
time_wk2_A_6 = wk2_A_6(:,1);
%Extract time (sec)
voltage_wk2_A_6 = wk2_A_6(:,2);
%Extract channel 1 voltage (volts)

%calculate the current
current_wk2_A_6 = voltage_wk2_A_6/R;
%amps

%We can see when the motor starts to turn by
comparing when the current
%starts to drop.  From inspection of the
data, it appears that this occurs
%at t = -3.83*10^-3 seconds
dropTime = -3.83*10^-3;
dropIndex = find(time_wk2_A_6==dropTime);

%lets find out when the current reaches
steady state.  From inspection of
%the graph, the current reaches steady state
again at roughly t =
%-1.84*10^-3 seconds
SteadyStateTime = -1.84*10^-3;
SteadyStateIndex =
find(time_wk2_A_6==SteadyStateTime);

%lets calculate how the time period during
which the current is changing
ChangingCurrentTime = SteadyStateTime -
dropTime;

sprintf('Current is changing for %1.5f
seconds.  It is probably safe to ignore
since it is changing for only a short
time',ChangingCurrentTime)


%We need to calculate the slope between two
points spaced a time apart
%in order to filter out the noise.  We only
need to calculate the slope for
%when the current is changing
IndexInterval = 15;

for IndexCounter=dropIndex:SteadyStateIndex

    %We need to check that we aren't at the
end of the array (can't add
    %IndexInterval to the end of the array
    if IndexCounter >
length(current_wk2_A_6) - IndexInterval
        break
    else
    end

    %Calculate the run
    TimeInterval = time_wk2_A_6(IndexCounter
+ IndexInterval) -
time_wk2_A_6(IndexCounter);

    %Calculate the slope from this point to
a point that is an IndexInterval away.
    DiDt(IndexCounter) =
(current_wk2_A_6(IndexCounter +
IndexInterval) -
current_wk2_A_6(IndexCounter))/TimeInterval;
end

%We need to create an array of time to plot
the slopes against.  We need to calculate
%the time intervals that the oscilloscope
uses
scopeTimeInterval = time_wk2_A_6(2) -
time_wk2_A_6(1);

%What is the last time that the slope was
calculated for?  Remember that the time
starts
%out negative on the scope
endtime = time_wk2_A_6(IndexCounter)-
time_wk2_A_6(1);

%create an array of times
DiDtTime =
[scopeTimeInterval:scopeTimeInterval:endtime
-scopeTimeInterval*3];

if graphselection==1
    figure
    plot(time_wk2_A_6 -
dropTime,current_wk2_A_6)
    title('i(t) vs. Time When Motor is
Subjected to a Pulse. Notice Current is not
Changing for Very Long.')
    xlabel('Time (seconds)')
    ylabel('Current (amps)')
    axis([min(time_wk2_A_6)-dropTime
max(time_wk2_A_6) min(current_wk2_A_6)
max(current_wk2_A_6)])
    grid
else
end

%If we are ignoring the changing current, we
can now calculate the current.
% The current is now given by
%
%   i(t) = (Ka*Vsupply)/(R+Rm)      where Ka
= amplifer gain
%
%We can solve for the amplifier gain by
measuring the voltage before and
%after the amplifier
VbeforeAmp = 0.996;             %volts
VafterAmp = -3.560;             %volts

Ka = VafterAmp/VbeforeAmp;      %note that
amp flips sign of signal

%calculate current
current_wk2_A_6 = (Ka*VbeforeAmp)/(R+Rm);
%amps

%We now need to calculate the friction force
at small values of omega.  From
%section "determining bm", we can find the
friction force at small omega.  We
%need to first determine if the motor is
spinning in the positive or negative
%direction.  Recall that we can use the
curve fitted data coefficients to determine
%the direction or rotation
%
%   1 = CCW (positive)
%   2 = CW (negative)
%
%
if coefficients(1)>1
    rotationDirection = 1;      %CCW
(positive)
else
    rotationDirection = 2;      %CW
(negative
end

%We can now find out which value of bm*omega
to use.
if rotationDirection == 1
    T_friction_small_w = offsetPositive;
%use positive fitted value of torque at zero
omega
else
    T_friction_small_w = offsetNegative;
%use negative fitted value of torque at zero
omega
end

%Now calculate Jm
Jm = ((current_wk2_A_6*Kt)-
T_friction_small_w)/thetadoubledot;

sprintf('Jm = %1.5f kg/m^2 (moment of
inertia of wheel)',Jm)


disp('Total time required to run
calculations')
toc


%-----------------------DSA OUTPUT----------
----------------------------
%
```

```
%We would like to analyze the DSA output.
We will call the function
%DSA_data to do this.  This will calculate
the theoretical bode diagram
%and convert the output of the DSA into bode
diagrams.
%
%We need to decide which value of La it
should use to calculate the
%theoretical bode diagram (Eq.10 in the
report).
La_DSA = La;

%Do you want to analyze DSA data?
disp('Do you want to analyze DSA data?')
disp('  1 = yes')
disp('  2 = no')
disp('')
disp('')
selection = input('Enter selection:  ');

if selection == 1
    %Call function to create plots
    DSA_data(R,Rm,La_DSA,Ka);
else
end



%-----------------------MODEL VALIDATION----
-----------------------------


%----------------Open Loop Pulse Response---
------------------------------
%Do you want to analyze Pulse data?
disp('Do you want to analyze Pulse Voltage
data?')
disp('  1 = yes')
disp('  2 = no')
disp('')
disp('')
selection3 = input('Enter selection:  ');

if selection3 == 1
    disp('---------------------Analyzing
Pulse Voltage Data--------------------------
')

    %run the pulse simulation
    sim('lab2_pulse')

    %load the simulated parameters
    sim_time = simVa.time;

    sim_Va = simVa.signals.values;
    sim_theta = simTheta.signals.values;
    sim_omega = simOmega.signals.values;

    sim_Vin = sim_Va/Ka;

    %Lets see if the two pulses are the same
    max_pulse = max(pulseVoltage);
    max_sim_pulse = max(sim_Vin);

    sprintf('Actual Pulse Magnitude = %1.3
volts.\nSimulated Pulse Magnitude = %1.3
volts',max_pulse,max_sim_pulse)

    %plot the voltages to ensure that the
step is the same
    figure

plot(time,pulseVoltage,sim_time,sim_Vin,'rx'
)
    title('Actual Pulse of V_i_n and
Simulated V_i_n vs. Time.  Ka = -3.5743')
    xlabel('Time (seconds)')
    ylabel('V_i_n (volts)')
    legend('Actual','Simulated',2)
    axis([0.98 1.18 -0.1 1.1])
    grid

