Lab Assignment 3: DC Motor Servo System

Team members:
Shiva Ghose, @gshiva
John Peterson, @jrpeters
Peter Turpel, @pturpel
Chan-Rong Lin, @pmelin

Teamwork Participation Pledge :: Team 1

I attest that I have made a fair and equitable contribution to this lab and submitted assignment.

My signature also indicates that I have followed the University of Michigan Honor Code, while working on this lab and assignment.

I accept my responsibility to look after all of the equipment assigned to me and my team, and that I have read and understood the X50 Lab Rules.

Name	Email	Signature
Shiva Ghose	gshiva@umich.edu	
John Peterson	jrpeters@umich.edu	
Peter Turpel	pturpel@umich.edu	
Chan-Rong Lin	pmelin@umich.edu	

1.

a.

U1

U1 serves as a difference circuit between ^{-}Ref and ^{+}Ref with a variable overall gain dictated by the potentiometer.

$$V_{out} = \left(\frac{R_6}{R_5 + R_6}\right)\left(\left(1 + \frac{R_2}{R_1}\right)\left(\frac{R_4}{R_3 + R_4}\right)(^{-}Ref) - \left(\frac{R_2}{R_1}\right)(^{+}Ref)\right)$$

$$V_{out} = \left(\frac{R_6}{R_5 + R_6}\right)\left(\left(1 + \frac{20}{50}\right)\left(\frac{20}{50 + 20}\right)(^{-}Ref) - \left(\frac{20}{50}\right)(^{+}Ref)\right)$$

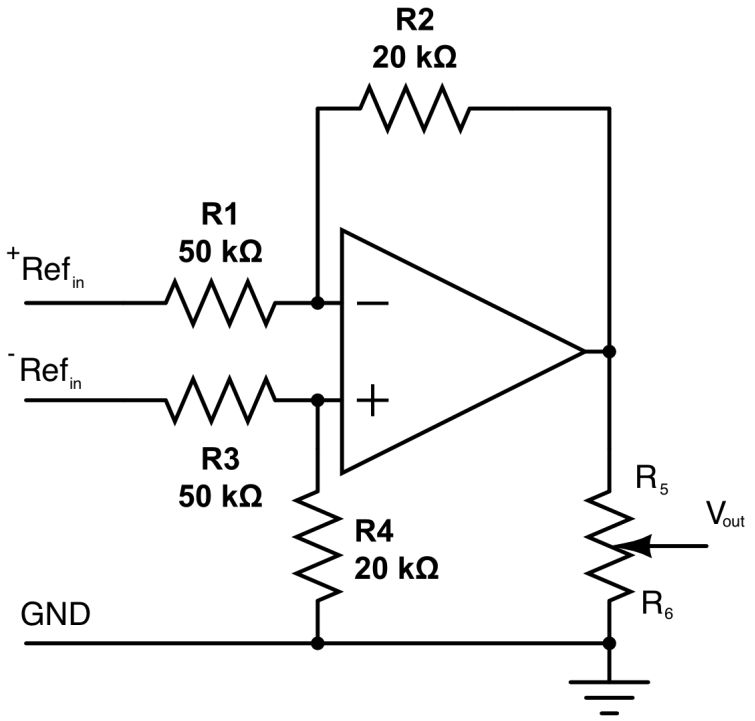

Figure 1: Unit 1 within AMC servo-amplifier

U3

U3 serves as an inverting amplifier which can be tuned by the potentiometer.

$$U3_{out} = -\left(\frac{R_3 + R_4}{R_4}\right)\left(\frac{R_2}{R_1}\right) Ref_{gain}$$

$$U3_{out} = -\left(\frac{R_3 + R_4}{R_4}\right)\left(\frac{10}{5}\right) Ref_{gain}$$

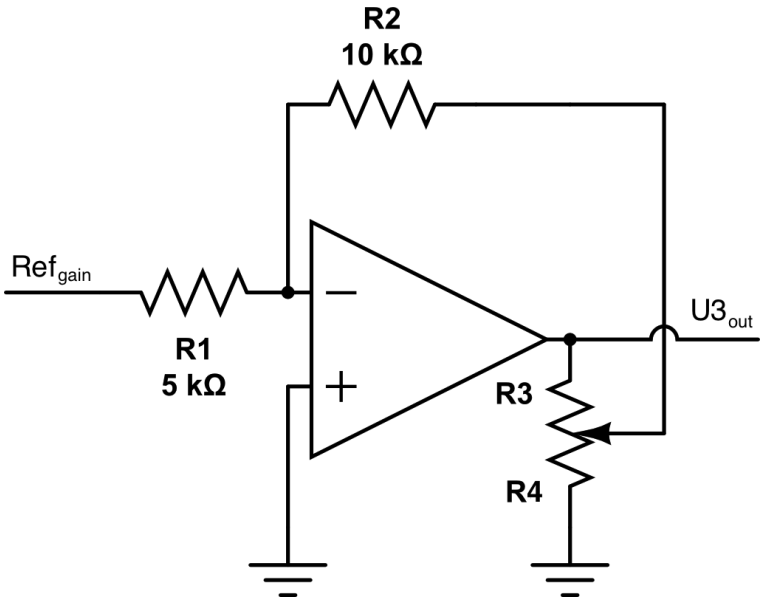

Figure 2: Unit 3 within AMC servo-amplifier

U4

U4 serves as an inverting integrator with a proportional gain when switch 3 is open, and a simple inverting amplifier with the switch closed.

$$U4_{out} = -\left(\frac{1}{R_1}\right)\left(R_2 + \frac{C_1}{s}\right)U3_{out}$$

$$U4_{out} = -\left(\frac{1}{500}\right)\left(500 + \frac{0.01}{s}\right)U3_{out}$$

b.

This resistor serves to limit the current that flows through this section of the circuit to the ground because the inductor which is. A larger resistance value decreases this current and waste less power.

c.

The motor data sheet specifies a max winding temperature of 155°C, above which the motor can become irreparably damaged, and a thermal impedance of $11.2\,^{\circ}C/W$. Assuming a room temperature of 15°C, the motor's temperature can only increase by 130°C before overheating. With these values and the motor's internal resistance of $1.85\,\Omega$, we can solve for the maximum continuous current:

$$130 = 11.2 * i^2_{continuous} * 1.85$$

$$i_{continuous} = 2.5048\,A$$

With this current and the given motor constant we can then calculate the maximum continuous torque:

$$T_m = k_t * i_{continuous} = 4.24 \times 10^{-2} * 2.5048 = 10.6 \times 10^{-2}\,Nm$$

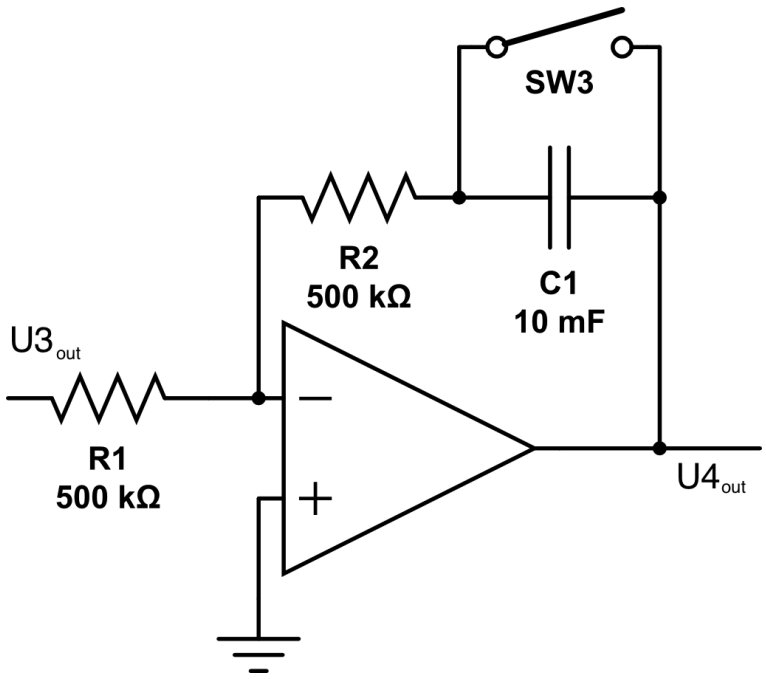

Figure 3: Unit 4 within AMC servo-amplifier

This value is greater than the torque value given in the data sheet, implying that our calculated current value is less conservative than the ratings in the data sheet. Based on the given rating for max continuous torque, $8.1 \times 10^{-2}\, Nm$, the max continuous current would be $1.92A$, but at that value the temperature rise would be 75°C above ambient. This means that at the max torque given by the data sheet, the motor would be well under the temperature limit. It's possible that the given value is either based on a safety factor to prevent overheating, or there are other modes of breakdown that can result from higher currents. If the lower limit is based on temperature, then we should be able to use slightly higher currents given that we are running for short periods.

The motor's data sheet lists a peak current rating of 13.0 A and a stall torque of 0.54 $N \cdot m$. With this torque value and the given torque constant ($4.24 \times 10^{-2 (N \cdot m)}/_{A}$), the peak current should be:

$$i_{peak} = \frac{T_{stall}}{k_t} = \frac{0.54}{4.24 \times 10^{-2}} = 12.74\, A$$

This is less than the given rating for peak current. At this peak current limit, the motor dissipates approximately 300 W. If the system were allowed to run at this current, and assuming that the motor does not fail in some way, it would reach a steady state temperature of:

$$P = i_{peak}^2 * R_m = 12.74^2 * 1.85 = 300.269\, W$$

$$\Delta Temp = P * R_t = 300.269 * 11.2 = 3363.0128\, ^\circ C\ (above\ ambient)$$

Clearly the system could not physical reach this temperature without failing, but it illustrates the reason that peak current can only be sustained for short times. With the given thermal time constant of 13.8 min, we can plot the temperature rise of the motor, as shown in figure 4, and estimate that it would overheat after approximately 32.4 seconds at peak current.