    %Now lets compare how far the wheels
rotated.  Its eaiser to look at
    %the graph in degrees
    theta_pulse_deg = theta*(180/pi);
    sim_theta_deg = sim_theta*(180/pi);

    figure

plot(time,theta_pulse_deg,sim_time,sim_theta
_deg)
    title('\theta vs. Time For Pulse Signal
with \Deltat = 0.13 seconds')
    xlabel('time (seconds)')
    ylabel('\theta (degrees)')
    legend('Actual','Simulated')
    axis([1 1.75 -200 0])
    grid

    %what happens if we modifiy Ka?  What is
Ka is 10% less that measured
    %at steady state
    Ka = Ka*(1.1);

    sim('lab2_pulse')

    sim_theta_deg =
simTheta.signals.values*(180/pi);

    figure

plot(time,theta_pulse_deg,sim_time,sim_theta
_deg)
    title('\theta vs. Time For Pulse Signal
with 10% change in K_a')
    xlabel('time (seconds)')
    ylabel('\theta (degrees)')
    legend('Actual','Simulated')
    axis([1 1.75 -200 0])
    grid

else
end


%----------------Velocity Control System----
-----------------------------
disp('Do you want to analyze velocity
controller data?')
disp('  1 = yes')
disp('  2 = no')
disp('')
disp('')
selection4 = input('Enter selection:  ');

if selection4==1
    disp('---------------------Analyzing
Velocity Controller-------------------------
-')

    %load the pertinent run.  We need to
keep in mind that the structure is
    %different depending on the run number.
At run 14, and extra output
    %was added to channel 8 as theta_ref

    %This data only contains 7 channels in
the form
    %
    %   ch.1 = theta
    %   ch.2 = Vin
    %   ch.3 = omega
    %   ch.4 = omega_ref
    %   ch.5 = proportional component
    %   ch.6 = integral component
    %   ch.7 = velocity error

    %load the appropriate run (MUST BE DONE
BY OPERATOR!!)
    load run_12

    %Set the parameters for this simulation
    Kvp = 0.22;
    Kvi = 0;

    TL = 0.083;

    %extract the data
    vel_time = time;
    vel_theta = computer_output(:,1);
    vel_Vin = computer_output(:,2);
    vel_omega = computer_output(:,3);
    vel_omega_ref = computer_output(:,4);
    vel_p_component = computer_output(:,5);
    vel_i_component = computer_output(:,6);
    vel_vel_error = computer_output(:,7);



    %Run the simulation
    sim('lab2_velocity_control')

    %extract simulated values
    vel_sim_time = simTheta.time;
    vel_sim_theta = simTheta.signals.values;
    vel_sim_Va = simVa.signals.values;
    vel_sim_Vin = vel_sim_Va/Ka;
    vel_sim_Omega = simOmega.signals.values;
    vel_sim_Omega_Ref =
simOmegaRef.signals.values;
    vel_sim_Omega_Cmd =
simOmegaCmd.signals.values;
    vel_sim_p_component =
simPComponent.signals.values;
    vel_sim_i_component =
simIComponent.signals.values;
    vel_sim_Omega_Estimate =
simOmegaEstimate.signals.values;

    %plot some stuff
    figure

plot(vel_time,vel_omega_ref,vel_sim_time,vel
_sim_Omega_Ref)
    legend('Cmd','Ref')
```

```
    title('Reference Signals for Velocity
Control System')
    xlabel('Time (sec)')
    ylabel('\omega (rad/s)')
    legend('Lab','Simulation')
    grid

    figure

plot(time,vel_omega,vel_sim_time,vel_sim_Ome
ga)
    title('\omega vs. Time for Velocity
Control System')
    xlabel('Time (sec)')
    ylabel('\omega (rad/s)')
    legend('Lab','Simulation')
    grid

    figure

plot(vel_time,vel_omega_ref,vel_time,vel_ome
ga,vel_sim_time,vel_sim_Omega_Ref,vel_sim_ti
me,vel_sim_Omega)
    title('\omega vs. Time for Velocity
Control System with Reference Signals')
    xlabel('Time (sec)')
    ylabel('\omega (rad/s)')
    legend('Lab Reference','Lab
\omega','Simulation Reference','Simulation
\omega',4)
    grid

else
end

%----------------Position Control System----
-----------------------------
disp('Do you want to analyze position
controller data?')
disp('  1 = yes')
disp('  2 = no')
disp('')
disp('')
selection5 = input('Enter selection:  ');

if selection5==1
    disp('---------------------Analyzing
Position Controller-------------------------
-')

    %load the pertinent run.  We need to
keep in mind that the structure is
    %different depending on the run number.
At run 14, and extra output
    %was added to channel 8 as theta_ref

    %This data 8 channels in the form
    %
    %   ch.1 = theta
    %   ch.2 = Vin
    %   ch.3 = omega
    %   ch.4 = omega_ref
    %   ch.5 = proportional component
    %   ch.6 = integral component
    %   ch.7 = velocity error
    %   ch.8 = theta_ref

    %load the appropriate run (MUST BE DONE
BY OPERATOR!!)
    load run_20

    %Set the parameters for this simulation
    Kvp = 1;
    Kvi = 1;
    Kp = 7;

    TL = 0;

    %extract the data
    pos_time = time;
    pos_theta = computer_output(:,1);
    pos_Vin = computer_output(:,2);
    pos_omega = computer_output(:,3);
    pos_omega_ref = computer_output(:,4);
    pos_vel_p_component =
computer_output(:,5);
    pos_vel_i_component =
computer_output(:,6);
    pos_vel_error = computer_output(:,7);
    pos_theta_ref = computer_output(:,8);


    %Run the simulation
    sim('lab2_position_control')

    %extract simulated values
    pos_sim_time = simTheta.time;
    pos_sim_theta = simTheta.signals.values;
    pos_sim_Va = simVa.signals.values;
    pos_sim_Vin = pos_sim_Va/Ka;
    pos_sim_Omega = simOmega.signals.values;
    pos_sim_Omega_Ref =
simOmegaRef.signals.values;
    pos_sim_Omega_Cmd =
simOmegaCmd.signals.values;
    pos_vel_sim_p_component =
simPComponent.signals.values;
    pos_vel_sim_i_component =
simIComponent.signals.values;
    pos_sim_Omega_Estimate =
simOmegaEstimate.signals.values;
    pos_sim_theta_cmd =
simThetaCmd.signals.values;
    pos_sim_theta_ref =
simThetaRef.signals.values;

    %plot some stuff
    figure

plot(pos_time,pos_theta_ref,pos_time,pos_the
ta,pos_sim_time,pos_sim_theta_ref,pos_sim_ti
me,pos_sim_theta)
    title('\theta vs. Time For Position
Control System with Reference Signals')
    xlabel('Time (seconds)')
    ylabel('\theta (radians)')
    legend('Lab Reference','Lab
\theta','Simulation Reference','Simulation
\theta',4)
    grid

else
end

%-----------------LABORATORY DISCUSSION
ITEMS---------------------------
disp('Do you want to analyze velocity
controller design frequency response?')
disp('  1 = yes')
disp('  2 = no')
disp('')
disp('')
selection6 = input('Enter selection:  ');

if selection6==1
    tic
    disp('-------------------Analyzing Non-
Linear Frequency Response------------')
    %Set the gains of the velocity
controller
    Kvp = 1;
    Kvi = 1;
    TL = 0;

    %Simulink computers the sin wave
according to the equation
    %
    %   y(t) = A*sin(frequency*t + phase)
    %
    %

    %We would like to increment the
frequency in logarithmic intervals.
    Frequency_range = logspace(1,3,40);
    SinAmplitude = 10;

    disp('Do you want to look at which
points the program is using to calculate
phase lag?')
    disp('  1 = yes')
    disp('  2 = no')
    disp('')
    disp('')
    selection7=  input('Enter selection:  ')

    disp('Starting Iterations')

    for
frequency_counter=1:length(Frequency_range)
        sprintf('Current Interation =
%1.1f',frequency_counter)
        desiredFrequency =
Frequency_range(frequency_counter);
        SinFrequency =
desiredFrequency*2*pi;

        sim('lab2_velocity_control_bode')

        %Extract Data
        vel_sim_time = simOmega.time;
        vel_sim_theta =
simTheta.signals.values;
        vel_sim_Omega =
simOmega.signals.values;
        vel_sim_Omega_Cmd =
simOmegaCmd.signals.values;

        %We need to pick a time to wait for
the response to settle out.
        wait_time = .25;      %second

        settled_indices =
find(vel_sim_time>wait_time);

        start_index = min(settled_indices);
        end_index = max(settled_indices);

        settled_time =
vel_sim_time(start_index:end_index);
        settled_input =
vel_sim_Omega_Cmd(start_index:end_index);
        settled_output =
vel_sim_Omega(start_index:end_index);
```

```
        %Now that we have these, we need to
check for when the input crosses
        %0 twice consecutively

        %What is the sign of current input
wave?
        start_sign = sign(settled_input(1));

        if start_sign==1
            disp('Input wave starts
positive')

            current_sign = start_sign;

            %Check for when the sign changes
            for
counter=2:length(settled_input)
                current_sign =
sign(settled_input(counter));
                if current_sign==start_sign
                else
                    break
                end
            end

            %counter is now on the index
where the sign changes
            first_sign_change_index =
counter;


            start_sign =
sign(settled_input(first_sign_change_index))
;

            for
counter=first_sign_change_index:length(settl
ed_input)
                current_sign =
sign(settled_input(counter));
                if current_sign==start_sign
                else
                    break
                end
            end

            %counter is now on the index
where the sign changes a second time
            second_sign_change_index =
counter;

        else
            disp('Input wave starts
negative')

            current_sign = start_sign;

            %Check for when the sign changes
            for
counter=2:length(settled_input)
                current_sign =
sign(settled_input(counter));
                if current_sign==start_sign
                else
                   break
                end
            end

            %counter is now on the index
where the sign changes
            first_sign_change_index =
counter;


            start_sign =
sign(settled_input(first_sign_change_index))
;

            for
counter=first_sign_change_index:length(settl
ed_input)
                current_sign =
sign(settled_input(counter));
                if current_sign==start_sign
                else
                    break
                end
            end

            %counter is now on the index
where the sign changes a second time
            second_sign_change_index =
counter;
        end

        %We now would like to zoom in on
only the section where one wave occurs
        input_one_wave_time =
settled_time(first_sign_change_index:second_
sign_change_index);
        input_one_wave_values =
settled_input(first_sign_change_index:second
_sign_change_index);

        %The period is simply twice this
hump time.  Likewise, the number of
        %indices that correspond to one
period is also found.
        period =
2*(max(input_one_wave_time)-
min(input_one_wave_time));
        period_indices =
2*length(input_one_wave_time);

        %we need to see if this is a max or
a min hump.  This will let us know
        %if we need to find the max or min
amplitude
        middle_input_index =
round(length(input_one_wave_time)/2);
        sign_of_input_hump =
sign(input_one_wave_values(middle_input_inde
x));

        if sign_of_input_hump==1
            disp('Positive Hump')


[input_max_value,input_max_index] =
max(input_one_wave_values);
            input_max_time =
input_one_wave_time(input_max_index);

            %We now want to look for the
next maximum from the output.  Since
            %the output always lags, we need
to start looking for the maximum
            %after the max_time. The output
should have the same period, so
            %can plot one period after the
hump
            output_start_index =
find(settled_time==input_max_time);
            output_end_index =
output_start_index + period_indices;

            output_one_wave_time =
settled_time(output_start_index:output_end_i
ndex);
            output_one_wave_values =
settled_output(output_start_index:output_end
_index);

            %Since we plotted one period,
there should only be one maximum

[output_max_value,output_max_index] =
max(output_one_wave_values);

            %What time does the max output
occur at?
            output_max_index =
find(output_one_wave_values==output_max_valu
e);
            output_max_time =
output_one_wave_time(output_max_index);

            %now find the phase lag
            deltaT = (output_max_time -
input_max_time);

            phaseLag = (deltaT/period)*(-
360);

            output_input_ratio =
output_max_value/input_max_value;

            if selection7==1
                close all
                figure

plot(input_one_wave_time,input_one_wave_valu
es,output_one_wave_time,output_one_wave_valu
es,input_max_time,input_max_value,'ro',outpu
t_max_time,output_max_value,'rx')
                title(['Input and Output Sin
Waves with Input Frequency =
',num2str(desiredFrequency),' rad/s'])
                xlabel('Time (sec)')
                ylabel('Amplitude')

legend('Input','Output','Input Max','Output
Max')
                grid
            else
            end



        else
            disp('Negative Hump')

[input_max_value,input_max_index] =
min(input_one_wave_values);
            input_max_time =
input_one_wave_time(input_max_index);

            output_start_index =
find(settled_time==input_max_time);
            output_end_index =
output_start_index + period_indices;

            output_one_wave_time =
settled_time(output_start_index:output_end_i
ndex);
```

```
            output_one_wave_values =
settled_output(output_start_index:output_end
_index);


[output_max_value,output_max_index] =
min(output_one_wave_values);

            output_max_index =
find(output_one_wave_values==output_max_valu
e);
            output_max_time =
output_one_wave_time(output_max_index);

            deltaT = (output_max_time -
input_max_time);

            phaseLag = (deltaT/period)*(-
360);

            output_input_ratio =
output_max_value/input_max_value;

            if selection7==1
                close all
                figure

plot(input_one_wave_time,input_one_wave_valu
es,output_one_wave_time,output_one_wave_valu
es,input_max_time,input_max_value,'ro',outpu
t_max_time,output_max_value,'rx')
                title(['Input and Output Sin
Waves with Input Frequency =
',num2str(desiredFrequency),' rad/s'])
                xlabel('Time (sec)')
                ylabel('Amplitude')

legend('Input','Output','Input Max','Output
Max')
                grid
            else
            end

        end

        phaseLagArray(frequency_counter) =
phaseLag;

output_input_ratio_Array(frequency_counter)
= output_input_ratio;

    end

    %calculate decibels
    decibels =
20*log10(output_input_ratio_Array);

    %----------------GAIN MARGIN------------
-----------------------
    %We now need to find the gain and phase
margin as well as the
    %bandwidth.