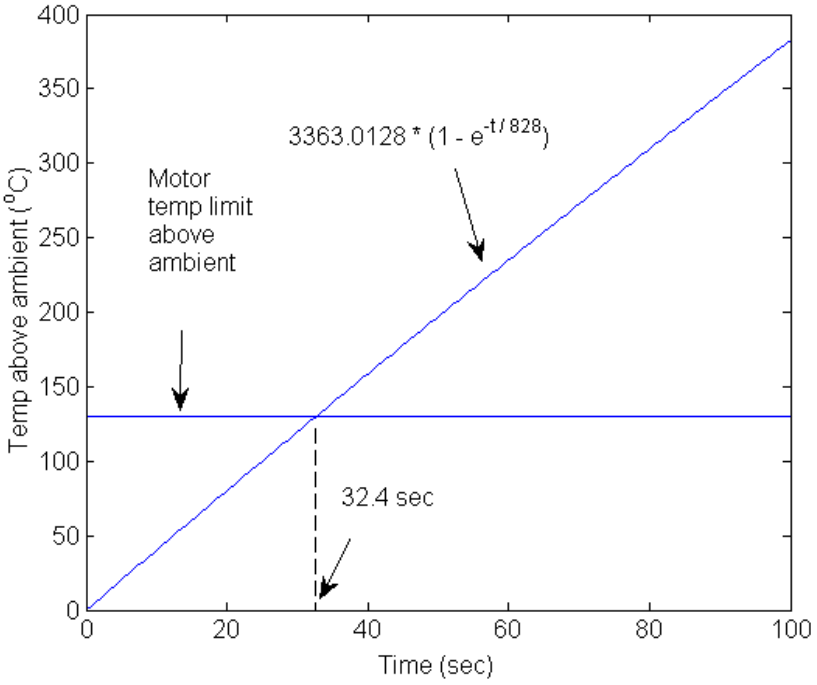

Figure 4: Motor temperature over time at peak current

d.

The 12A8E servo amplifier is capable of outputting maximum continuous and peak currents of $\pm 6A$ and $\pm 12A$, respectively (from the data sheet). However, the servo amplifier includes a potentiometer that can be used to reduce both the continuous and peak currents while maintaining a 1:2 relationship between them. The lab instructions specify that the potentiometer should be adjusted so that the current limits will be $\pm 3A$ and $\pm 6A$. The continuous current limit can be further reduced, without affecting the peak current, by connecting a current limiting resistor between pins P1-10 and P1-2 on the servo-amp. That was not done in this lab. Assuming the adjustment on the current limiting potentiometer was done correctly, the servo-amp should be capable of sending $\pm 3A$ continuously and $\pm 6A$ intermittently to the motor. The continuous current limit from the servo-amp is greater than the continuous current rating of the motor, so we can't rely on the servo-amp to prevent the motor from drawing too much current.

The data sheet for the servo-amp specifies a peak current of $\pm 12A$ for a maximum of 2 seconds. From information on the Advanced Motion Controls website, this can only be achieved if the current switches across its entire range (-12 A to +12 A, and vice-versa). Any change of smaller magnitude will not be maintained as long (i.e. switching from 0 A to +12 A will only be held for 1 second before the current begins falling). The primary concern with maintaining high currents is overheating, and the servo-amp has a much lower temperature limit than the motor (65 °C vs 155 °C), as well as a much smaller thermal time constant than the motor. This is why the peak current can not be maintained for long, the maximum temperature would be exceeded very quickly and the servo-amp's internal safeties would shut it off. As an example, in the thermal model for the motor at peak current (which is lower than the 6 A rating of the servo-amp), 65 °C was reached in only 9.8 seconds. Of course, the servo-amp has different thermal properties than the motor, so that specific value is not accurate. However, the basic idea is valid: the servo-amp is capable of passing a lot of current, and has a lower temperature limit than the motor, so there is a danger of it overheating if

high currents are maintained. Luckily, the servo-amp has built in safeties, such as a thermal shut-off.

e.

In the above sections, we showed that the motor has a continuous and peak current limits of 2.5 A and 12.74 A, and the servo-amp has current limits of 3 A and 6 A, respectively. Within our LabView controller we chose to implement saturation limits of ± 6 A so that we could take advantage of the peak current capability of the servo-amp and motor. We considered setting a saturation limit based on the motor's continuous current limit of 2.5 A, but felt that might have a negative effect on the transient response of the motor. Also, we knew that the servo-amp has built in mechanisms to prevent the current from remaining at the peak value - it automatically lowers it after at most two seconds. Additionally, as we were tuning our LabView controller, we monitored a waveform graph of the attempted output of the controller. This allowed us to adjust our gain values not just for system stability, but also to keep the attempted output within safe limits. Given more resources, we could have implemented more safety measures. For example, we could implement a slow-blow fuse rated a little under the continuous current limit of the motor. This would protect the system from high currents for prolonged times, but still allow for short bursts of higher current during transient conditions.

2.

a.

Using expressions for the back-emf of the motor along with a simple electrical model of the motor as a resistance in series with an inductance we can model the motor-driver circuit. This derivation neglects the effects of changing current direction in the rotor windings during operation. We can combine this electrical model with a simple lumped parameter model of physical dynamics of the motor to derive an overall transfer function.

$$V_s - e = iR + L\left(\frac{di}{dt}\right) \qquad e = K_b\dot{\theta}$$

$$\Sigma_\tau = \tau_{motor} - \tau_{damping} + \tau_{coulomb} = J\ddot{\theta}$$

$$\tau_{motor} = K_T i \qquad \tau_{damping} = b\dot{\theta}$$

$$\tau_{coulomb} = \begin{cases} -\tau_{motor} & \text{if } \dot{\theta} = 0 \text{ and } |\tau_{motor}| < \tau_{friction} \\ -sgn(\tau_{motor}) \cdot \tau_{friction} & \text{if } \dot{\theta} = 0 \text{ and } |\tau_{motor}| > \tau_{friction} \\ -sgn(\dot{\theta}) \cdot \tau_{friction} & \text{if } \dot{\theta} \neq 0 \end{cases}$$

Where: K_t is the motor torque constant; J is the rotor inertia; b is the viscous damping on the rotor; K_b is the back-emf constant of the motor; R is the equivalent resistance of the motor; L is the effective motor inductance; i is the current flowing through the motor; and $\tau_{coulomb}$ is the force of coulomb friction on the rotor.

Neglecting coulomb friction:

$$V_s(s) - s\,K_b\,\theta(s) = I(s)(R + sL) \qquad \text{and} \qquad K_t\,I(s) = \theta(s)(Js^2 + bs)$$

$$\frac{\theta(s)}{V(s)} = \frac{K_T}{s\left[(Js + b)(Ls + R) + K_bK_T\right]}$$

The transfer function relating voltage and position is not particularly convenient and a much simpler one can be constructed relating motor current, $I(s)$, and angular position, $\theta(s)$.

$$K_t\,I(s) = \theta(s)(Js^2 + bs)$$

$$\frac{\theta(s)}{I(s)} = \frac{K_T}{(Js + b)s}$$

This model relating angular position with motor current requires the moment of inertia of the rotor, J, the viscous damping coefficient, b, and the torque constant K_T. Note that coulomb friction has been neglected. The driver operating in current mode gives us control over the torque exerted by the motor as long as the we avoid saturating our supply voltages of $\pm 24V$. This saturation occurs because without an external load on the motor, the rotor is free to spin up to a high angular velocity only limited by viscous damping and coulomb friction. At these high velocities, the back emf across the motor increases, meaning that the driver must supply higher voltages across the motor to pass the same current through it.

b.

The torque of friction and the viscous damping can be jointly derived in an experiment where for a given constant input voltage to the driver, V_{driver}, we read off the steady state angular velocity, $\dot{\theta}_{ss}$. At a constant velocity the following equation holds giving a relationship between τ_{motor}, b, and $\tau_{coulomb}$.

$$\Sigma_\tau = \tau_{motor} - b\dot{\theta} + \tau_{coulomb} = J\ddot{\theta} \qquad \ddot{\theta} = 0$$

$$\tau_{motor} = K_T I_m = b\dot{\theta}_{ss} - \tau_{coulomb}$$

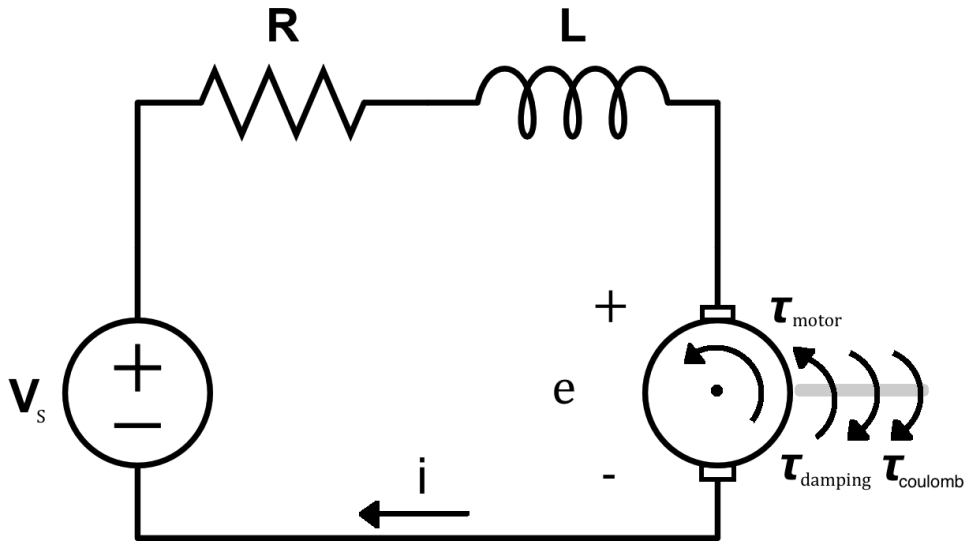

Figure 5: DC motor model. Note that the sign convention here is not the one used in the remainder of the report, but is presented to concisely reflect the nature of the system.