    %Check to see if the phase drops below
180 degrees
    if min(phaseLagArray)<-180

        Phase_indices_less_than_180 =
find(phaseLagArray<-180);

        Phase_180_index =
Phase_indices_less_than_180(1);
        Frequency_at_Phase_180 =
Frequency_range(Phase_180_index);

        %What is the actual phase used for
calculations? (should be around
        %-180)
        Actual_Phase =
phaseLagArray(Phase_180_index)

        %What is the dB at this index?
        decibels_at_Phase_180 =
decibels(Phase_180_index);

        %If the decibels is below zero, then
this is a positive gain margin.
        if sign(decibels_at_Phase_180)==1
            sprintf('Gain margin = %2.5f dB
(negative gain
margin)',decibels_at_Phase_180)
        else
            sprintf('Gain margin = %2.5f dB
(positive gain
margin)',abs(decibels_at_Phase_180))
        end
    else
        disp('Phase margin is infinite')
        Frequency_at_Phase_180 =
max(Frequency_range);
        decibels_at_Phase_180 =
min(decibels);
        Actual_Phase = max(Frequency_range);
    end


    %---------------PHASE MARGIN------------
------------------------

    %Where is decibels zero?
    decibels_indices_less_than_zero =
find(decibels<0);

    decibels_equal_zero_index =
decibels_indices_less_than_zero(1);
    Frequency_at_decibel_zero =
Frequency_range(decibels_equal_zero_index);

    %What is the actual decibel gain used
for calculatations? (should be
    %around zero, but negative).
    Actual_decibels =
decibels(decibels_equal_zero_index);

    %What is the phase at this index?
    phase_at_decibels_zero =
phaseLagArray(decibels_equal_zero_index);

    phase_margin = phase_at_decibels_zero +
180;

    %If the phase margin is more than -180,
it is positive
    if phase_margin > 0
        sprintf('Phase margin = %2.5f
degrees (positive phase
margin)',phase_margin)
    else
        sprintf('Phase margin = %2.5f
degrees (negative phase
margin)',phase_margin)
    end


    %We can now calculate the bandwidth.
This is defined where the
    %response drops 3 dB from the low
frequency value
    DC_gain_dB = decibels(1);

    decibel_indices_less_than_negative_3_dB
= find(decibels < DC_gain_dB-3);
    bandwidth_index =
decibel_indices_less_than_negative_3_dB(1);
    Actual_bandwidth_decibels =
decibels(bandwidth_index);

    %what frequency does this occur at?
    bandwidth_frequency =
Frequency_range(bandwidth_index);

    sprintf('Bandwidth = %3.3f
rad/s',bandwidth_frequency)

    figure
    subplot(2,1,1)

semilogx(Frequency_range,decibels,Frequency_
at_Phase_180,decibels_at_Phase_180,'ro',Freq
uency_at_decibel_zero,Actual_decibels,'ko',b
andwidth_frequency,Actual_bandwidth_decibels
,'go')
    title('Bode Plot Velocity Control System
with Non-Linear Lookup Table')
    ylabel('Magnitude (dB)')
    legend('Decibels','Gain Margin','Gain =
0 dB','Bandwidth',3)
    grid

    subplot(2,1,2)

semilogx(Frequency_range,phaseLagArray,Frequ
ency_at_Phase_180,Actual_Phase,'ro',Frequenc
y_at_decibel_zero,phase_at_decibels_zero,'ko
')
    ylabel('Phase (deg)')
    xlabel('Frequency (rad/s)')
    legend('Phase','Phase = -180','Phase
Margin',3)
    axis([10 1000 min(phaseLagArray)
max(phaseLagArray)])
    grid




else    %closes do you want to analyze
velcoity control frequency response if
end
toc

disp('Do you want to analyze the linearized
velocity control model?')
disp('  1 = yes')
disp('  2 = no')
disp('')
disp('')
selection8 = input('Enter selection:  ');

if selection8==1
    disp('----------------------Analyzing
Linearized Velocity Control Model----------
-------')

    Kvp = 1;
    Kvi = 1;
```

```
    TL = 0;

    SinAmplitude = 0;
    SinFrequency = 0;

    %use linmod to obtain the linearied
state space model
    [a_vel,b_vel,c_vel,d_vel] =
linmod('lab2_velocity_control_bode');
    [num_vel,den_vel] =
ss2tf(a_vel,b_vel,c_vel,d_vel);

    disp('Linearized Transfer Function of
Velocity Control Model')
    linearized_velocity_control =
tf(num_vel,den_vel)

    figure
    bode(linearized_velocity_control)
    title('Bode Plot of Linearized Model of
Velocity Control System')
    grid

    figure
    margin(linearized_velocity_control)

    %Calcualte the bandwidth
    linearized_velocity_bandwidth =
bandwidth(linearized_velocity_control);

    sprintf('Bandwidth of linearized model =
%2.5f rad/s',linearized_velocity_bandwidth)


else
end
```

```
disp('Do you want to analyze position
controller design frequency response?')
disp('  1 = yes')
disp('  2 = no')
disp('')
disp('')
selection9 = input('Enter selection:  ');

if selection9==1
    tic
    disp('-------------------Analyzing Non-
Linear Frequency Response------------')
    %Set the gains of the velocity
controller
    Kvp = 1;
    Kvi = 1;
    Kp = 7;
    TL = 0;

    Frequency_range = logspace(1,3,40);
    SinAmplitude = 10;

    disp('Do you want to look at which
points the program is using to calculate
phase lag?')
    disp('  1 = yes')
    disp('  2 = no')
    disp('')
    disp('')
    selection10=  input('Enter selection:
')

    disp('Starting Iterations')

    for
frequency_counter=1:length(Frequency_range)
        sprintf('Current Interation =
%1.1f',frequency_counter)
        desiredFrequency =
Frequency_range(frequency_counter);
        SinFrequency =
desiredFrequency*2*pi;

        sim('lab2_position_control_bode')

        %Extract Data
        pos_sim_time = simOmega.time;
        pos_sim_Theta =
simTheta.signals.values;
        pos_sim_Omega =
simOmega.signals.values;
        pos_sim_Theta_Cmd =
simThetaCmd.signals.values;

        %We need to pick a time to wait for
the response to settle out.
        wait_time = 1;      %second
```

```
        settled_indices =
find(pos_sim_time>wait_time);

        start_index = min(settled_indices);
        end_index = max(settled_indices);

        settled_time =
pos_sim_time(start_index:end_index);
        settled_input =
pos_sim_Theta_Cmd(start_index:end_index);
        settled_output =
pos_sim_Theta(start_index:end_index);

        %Now that we have these, we need to
check for when the input crosses
        %0 twice consecutively

        %What is the sign of current input
wave?
        start_sign = sign(settled_input(1));

        if start_sign==1
            disp('Input wave starts
positive')

            current_sign = start_sign;

            %Check for when the sign changes
            for
counter=2:length(settled_input)
                current_sign =
sign(settled_input(counter));
                if current_sign==start_sign
                else
                    break
                end
            end

            %counter is now on the index
where the sign changes
            first_sign_change_index =
counter;


            start_sign =
sign(settled_input(first_sign_change_index))
;

            for
counter=first_sign_change_index:length(settl
ed_input)
                current_sign =
sign(settled_input(counter));
                if current_sign==start_sign
                else
                    break
                end
            end

            %counter is now on the index
where the sign changes a second time
            second_sign_change_index =
counter;

        else
            disp('Input wave starts
negative')

            current_sign = start_sign;

            %Check for when the sign changes
            for
counter=2:length(settled_input)
                current_sign =
sign(settled_input(counter));
                if current_sign==start_sign
                else
                   break
                end
            end

            %counter is now on the index
where the sign changes
            first_sign_change_index =
counter;


            start_sign =
sign(settled_input(first_sign_change_index))
;

            for
counter=first_sign_change_index:length(settl
ed_input)
                current_sign =
sign(settled_input(counter));
                if current_sign==start_sign
                else
                    break
                end
            end

            %counter is now on the index
where the sign changes a second time
            second_sign_change_index =
counter;
        end
```

```
        %We now would like to zoom in on
only the section where one wave occurs
        input_one_wave_time =
settled_time(first_sign_change_index:second_
sign_change_index);
        input_one_wave_values =
settled_input(first_sign_change_index:second
_sign_change_index);

        %The period is simply twice this
hump time.  Likewise, the number of
        %indices that correspond to one
period is also found.
        period =
2*(max(input_one_wave_time)-
min(input_one_wave_time));
        period_indices =
2*length(input_one_wave_time);

        %we need to see if this is a max or
a min hump.  This will let us know
        %if we need to find the max or min
amplitude
        middle_input_index =
round(length(input_one_wave_time)/2);
        sign_of_input_hump =
sign(input_one_wave_values(middle_input_inde
x));

        if sign_of_input_hump==1
            disp('Positive Hump')


[input_max_value,input_max_index] =
max(input_one_wave_values);
            input_max_time =
input_one_wave_time(input_max_index);

            %We now want to look for the
next maximum from the output.  Since
            %the output always lags, we need
to start looking for the maximum
            %after the max_time. The output
should have the same period, so
            %can plot one period after the
hump
            output_start_index =
find(settled_time==input_max_time);
            output_end_index =
output_start_index + period_indices;

            output_one_wave_time =
settled_time(output_start_index:output_end_i
ndex);
            output_one_wave_values =
settled_output(output_start_index:output_end
_index);

            %Since we plotted one period,
there should only be one maximum

[output_max_value,output_max_index] =
max(output_one_wave_values);

            %What time does the max output
occur at?
            output_max_index =
find(output_one_wave_values==output_max_valu
e);
            output_max_time =
output_one_wave_time(output_max_index);

            %now find the phase lag
            deltaT = (output_max_time -
input_max_time);

            phaseLag = (deltaT/period)*(-
360);

            output_input_ratio =
output_max_value/input_max_value;

            if selection10==1
                close all
                figure

plot(input_one_wave_time,input_one_wave_valu
es,output_one_wave_time,output_one_wave_valu
es,input_max_time,input_max_value,'ro',outpu
t_max_time,output_max_value,'rx')
                title(['Input and Output Sin
Waves with Input Frequency =
',num2str(desiredFrequency),' rad/s'])
                xlabel('Time (sec)')
                ylabel('Amplitude')

legend('Input','Output','Input Max','Output
Max')
                grid
            else
            end



        else
            disp('Negative Hump')
[input_max_value,input_max_index] =
min(input_one_wave_values);
            input_max_time =
input_one_wave_time(input_max_index);

            output_start_index =
find(settled_time==input_max_time);
            output_end_index =
output_start_index + period_indices;

            output_one_wave_time =
settled_time(output_start_index:output_end_i
ndex);
            output_one_wave_values =
settled_output(output_start_index:output_end
_index);


[output_max_value,output_max_index] =
min(output_one_wave_values);

            output_max_index =
find(output_one_wave_values==output_max_valu
e);
            output_max_time =
output_one_wave_time(output_max_index);

            deltaT = (output_max_time -
input_max_time);

            phaseLag = (deltaT/period)*(-
360);

            output_input_ratio =
output_max_value/input_max_value;

            if selection10==1
                close all
                figure

plot(input_one_wave_time,input_one_wave_valu
es,output_one_wave_time,output_one_wave_valu
es,input_max_time,input_max_value,'ro',outpu
t_max_time,output_max_value,'rx')
                title(['Input and Output Sin
Waves with Input Frequency =
',num2str(desiredFrequency),' rad/s'])
                xlabel('Time (sec)')
                ylabel('Amplitude')

legend('Input','Output','Input Max','Output
Max')
                grid
            else
            end

        end

        phaseLagArray(frequency_counter) =
phaseLag;

output_input_ratio_Array(frequency_counter)
= output_input_ratio;

    end

    %calculate decibels
    decibels =
20*log10(output_input_ratio_Array);

    %----------------GAIN MARGIN------------
----------------------
    %We now need to find the gain and phase
margin as well as the
    %bandwidth.