Assuming that we have the motor torque constant K_t, we can write a linear equation that relates steady state angular velocity $\theta_{steadystate}$ to motor current which is assumed to be directly proportional to V_{driver} for suitably small values of V_{driver}.

$$V_{driver} = I_m \qquad V_{driver} = \left(\frac{b}{K_T}\right)\dot{\theta}_{ss} - \frac{\tau_{coulomb}}{K_T}$$

From our plot in figure 6 we can obtain $({}^{b}/_{K_t})$ as the slope of each of the linear segments and ${}^{\tau_{coulomb}}/_{K_T}$ as the y intercepts shown in table 1. We can separately determine the force of static friction $\tau_{static\,coulomb}$ by measuring the smallest input voltage to the driver, and from this how much torque, is required to start the motor from rest.

$$\Sigma_\tau = \tau_{motor} - b\dot{\theta} + \tau_{coulomb} = J\ddot{\theta} \qquad \dot{\theta} = \ddot{\theta} = 0$$
$$V_{driver} = -\frac{\tau_{coulomb}}{K_T}$$

	Slope $\left(\frac{V*s}{rad}\right)$	y-intercept (V)
Forwards	5.4289e-04	0.46374
Reverse	6.72805e-04	-0.60717

Table 1: Extracted slopes and y-intercepts from current speed curve

If we assume the motor constant on the data-sheet of $0.0424\,{}^{(N\cdot m)}/_{A}$ then we can arrive at the following estimates for b and $\tau_{coulomb}$, shown in table 2 compared against the values obtained from the data sheet. Our values are within an order of magnitude of the provided values which is not close enough to make any particular claims besides that this motor seems to have worse performance than advertised.

To compute the torque constant, K_T we can first compute the back-emf constant, K_b, which we know to be numerically equal to K_T in SI units. To calculate K_b we can measure the internal resistance of the motor, R, using an ohm-meter. Then running the system at a constant current, we can measure the source

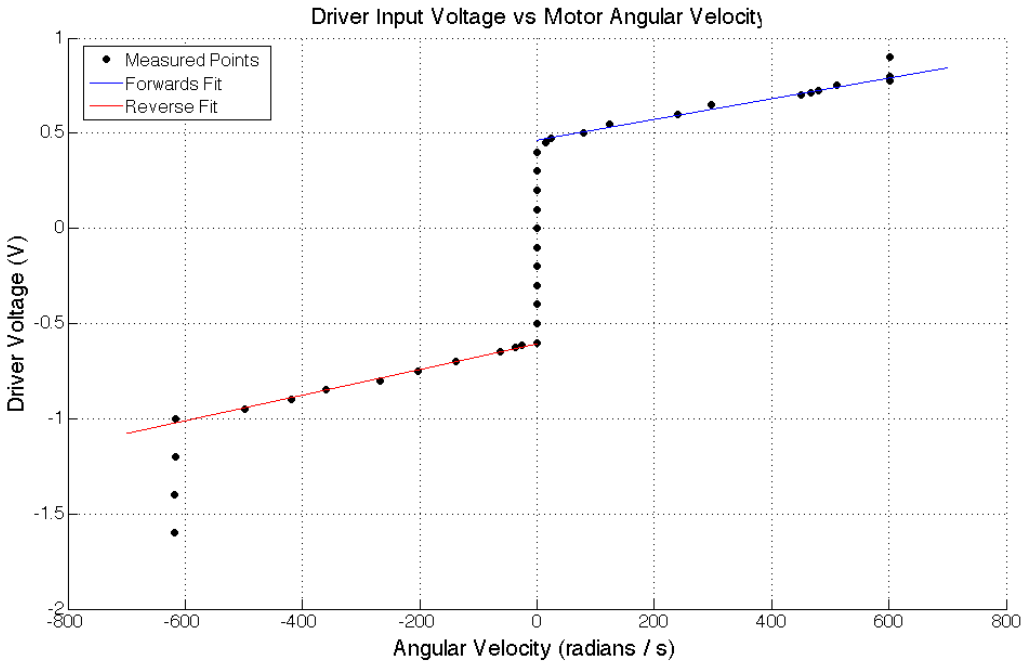

Figure 6: Relationship between angular velocity and steady state voltage applied to motor driver. Notice the saturation at $\pm 600\, {}^{radians}/_{s}$

	Data Sheet Values	Average Estimated Values
$b\ (N \cdot s)$	$3.7 \cdot 10^{-6}$	$2.5773 \cdot 10^{-5}$
$\tau_{coulomb}\ (N \cdot m)$	$5.6 \cdot 10^{-3}$	0.044978

Table 2: Comparison of viscous damping and friction torque estimates with values obtained from the data sheet

voltage, V_s that the driver outputs to maintain that current to obtain e. We can use the encoder to measure the angular velocity and use that value with e to solve for K_b:

$$\Sigma_V = V_s + V_R + V_L e = V_s + IR + e = 0$$
$$K_b = \frac{e}{\dot{\theta}}$$

We can determine the moment of inertia of the rotor, J by starting from rest and sending the motor a known torque while recording the position as a function of time stopping when the system has reached a constant velocity. Then by computing the 1^{st} and 2^{nd} order derivatives at each time step we can assemble a linear system to allow us to jointly solve for J, b, and $\tau_{coulomb}$.

$$A_t = \begin{bmatrix} \ddot{\theta}_t & \dot{\theta}_t & -1 \end{bmatrix} \qquad x = \begin{bmatrix} J \\ b \\ \tau_{coulomb} \end{bmatrix} \qquad z_t = \tau_{motor}$$
$$Ax = z$$

Rather than attempt the method above, we instead used MATLAB's numerical differential equation solving to simulate the trajectory of the system as a function of time, driver current, I_t and our unknown system

parameters J, b, K_t, and $\tau_{coulomb}$. By then comparing the predicted angular positions with the measured angular positions over time, we can construct a non-linear regression to estimate the unknown parameters. We collected a data set where the driver was given a sinusoidal voltage, and we assume that motor current, I_m, is identical to this driver voltage and recorded the motor motion that this input caused.

Table 3 below shows the parameters for our equation of motion obtained from the data sheet. The results, in figure 7 shows the estimated trajectory using these parameter values. It is clear that these parameter values are quite far off. Attempting to perform nonlinear optimization on our model parameters proved to be difficult, so we resorted to parameter sweeping to to identify appropriate starting points for regression. Figure 8 shows one of the best results discovered by this parameter sweep. In this case the error has been reduced, but the fit still appear inappropriate.

	Data Sheet Parameters
$J\ \left(\frac{N \cdot s^2}{m}\right)$	$8.5 \cdot 10^{-6}$
$b\ (N \cdot s)$	$3.7 \cdot 10^{-6}$
$\tau_{coulomb}\ (N \cdot m)$	$5.6 \cdot 10^{-3}$
$K_T\ \left(\frac{N \cdot m}{A}\right)$	$4.24 \cdot 10^{-2}$

Table 3: Original model parameter values obtained from the data sheet.

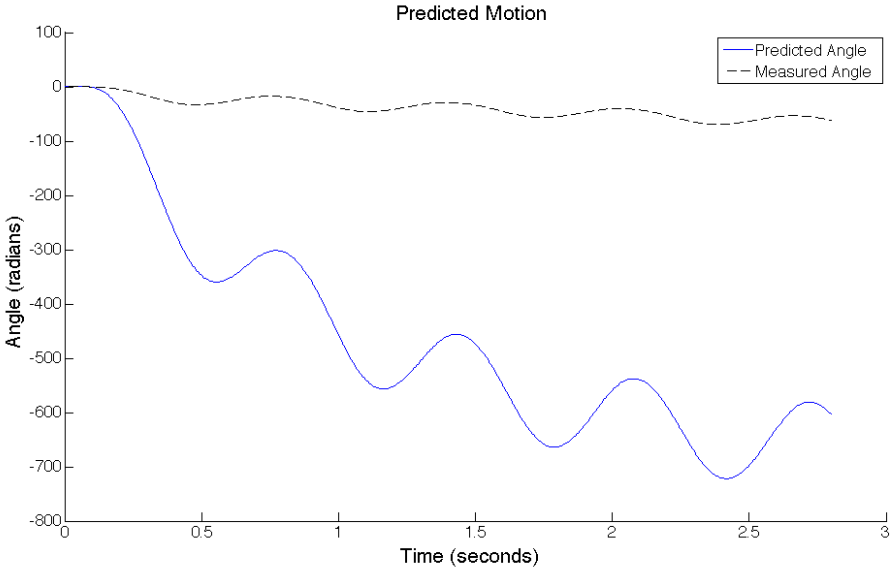

Figure 7: Predicted behavior vs Actual behavior using initial parameter values noted in table 3.

Manual parameter guessing revealed that it was not possible to obtain the measured curve using the model, repeated below.

$$\Sigma_{tau} = K_T I_m - b\dot{\theta} + \tau_{coulomb} = J\ddot{\theta}$$

Clearly there were some effects that this model neglected. Our previous experiment to attempt to determine b and $\tau_{coulomb}$ showed that these parameters are likely not the same in each direction, so we augmented the model to allow b and $\tau_{coulomb}$ to take on different values in the forwards direction and reverse directions yielding the following model:

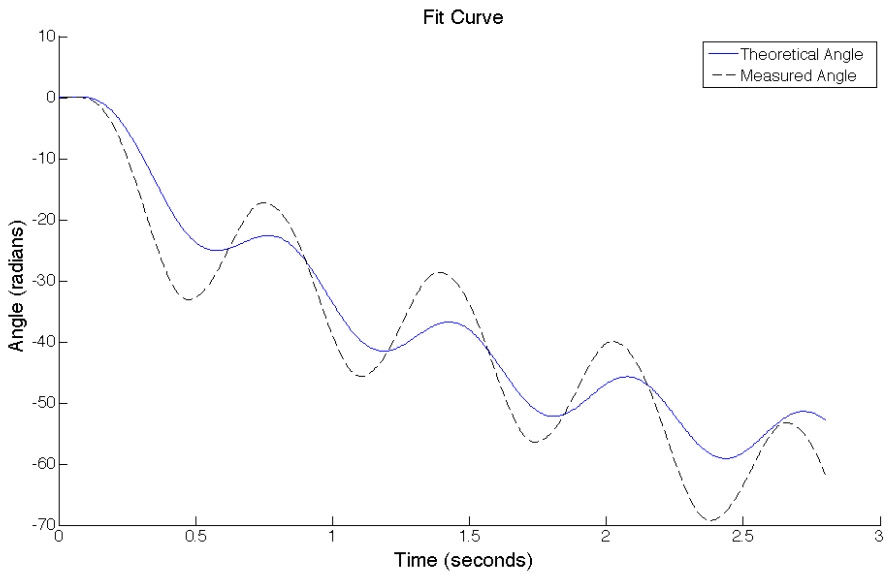

Figure 8: Fit by non-linear optimization and parameter sweeping: $J = 6.06724 \cdot 10^{-5}$ $B = 4.05562 \cdot 10^{-5}$ $F_f = 1.45312 \cdot 10^{-4}$ $K_t = 0.019020$

$$\Sigma_{tau} = \tau_{motor} - b\dot{\theta} + \tau_{coulomb} = J\ddot{\theta}$$

$$\tau_{coulomb} = \begin{cases} -\tau_{forwards} & \text{if } \dot{\theta} > 0 \\ +\tau_{reverse} & \text{if } \dot{\theta} < 0 \\ -\tau_{forwards} & \text{if } \dot{\theta} = 0 \text{ and } \tau_{motor} > \tau_{forwards} \\ -\tau_{motor} & \text{if } \dot{\theta} = 0 \text{ and } 0 <= \tau_{motor} <= \tau_{forwards} \\ +\tau_{reverse} & \text{if } \dot{\theta} = 0 \text{ and } \tau_{motor} < -\tau_{reverse} \\ +\tau_{motor} & \text{if } \dot{\theta} = 0 \text{ and } 0 >= \tau_{motor} >= -\tau_{reverse} \end{cases}$$

$$b = \begin{cases} b_{forwards} & \text{if } \dot{\theta} > 0 \\ b_{reverse} & \text{if } \dot{\theta} < 0 \end{cases}$$

With this improved model and some intelligent guessing for new parameter values, we arrived at an excellent fit shown in figure 9. The most notable feature of these values is the large difference between the torque of friction in the forwards and reverse directions.

	Data-Sheet Values	Fit Values
$J\ \left(\frac{N \cdot s^2}{m}\right)$	$8.5 \cdot 10^{-6}$	$3.50514 \cdot 10^{-5}$
$K_t\ \left(\frac{N \cdot m}{A}\right)$	0.0424	0.0314499
$b_{forwards}\ (N \cdot s)$	$3.7 \cdot 10^{-6}$	$1.08586 \cdot 10^{-5}$
$b_{reverse}\ (N \cdot s)$	$3.7 \cdot 10^{-6}$	$4.85930 \cdot 10^{-5}$
$\tau_{forwards}\ (N \cdot m)$	$5.6 \cdot 10^{-3}$	0.0224389
$\tau_{reverse}\ (N \cdot m)$	$5.6 \cdot 10^{-3}$	0.0143096

Table 4: Final Parameter Values vs Original Data Sheet Values

Table 4 shows our final fit values alongside the original values obtained from the data sheet, while the fit curve itself is shown in figure 9. This large difference in values is clearly illustrated by the large difference in behavior between the actual system and our model using the values obtained from the data sheet shown in figure 7.

We can also compare these fit values with values obtained from our earlier experiment, shown together in table 5, after adjusting for the difference in motor torque constant, and notice that our terms are quite similar, however it seems as if this experiment did not accurately estimate the force of friction as the relative values seem to be flipped between the two experiments.

	Predicted Value	Fit Value
$B_{forwards}$	$1.707338 \cdot 10^{-5}$	$1.08586 \cdot 10^{-5}$
$B_{reverse}$	$2.11611 \cdot 10^{-5}$	$2.49301 \cdot 10^{-5}$
$\tau_{forwards}$	0.0145846	0.0224389
$\tau_{reverse}$	0.0190954	0.0143096

Table 5: Damping and friction torques calculated from earlier experiment vs fit values

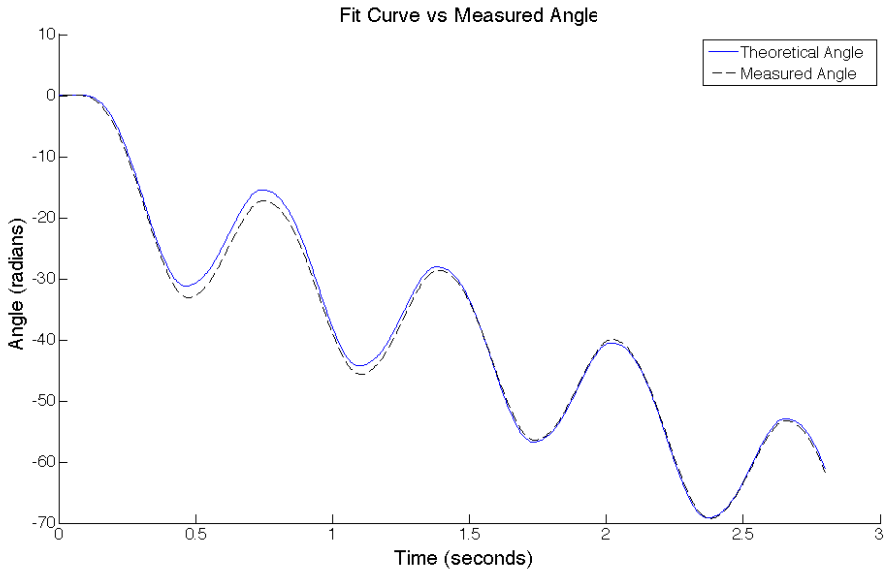

Figure 9: Predicted behavior vs Actual behavior using fit parameter values and enhanced modeling

3.

Simulink Model

Figure 10 shows the overall model of DC motor servo system for both velocity and position control. It consists of four main units:

- DC Motor
- Optical Encoder
- Servo Amplifier
- Controller

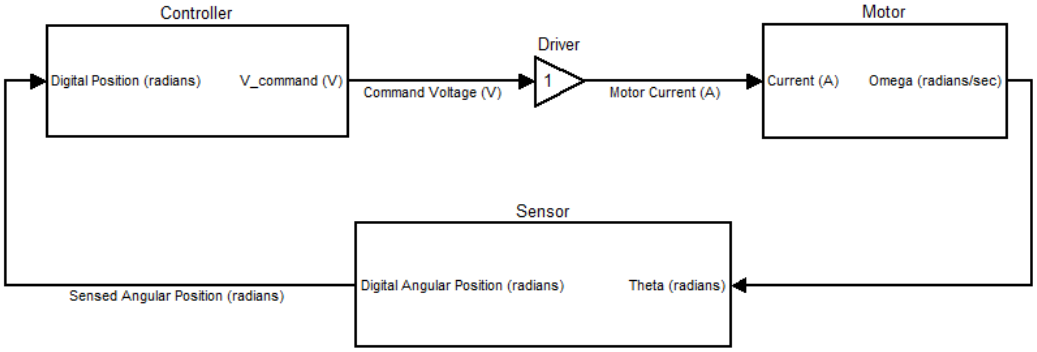

Figure 10: DC motor servo system

DC Motor

The DC motor subsystem, seen in figure 11 worked on the principle that the input torque generated was based on the conversion equation:

$$T = K_t * i$$

Where $K_t = 4.24 \times 10^{-2}$ is the torque constant of the motor. According to the motor datasheet the rotor inertia, J, is 8.5×10^{-6} , and the viscous damping coefficient $B = 3.7 \times 10^{-6}$.