    %Check to see if the phase drops below
180 degrees
    if min(phaseLagArray)<-180

        Phase_indices_less_than_180 =
find(phaseLagArray<-180);

        Phase_180_index =
Phase_indices_less_than_180(1);
        Frequency_at_Phase_180 =
Frequency_range(Phase_180_index);

        %What is the actual phase used for
calculations? (should be around
        %-180)
        Actual_Phase =
phaseLagArray(Phase_180_index)

        %What is the dB at this index?
        decibels_at_Phase_180 =
decibels(Phase_180_index);

        %If the decibels is below zero, then
this is a positive gain margin.
        if sign(decibels_at_Phase_180)==1
            sprintf('Gain margin = %2.5f dB
(negative gain
margin)',decibels_at_Phase_180)
        else
```

```
            sprintf('Gain margin = %2.5f dB
(positive gain
margin)',abs(decibels_at_Phase_180))
        end
    else
        disp('Phase margin is infinite')
        Frequency_at_Phase_180 =
max(Frequency_range);
        decibels_at_Phase_180 =
min(decibels);
        Actual_Phase = max(Frequency_range);
    end


    %---------------PHASE MARGIN------------
------------------------
    %Where is decibels zero?
    decibels_indices_less_than_zero =
find(decibels<0);

    decibels_equal_zero_index =
decibels_indices_less_than_zero(1);
    Frequency_at_decibel_zero =
Frequency_range(decibels_equal_zero_index);

    %What is the actual decibel gain used
for calculatations? (should be
    %around zero, but negative).
    Actual_decibels =
decibels(decibels_equal_zero_index);

    %What is the phase at this index?
    phase_at_decibels_zero =
phaseLagArray(decibels_equal_zero_index);

    phase_margin = phase_at_decibels_zero +
180;

    %If the phase margin is more than -180,
it is positive
    if phase_margin > 0
        sprintf('Phase margin = %2.5f
degrees (positive phase
margin)',phase_margin)
    else
        sprintf('Phase margin = %2.5f
degrees (negative phase
margin)',phase_margin)
    end


    %We can now calculate the bandwidth.
This is defined where the
    %response drops 3 dB from the low
frequency value
    DC_gain_dB = decibels(1);

    decibel_indices_less_than_negative_3_dB
= find(decibels < DC_gain_dB-3);
    bandwidth_index =
decibel_indices_less_than_negative_3_dB(1);
    Actual_bandwidth_decibels =
decibels(bandwidth_index);

    %what frequency does this occur at?
    bandwidth_frequency =
Frequency_range(bandwidth_index);

    sprintf('Bandwidth = %3.3f
rad/s',bandwidth_frequency)

    figure
    subplot(2,1,1)

semilogx(Frequency_range,decibels,Frequency_
at_Phase_180,decibels_at_Phase_180,'ro',Freq
uency_at_decibel_zero,Actual_decibels,'ko',b
andwidth_frequency,Actual_bandwidth_decibels
,'go')
    title('Bode Plot Position Control System
with Non-Linear Lookup Table')
    ylabel('Magnitude (dB)')
    legend('Decibels','Gain Margin','Gain =
0 dB','Bandwidth',3)
    grid

    subplot(2,1,2)

semilogx(Frequency_range,phaseLagArray,Frequ
ency_at_Phase_180,Actual_Phase,'ro',Frequenc
y_at_decibel_zero,phase_at_decibels_zero,'ko
')
    ylabel('Phase (deg)')
    xlabel('Frequency (rad/s)')
    legend('Phase','Phase = -180','Phase
Margin',3)
    axis([10 1000 min(phaseLagArray)
max(phaseLagArray)])
    grid




else    %closes do you want to analyze
position control frequency response if
end
toc
```

```
disp('Do you want to analyze the linearized
position control model?')
disp('  1 = yes')
disp('  2 = no')
disp('')
disp('')
selection11 = input('Enter selection:  ');

if selection11==1
    disp('----------------------Analyzing
Linearized Position Control Model-----------
-------')

    Kvp = 1;
    Kvi = 1;
    TL = 0;
    Kp = 7;

    SinAmplitude = 0;
    SinFrequency = 0;

    %use linmod to obtain the linearied
state space model
    [a_pos,b_pos,c_pos,d_pos] =
linmod('lab2_position_control_bode');
    [num_pos,den_pos] =
ss2tf(a_pos,b_pos,c_pos,d_pos);

    disp('Linearized Transfer Function of
Position Control Model')
    linearized_position_control =
tf(num_pos,den_pos)

    figure
    bode(linearized_position_control)
    title('Bode Plot of Linearized Model of
Position Control System')
    grid

    figure
    margin(linearized_position_control)

    %Calcualte the bandwidth
    linearized_position_bandwidth =
bandwidth(linearized_position_control);

    sprintf('Bandwidth of linearized model =
%2.5f rad/s',linearized_position_bandwidth)


else
end
```

## DSA_data.m

```
function DSA_data(R,Rm,La,Ka)
%INPUT:  R, Rm, and La used to create
theoretical bode diagram.  Ka is
%amplifier gain
%OUTPUT: None

disp('----------------------------Analyzing
DSA Data------------------------------')

%------------------CREATE THEORETICAL BODE
PLOT---------------------------
%
%Plot the bode diagram of Eq.10
num = [1/(Rm+R)];
den = [La/(Rm+R) 1];

theoretical_transfer_function = tf(num,den);

figure
bode(theoretical_transfer_function)
title('Theoretical Bode Diagram of
I(s)/V_a(s)')

%------------------ANALYZING DSA DATA------
--------------------------------
%
%We just ran the program to convert the DSA
data to matlab data.  We can
%now load it

%---------------------Sin-Sweep in Position
1----------------------------
load ss1

%this loads the variables o2i1x and o2i1, we
need to make these specific to
%this wheel location
frequency_SS1 = o2i1x;
o2i1SS1 = o2i1;

%the frequency response is given as a
complex number.  We need to take the
%absolute magnitude to obtain the magnitude
and calcualate the angle of the
%complex number to obtain the phase
voltage_magnitude_SS1 = abs(o2i1SS1);

%we need to keep in mind that the DSA is
actually computing the transfer
%function between Vr and Vin (voltage before
the amplifier)
%
%       H(s) = Vr(s)/Vin(s)
%
%We would like to compare the DSA output
with the transfer function given
%by the electrical equation given by
%
%       G(s) = I(s)/Va(s)
%
%H(s) can be converted to G(s) by simply
taking into account the
%appropriate factors.  By dividing H(s) by
R*Ka, we obtain
%
%       H(s)/(R*Ka) = Vr(s)/(Vin(s)*R*Ka) =
I(s)/Va(s)
%
%       G(s) = H(s)/(R*Ka)
%
%Since the DSA is giving us H(s), we simply
need to account for the
%constant gain of (1/(R*Ka)).  This will
only affect the magnitude.  Also
%keep in mind that we need to take the
absolute magnitude of Ka since we
%have defined it as negative, but we are
only interested in the magnitude.
%
%We can calculate |H(s)| in decibels
decibels_voltage_SS1 =
20*log10(voltage_magnitude_SS1);

%We can now simply add the factor or
(1/(R*Ka)) in decibels
shift_factor_decibels =
20*log10(1/(abs(Ka)*R));

%The transfer function of G(s) is now given
by
current_magnitude_SS1_dB =
decibels_voltage_SS1 +
shift_factor_decibels;

%break up the real and imaginary parts of
the array
real_SS1 = real(o2i1SS1);
imaginary_SS1 = imag(o2i1SS1);