Friction torque was modeled as a simple coulomb viscous model, shown in figure 12. As the viscous friction was already taken into account in the motor transfer function, the model of coulomb friction was created using the following simple function:

$$T_f^{+/-}(\omega) = T_{coul}^{+/-}$$

Where, Coulomb Friction Torque: $T_f = 5.6 \times 10^{-3}$ $[Nm/A]$

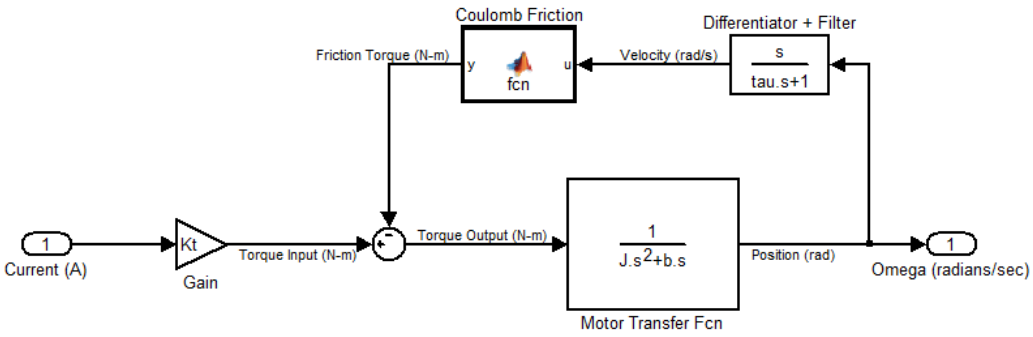

Figure 11: DC Motor Subsystem

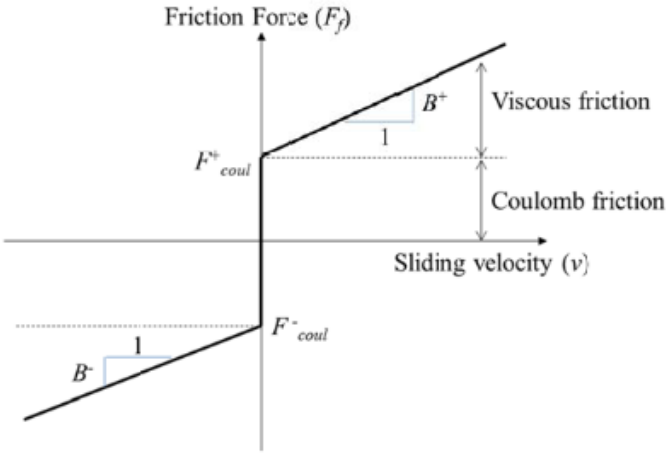

Figure 12: Plot of Friction Force v.s. Velocity

Optical Encoder

The optical encoder (see figure 13) was modeled as a quantizer with the interval size equal to $2\pi/2000$ was used to simulate the effects of the ADC.

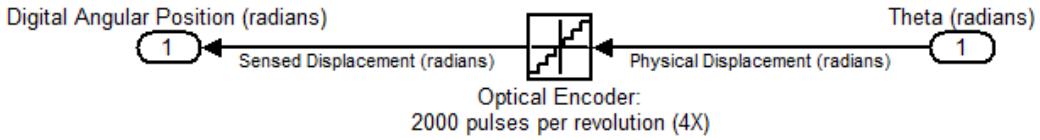

Figure 13: The Optical Encoder Subsystem

Servo Amplifier

The Servo Amplifier was treated as a gain equal to 1 due to its high bandwidth (2.5 kHz). It can be seen in figure 10.

Controller

Two controllers were used to achieve both position and velocity control, they are visible in figures 14 and 15. The saturation limit was set to $\pm 6A$ based on the maximum peak current of the servo-amplifier. An additional concern was that the RMS command voltage needed to be lower than the maximum continuous current of the motor ($1.92A$). We ensured that limiting our output to ± 6 Volts generally allowed for better tracking stability while not often applying more than $1.92A$ for continuous periods of time.

Additionally, the PID block represents the controller while τ (tau) represents the filter time-constant for the transfer function from angular position to velocity. It was treated as a variable coefficient.

J Rotor inertia $[kgm^2]$	B Viscous damping $[Nms]$	K_t Torque constant $[Nm/A]$	T_f Friction torque $[Nm]$	K_{amp} Gain of driver $[A/V]$
8.5×10^{-6}	3.7×10^{-6}	4.24×10^{-2}	5.6×10^{-3}	1

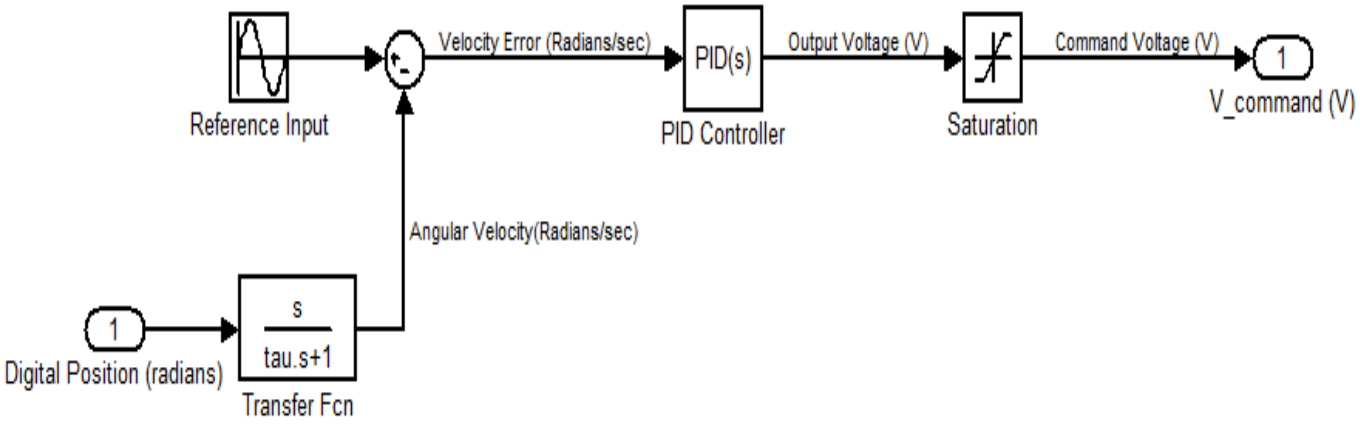

Figure 14: The Velocity Control Subsystem

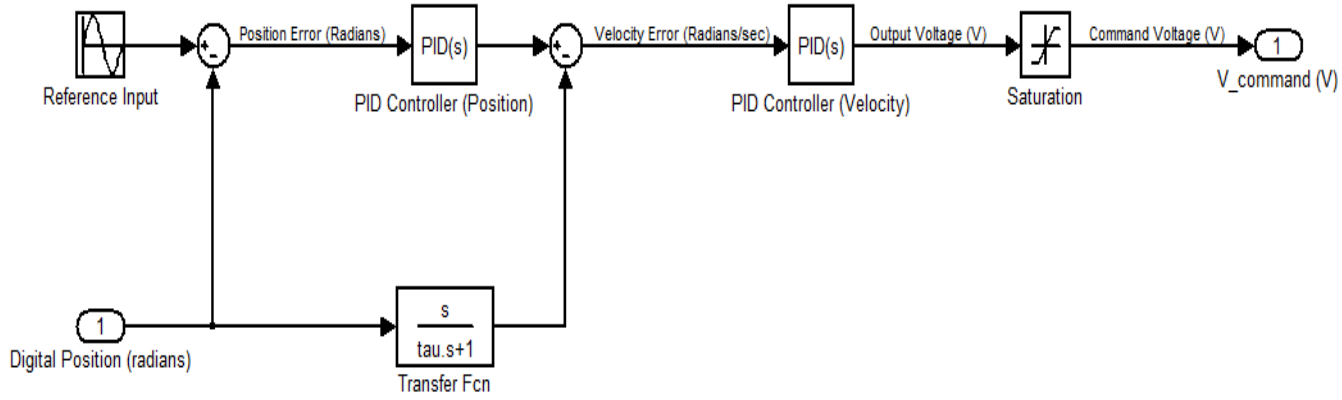

Figure 15: The Position Control Subsystem

Controller Design

The cascaded P/PI controller (see figure 16) is one of the most commonly used controller structures in motion control systems. It is simple and has a systematic tuning procedure. This configuration allowed us to reuse the velocity controller within our position controller.

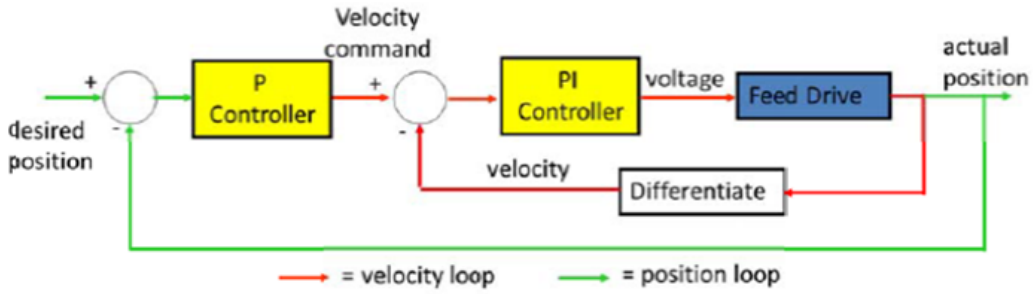

Figure 16: Basic Structure of P/PI Controller

This design consists of two cascaded loops. The inner one is the velocity loop. It is a PI controller which aims to ensure that the velocity command is tracked accurately. It is surrounded by the outer position loop which is a P controller that ensures the desired reference point is achieved. Having both loops makes it easy to deal with each loop one at a time. The first step is to optimize the PI controller, achieving the best performance in the velocity loop, and then tune the P controller to ensure position tracking performance.

Velocity Control System - Dynamics Analysis

Before we began tuning, we needed to know the overall system dynamics, which provide essential information to guide the tuning process. Frequency response was used to analyze the velocity loop based on different values of the time constant, τ. The following figures (17, 18 and 19) show Bode diagrams for the transfer functions of open-loop (G_{OL}), close-loop (G_{CL}), input disturbance rejection (G_{di}), and output disturbance rejection (G_{do}) with $\tau = 0.05, 0.1,$ and 1, respectively.

- G_{OL} (see figure 17): when τ increases, the gain of G_{OL} becomes lower in medium and high frequencies, so does the phase behavior. It can be observed that a larger τ may make the system unstable in lower frequencies since the phase is close to -180 degrees (smaller Phase Margin) and there are still other delays neglected by assumptions, such as the dynamics of servo amplifier.
- G_{CL} (see figure 18): the bandwidth of G_{CL} decreases as τ increases.
- G_{di} (see figure 18): the bandwidth of G_{di} is the same as that of G_{CL} because the system is uncontrolled.
- G_{do} (see figure 19): its quality in higher frequencies becomes better when τ increases.

We find that as τ increases, the quality of output disturbance rejection becomes better but this comes at the expense of and undesirable reduction in bandwidth. In terms of our goals, high bandwidth (up to 5Hz), and good quality of disturbance rejection (60Hz noise signal) need to be achieved at the same time; therefore, the value of τ has to be carefully determined with controllers.

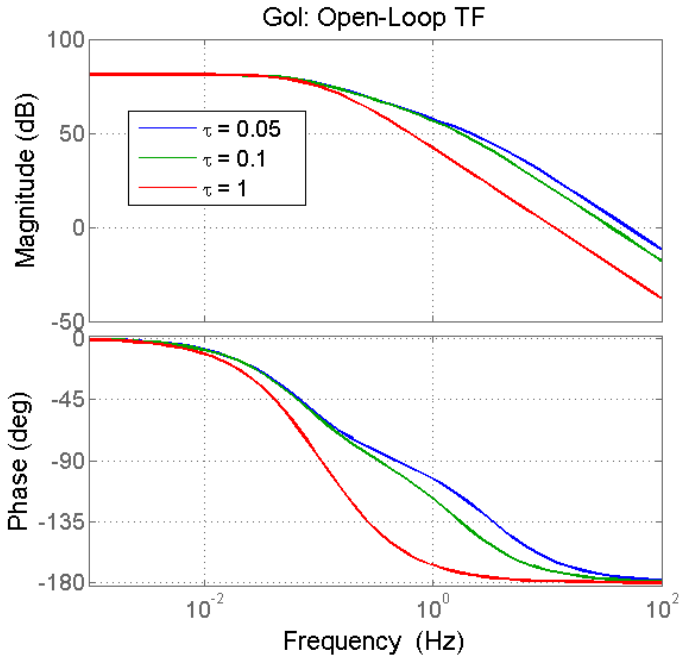

Figure 17: Changes in G_{OL} as τ is increased from 0.05 to 1

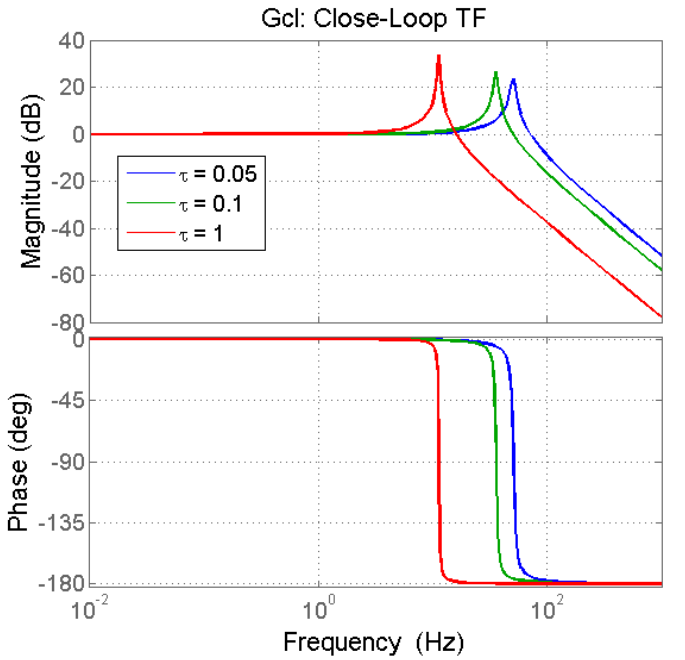

Figure 18: Changes in G_{CL} and G_{di} as τ is increased from 0.05 to 1

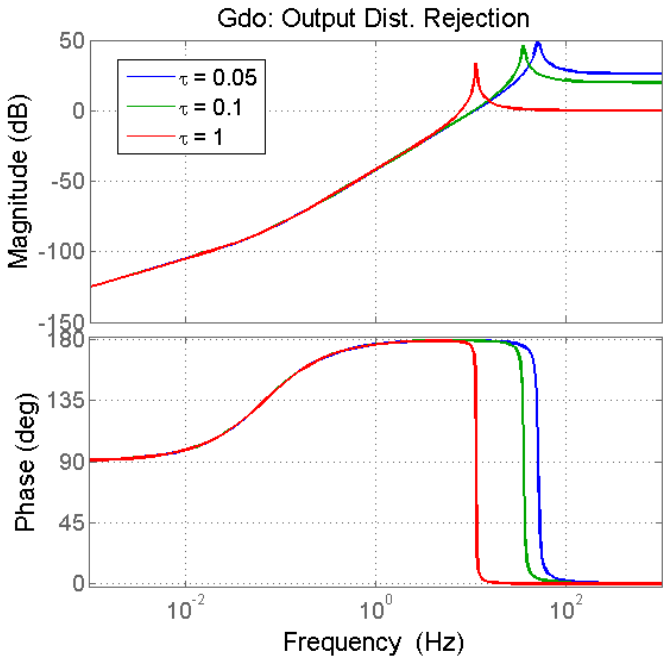

Figure 19: Changes in G_{do} as τ is increased from 0.05 to 1

Velocity Control System Controller Tuning

K_{VP} : Proportional Gain

K_{VI} : Integral Gain

K_{VD} : Derivative Gain

d) Keep steady state error below 2% for a step command of 2π radians/sec

Tuning practices to achieve satisfactory step response:

- Initialize τ as 0.05 to begin with.
- Tune K_{VP} as high as possible because when K_{VP} is increased, the crossover frequency as well as the bandwidth of velocity loop will also increase.
- If K_{VP} exceeds a certain value, it may affect the system stability or saturate either the actuator or the current driver. Considering this issue, K_{VP} was limited by the maximum peak current (6A) and continuous current (1.92A), respectively. Thus, K_{VP} was tuned to 15, ensuring the commanded voltage less than 1.92 A in steady state.
- K_{VI} was not required here since the steady-state error was already less than 2%. The reason may be because the filter seemed to provide the similar function to the integral gain.

e) Provide command tracking: Amplitude should remain within 5% of command up until 5Hz for command amplitude of $\pi/2$ radians/sec

- The tracking error of 5Hz command was up to 10%, so that we tried to change τ to achieve better performance.
- When τ was 0.7, the error was minimum and fluctuated below 2%. This modification also generated better track-ability for the step input.

f) Attenuate a 60Hz noise signal coming in either at the servo-amplifier or the sensor by at least 5 times.

- A noise signal with amplitude 0.1 was used to test disturbance rejection. Over 90% of the input disturbance from the servo amplifier was rejected; while none of the output disturbance, from the sensor was rejected but, even enlarged. Thus increasing was required to get better results.
- When τ was changed from 0.7 to 6, the goal of five-times attenuation was achieved. However, we were unable to maintain sinusoidal command tracking due to the lower bandwidth. This was a trade-off which was expected.
- Considering the limitation of the PI controller, the derivative controller was introduced to add phase lead in our control loop, broadening the bandwidth. By trial and error, finally the three objectives were achieved with the following values of the coefficients:

$$K_{VP} = 25$$

$$K_{VI} = 0$$

$$K_{VD} = 0.2 \quad (N^1 = 100), \text{and}$$

$$\tau = 6.6$$

[1] Causality filter for the differentiator

Velocity Control System Simulation Results

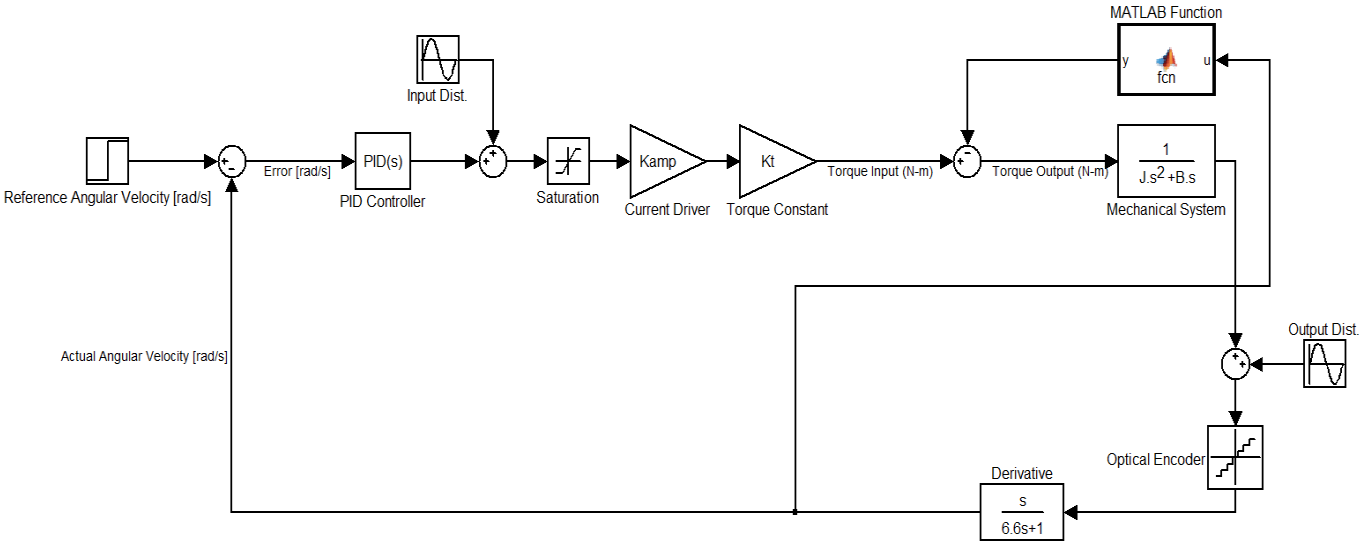

Figure 20: Block Diagram of Velocity Control system

K_{VP}	K_{VI}	K_{VD}	N	τ
25	0	0.2	100	6.6

Reference	Input Amplitude [rad/s]	Output Amplitude [rad/s]	%Error	Commanded Voltage [Max. V]
(d)Step	2π	6.278	0.08%	0.15
(e)Sinusoidal (5Hz)	$\pi/2$	1.641	4.46%	1.99
(f)Input Disturbance	0.1	0.002	2.05%	0.18
Output Disturbance	0.1	0.02	20.0%	0.91

Although we achieved all objectives simultaneously, it took large amounts of controller re-tuning to get to this point.We believe this was due to the following reasons:

- There was a trade-off between sinusoidal reference tracking and high-frequency noise reduction, which was corresponding with what the Bode diagrams showed.
- The continuous current limit was only 2A. The limitation provided much less room for the value of controller gains.

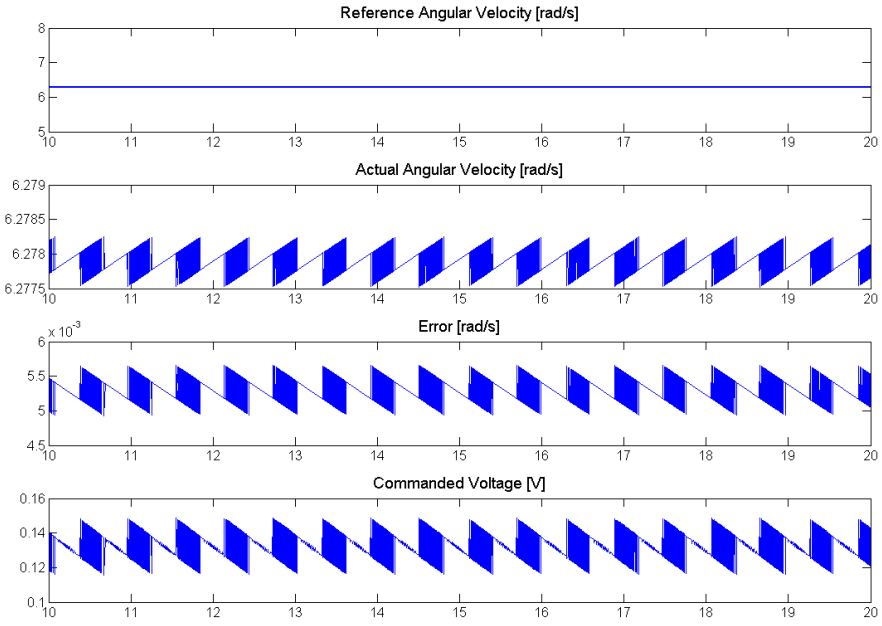

Figure 21: Step Command of 2π radians/sec

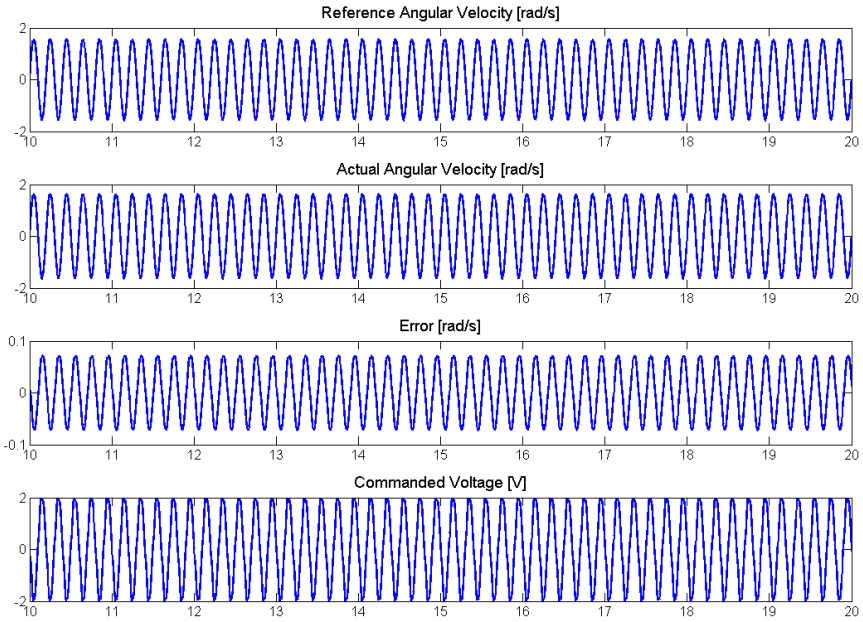

Figure 22: 5Hz Sinusoidal Command of $\pi/2$ radians/sec

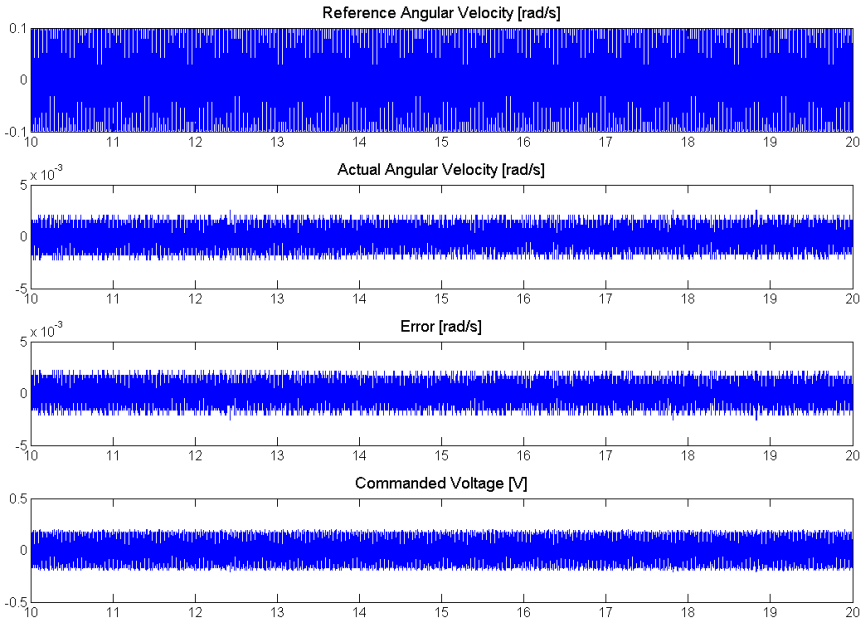

Figure 23: 60Hz Input Disturbance of 0.1 radians/sec

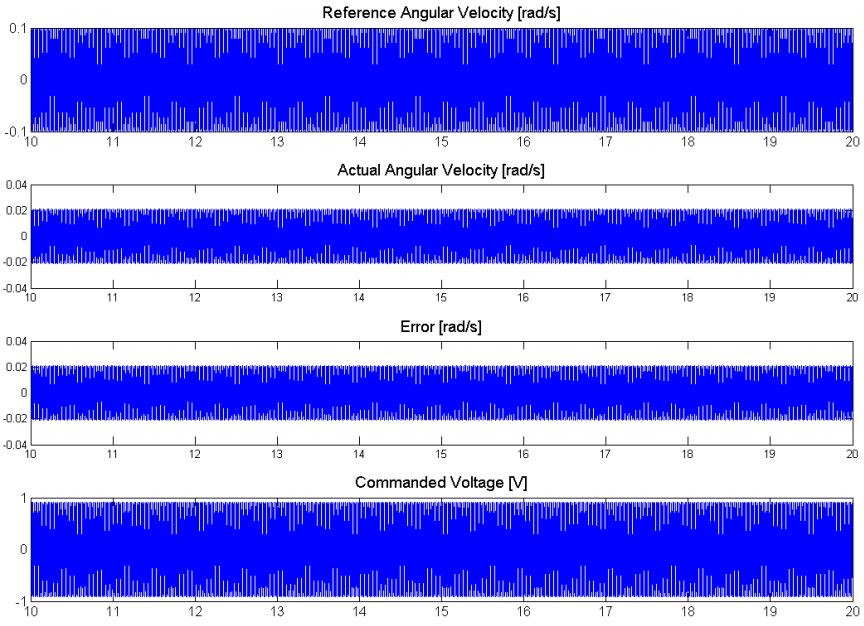

Figure 24: 60Hz Output Disturbance of 0.1 radians/sec

Position Control System - Dynamics Analysis

As shown in figure 25, there is steady-state error in the tracking function (G_{CL}) within the bandwidth. Furthermore, despite all frequencies below 0 dB for the transfer function of output disturbance rejection G_{do}, there is a peak around 60Hz which should be noticed. Both issues were critical to our objectives of control.