%angle is simple given by
phase_SS1_radians =
atan(imaginary_SS1./real_SS1);
phase_SS1_degrees =
(phase_SS1_radians/(2*pi))*360;

%---------------------Sin-Sweep in Position
2----------------------------
%performing the same operation for the wheel
in position 2
load ss2

frequency_SS2 = o2i1x;
o2i1SS2 = o2i1;

voltage_magnitude_SS2 = abs(o2i1SS2);

decibels_voltage_SS2 =
20*log10(voltage_magnitude_SS2);

current_magnitude_SS2_dB =
decibels_voltage_SS2 +
shift_factor_decibels;

real_SS2 = real(o2i1SS2);
imaginary_SS2 = imag(o2i1SS2);

phase_SS2_radians =
atan(imaginary_SS2./real_SS2);
phase_SS2_degrees =
(phase_SS2_radians/(2*pi))*360;

%---------------------Sin-Sweep in Position
3----------------------------
%performing the same operation for the wheel
in position 3
load ss3

frequency_SS3 = o2i1x;
o2i1SS3 = o2i1;

voltage_magnitude_SS3 = abs(o2i1SS3);

decibels_voltage_SS3 =
20*log10(voltage_magnitude_SS3);

current_magnitude_SS3_dB =
decibels_voltage_SS3 +
shift_factor_decibels;

real_SS3 = real(o2i1SS3);
imaginary_SS3 = imag(o2i1SS3);

phase_SS3_radians =
atan(imaginary_SS3./real_SS3);
phase_SS3_degrees =
(phase_SS3_radians/(2*pi))*360;

%---------------------Sin-Sweep in Position
4----------------------------
%performing the same operation for the wheel
in position 4
load ss4

frequency_SS4 = o2i1x;
o2i1SS4 = o2i1;

voltage_magnitude_SS4 = abs(o2i1SS4);

decibels_voltage_SS4 =
20*log10(voltage_magnitude_SS4);

current_magnitude_SS4_dB =
decibels_voltage_SS4 +
shift_factor_decibels;

real_SS4 = real(o2i1SS4);
imaginary_SS4 = imag(o2i1SS4);

phase_SS4_radians =
atan(imaginary_SS4./real_SS4);
phase_SS4_degrees =
(phase_SS4_radians/(2*pi))*360;

%Plot all of these together to make sure
they are similar
figure
subplot(2,1,1)
semilogx(frequency_SS1,current_magnitude_SS1
_dB,frequency_SS2,current_magnitude_SS2_dB,f
requency_SS3,current_magnitude_SS3_dB,freque
ncy_SS4,current_magnitude_SS4_dB)
title('Bode Plot of DC Motor I(s)/V_a(s)
Using Sweeping Sine Wave as Excitation')
ylabel('Magnitude (dB)')
legend('Position 1','Position 2','Position
3','Position 4',3)

subplot(2,1,2)
semilogx(frequency_SS1,phase_SS1_degrees,fre
quency_SS2,phase_SS2_degrees,frequency_SS3,p
hase_SS3_degrees,frequency_SS4,phase_SS4_deg
rees)
ylabel('Phase (deg)')
xlabel('Frequency (rad/s)')
legend('Position 1','Position 2','Position
3','Position 4',3)

%We now need to now calculate the break
frequency.  By analyzing the plot,
%it appears  that position 3 yields the best
results, therefore we will use
%this as the reference.
DC_gain_SS = current_magnitude_SS3_dB(1);
```

```
%We would like to find out when the signal
drops by 3 dB
critical_dB = DC_gain_SS-3;

for
counter=1:length(current_magnitude_SS3_dB)
    if current_magnitude_SS3_dB(counter) <
critical_dB
        break
    else
    end
end

%Now find the frquency where the magnitude
drops by 3 dB
break_frequency_SS = frequency_SS3(counter);

%Using this, calculate La
La_SS = (Rm+R)/break_frequency_SS;

sprintf('omega break = %2.3f (break
frequency from sin wave excitation)\nLa =
%2.5f (motor inductance using sin wave bode
plots)',break_frequency_SS,La_SS)

%-----------------------RANDOM NOISE--------
----------------------------
%
%We can also generate a bode plot using the
random noise function.
%Instead of methodically increasing the
frequency of the excitation sin
%wave in order to obtain a bode plot, the
random noise excites the system
%with a wave with many frequencies.  The
output signal can then be analyzed
%using fast fourier transforms in order to
obtain the frequency response.

%----------------------Random Noise in
Position 1----------------------------
load rn1

frequency_rn1 = o2i1x;
o2i1rn1 = o2i1;

voltage_magnitude_rn1 = abs(o2i1rn1);

decibels_voltage_rn1 =
20*log10(voltage_magnitude_rn1);

current_magnitude_rn1_dB =
decibels_voltage_rn1 +
shift_factor_decibels;

real_rn1 = real(o2i1rn1);
imaginary_rn1 = imag(o2i1rn1);

phase_rn1_radians =
atan(imaginary_rn1./real_rn1);
phase_rn1_degrees =
(phase_rn1_radians/(2*pi))*360;


%----------------------Random Noise in
Position 2----------------------------
load rn2

frequency_rn2 = o2i1x;
o2i1rn2 = o2i1;

voltage_magnitude_rn2 = abs(o2i1rn2);

decibels_voltage_rn2 =
20*log10(voltage_magnitude_rn2);

current_magnitude_rn2_dB =
decibels_voltage_rn2 +
shift_factor_decibels;

real_rn2 = real(o2i1rn2);
imaginary_rn2 = imag(o2i1rn2);

phase_rn2_radians =
atan(imaginary_rn2./real_rn2);
phase_rn2_degrees =
(phase_rn2_radians/(2*pi))*360;


%----------------------Random Noise in
Position 3----------------------------
load rn3

frequency_rn3 = o2i1x;
o2i1rn3 = o2i1;

voltage_magnitude_rn3 = abs(o2i1rn3);

decibels_voltage_rn3 =
20*log10(voltage_magnitude_rn3);

current_magnitude_rn3_dB =
decibels_voltage_rn3 +
shift_factor_decibels;

real_rn3 = real(o2i1rn3);
imaginary_rn3 = imag(o2i1rn3);

phase_rn3_radians =
atan(imaginary_rn3./real_rn3);
phase_rn3_degrees =
(phase_rn1_radians/(2*pi))*360;

%----------------------Random Noise in
Position 4----------------------------
load rn4

frequency_rn4 = o2i1x;
o2i1rn4 = o2i1;

voltage_magnitude_rn4 = abs(o2i1rn4);

decibels_voltage_rn4 =
20*log10(voltage_magnitude_rn4);

current_magnitude_rn4_dB =
decibels_voltage_rn4 +
shift_factor_decibels;

real_rn4 = real(o2i1rn4);
imaginary_rn4 = imag(o2i1rn4);

phase_rn4_radians =
atan(imaginary_rn4./real_rn4);
phase_rn4_degrees =
(phase_rn4_radians/(2*pi))*360;

%plot the bode plot
figure
subplot(2,1,1)
semilogx(frequency_rn1,current_magnitude_rn1
_dB,frequency_rn2,current_magnitude_rn2_dB,f
requency_rn3,current_magnitude_rn3_dB,freque
ncy_rn4,current_magnitude_rn4_dB)
title('Bode Plot of DC Motor I(s)/V_a(s)
Using Random Noise as Excitation')
axis([10 1000 -25 -5])
ylabel('Magnitude (dB)')
legend('Position 1','Position 2','Position
3','Position 4',3)

subplot(2,1,2)
semilogx(frequency_rn1,phase_rn1_degrees,fre
quency_rn2,phase_rn2_degrees,frequency_rn3,p
hase_rn3_degrees,frequency_rn4,phase_rn4_deg
rees)
axis([10 1000 -80 0])
ylabel('Phase (deg)')
xlabel('Frequency (rad/s)')
legend('Position 1','Position 2','Position
3','Position 4',3)

%We now need to now calculate the break
frequency.  By analyzing the plot,
%it appears  that position 3 yields the best
results, therefore we will use
%this as the reference.
DC_gain_rn = current_magnitude_rn3_dB(1);

%We would like to find out when the signal
drops by 3 dB
critical_dB_rn = DC_gain_rn-3;

for
counter=1:length(current_magnitude_rn3_dB)
    if current_magnitude_rn3_dB(counter) <
critical_dB_rn
        break
    else
    end
end

%Now find the frquency where the magnitude
drops by 3 dB
break_frequency_rn = frequency_rn3(counter);

%Using this, calculate La
La_rn = (Rm+R)/break_frequency_rn;
sprintf('omega break = %2.3f (break
frequency from random noise excitation)\nLa
= %2.5f (motor inductance using random noise
bode plots)',break_frequency_rn,La_rn)

toc
```

## loaddata.m

```
clc
disp(' ');

%Run 1
runlog = '01 - P: 1 - D: 60';

disp('Loading Run 1');
load 'run_1.mat';
t = time;
theta = computer_output(:,1);
Va = computer_output(:,2);
omega = computer_output(:,3);
omega_ref = computer_output(:,4);
P = computer_output(:,5);
I = computer_output(:,6);
omega_err = computer_output(:,7);

for r = 2:13
    disp(['Loading Run ' num2str(r)]);
    load(['run_' num2str(r) '.mat']);
    n = length(time) - length(t);
    if n > 0
        t = [t; zeros(n,size(t,2))];
        theta = [theta;
zeros(n,size(theta,2))];
        Va = [Va; zeros(n,size(Va,2))];
        omega = [omega;
zeros(n,size(omega,2))];
        omega_ref = [omega_ref;
zeros(n,size(omega,2))];
        P = [P; zeros(n,size(P,2))];
        I = [I; zeros(n,size(I,2))];
        omega_err = [omega_err;
zeros(n,size(omega_err,2))];
    elseif n < 0
        n = abs(n);
        computer_output = [computer_output;
zeros(n,size(computer_output,2))];
        time = [time;
zeros(n,size(time,2))];
    end
    t = [t time];
    theta = [theta computer_output(:,1)];
          Va = [Va computer_output(:,2)];
          omega = [omega
computer_output(:,3)];
          omega_ref = [omega_ref
computer_output(:,4)];
          P = [P computer_output(:,5)];
          I = [I computer_output(:,6)];
          omega_err = [omega_err
computer_output(:,7)];
end
theta_ref = zeros(size(theta));
for r = 14:20
    disp(['Loading Run ' num2str(r)]);
    load(['run_' num2str(r) '.mat']);
    n = length(time) - length(t);
    if n > 0
        t = [t; zeros(n,size(t,2))];
        theta = [theta;
zeros(n,size(theta,2))];
        Va = [Va; zeros(n,size(Va,2))];
        omega = [omega;
zeros(n,size(omega,2))];
        omega_ref = [omega_ref;
zeros(n,size(omega,2))];
        P = [P; zeros(n,size(P,2))];
        I = [I; zeros(n,size(I,2))];
        omega_err = [omega_err;
zeros(n,size(omega_err,2))];
        theta_ref = [theta_ref;
zeros(n,size(theta_ref,2))];
    elseif n < 0
        n = abs(n);
        computer_output = [computer_output;
zeros(n,size(computer_output,2))];
        time = [time;
zeros(n,size(time,2))];
    end
    t = [t time];
    theta = [theta computer_output(:,1)];
          Va = [Va computer_output(:,2)];
          omega = [omega
computer_output(:,3)];
          omega_ref = [omega_ref
computer_output(:,4)];
          P = [P computer_output(:,5)];
          I = [I computer_output(:,6)];
          omega_err = [omega_err
computer_output(:,7)];
    theta_ref = [theta_ref
computer_output(:,8)];
end

%Run 2
runlog = strvcat(runlog,'02 - P: 1 - D:
50');

%Run 3
runlog = strvcat(runlog,'03 - P: 1 - D:
40');

%Run 4
runlog = strvcat(runlog,'04 - P: 1 - D:
70');

%Run 5
runlog = strvcat(runlog,'05 - P: 1 - D:
80');

%Run 6
runlog = strvcat(runlog,'06 - P: .25 - D:
60');

%Run 7
runlog = strvcat(runlog,'07 - P: .5 - D:
60');

%Run 8
runlog = strvcat(runlog,'08 - P: .75 - D:
60');

%Run 9
runlog = strvcat(runlog,'09 - : 3 - D: 60');

%Run 10
runlog = strvcat(runlog,'10 - : .1 - D:
60');

%Run 11
runlog = strvcat(runlog,'11 - P: .22 - D:
60');

%Run 12
runlog = strvcat(runlog,'12 - P: .22 - D: 60
- EC:100%');

%Run 13
runlog = strvcat(runlog,'13 - KP: 1 - KI: 1
- D: 30 - PF: 30');

%Position Controller
%Run 14
runlog = strvcat(runlog,'14 - KP: 1');

%Run 15
runlog = strvcat(runlog,'15 - KP: 1 - ECD:
100%');

%Run 16
runlog = strvcat(runlog,'16 - KP: 5 - ECD:
100%');

%Run 17
runlog = strvcat(runlog,'17 - KP: 5');

%Run 18
runlog = strvcat(runlog,'18 - KP: 15 - PF:
10');

%Run 19
runlog = strvcat(runlog,'19 - KP: 15 - PF:
20');

%Run 20
runlog = strvcat(runlog,'20 - KP: 7 - PF:
20');
```