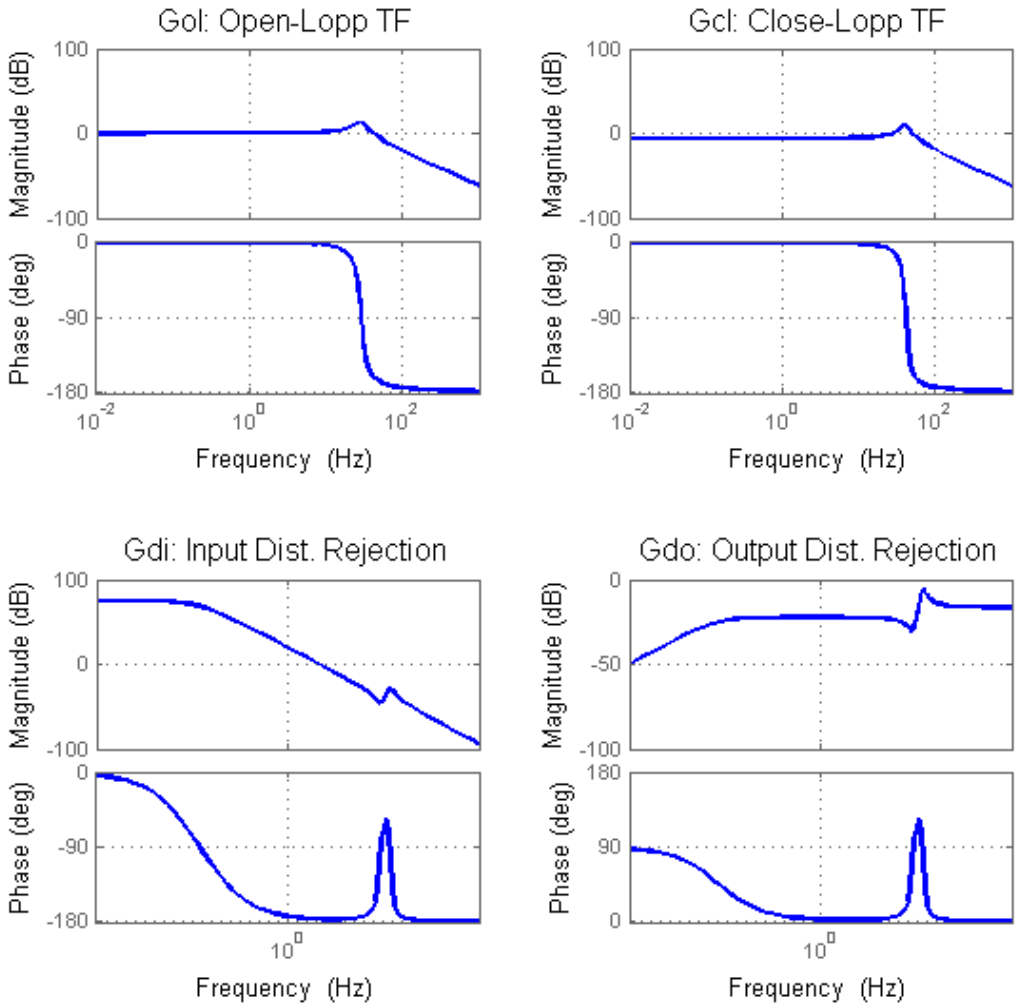

Figure 25: Bode Diagrams for the Transfer Functions of Open-Loop (G_{POL}), close-loop (G_{PCL}), input disturbance rejection (G_{di}), and output disturbance rejection (G_{do})

Position Control System Controller Tuning

K_{PP} : Proportional Gain

K_{PD} : Derivative Gain

K_{VF} : Feedforward Gain

a) Keep steady state error below 2% for a step command of π radians.

- Setting $K_{PP} = 1$, we see that the error did not fall below the required value. This is because if K_{PP} was increased until the goal was attained, the commanded voltage exceeded the continuous current limit (2A).
- In order to solve the problem, velocity feedforward control (K_{VF}) was introduced to provide better tracking performance without affecting the stability of the closed-loop system.
- When $K_{VF} = 1$, the tracking error became nearly zero.

b) Provide command tracking: Amplitude should remain within 5% of commanded amplitude of pi radians, up until 5Hz.

- With the identical setting, the tracking error of 5Hz command was close to zero as well.

c) Attenuate a 60Hz noise signal coming in either at the servo-amplifier or the sensor by at least 10 times.

- A noise signal with amplitude 0.1 was used to test disturbance rejection. For the input disturbance (from servo amplifier), over 90% was rejected; while none of the output disturbance (from the sensor) was rejected but at times, it was seen to even enlarge.
- The above situation was similar to that in the velocity control system, but τ was already tuned for the velocity loop. Therefore, we needed the derivative controller to improve the quality of disturbance rejection.
- Achieving all objectives simultaneously seemed to be impossible due to the limit on the commanded voltage (2A); in particular, the 60Hz noise was hardly attenuated five times. Thus, we removed the quantizer to get a more favorable system.
- By trial and error, finally three objectives were achieved with $K_{PP} = 2$, $K_{PD} = 0.09(N^2 = 150)$, $K_{VF} = 1$, and $\tau = 6.6$. However, the ability of output disturbance rejection was limited with amplitude 0.08. If the noise exceeded this limit, the commanded voltage would go beyond the current limit; that is, more power was required.

K_{VF}	K_{PP}	K_{PD}	N	τ
1	2	0.09	150	6.6

Table 6: Controller Gains and τ of Position Control system

Without a quantizer in the sensor, the three goals were achieved by similar tuning procedures. Due to the fact that position control was based on the velocity control, it seemed reasonable that the errors would be larger by combination if all settings were the same. Additionally, feedforward control was used in the position loop. It improved tracking performance and also eliminated the steady-state error for sinusoidal inputs, but it seemed to have no effect on the velocity control.

[2]Simulinks causality filter

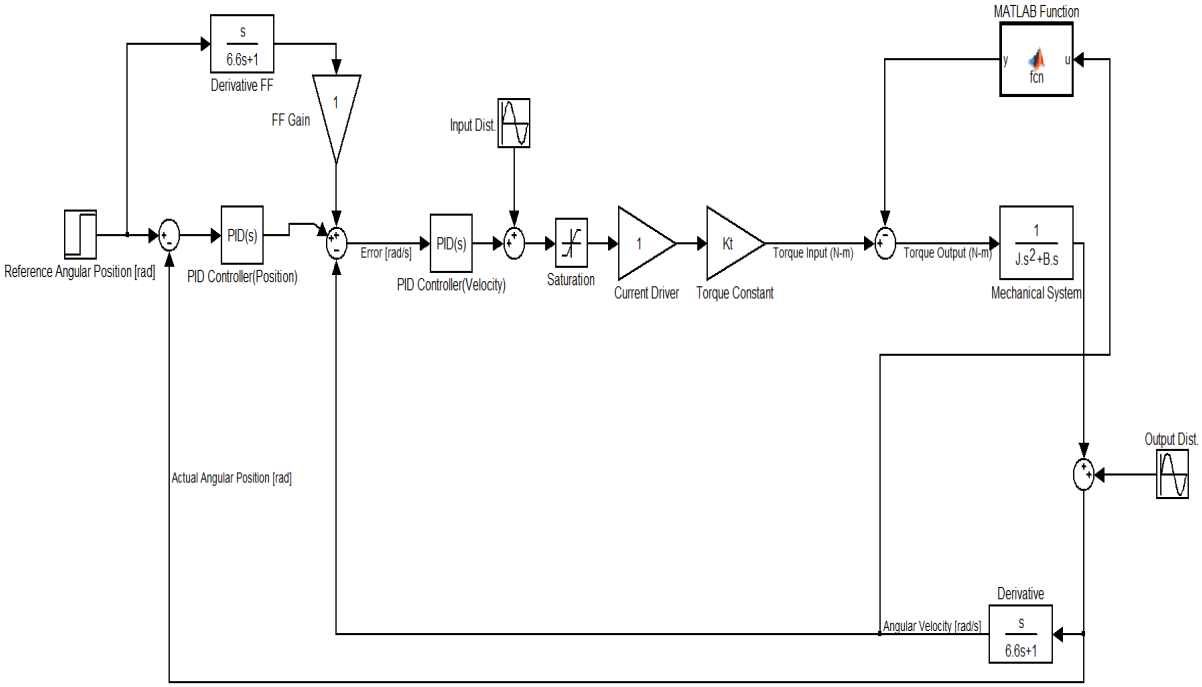

Figure 26: Block Diagram of Position Control system

Reference	Input Amplitude [rad]	Output Amplitude [rad]	%Error	Commanded Voltage [Max. V]
(d) Step	π	3.139	0.10%	0.13
(e) Sinusoidal (5Hz)	π	3.144	0.16%	0.3
(f) Input Disturbance	0.1	0.002	2.02%	1.55
Output Disturbance	0.08	0.003	3.75%	1.95

Table 7: Simulation Results for Different Reference Inputs

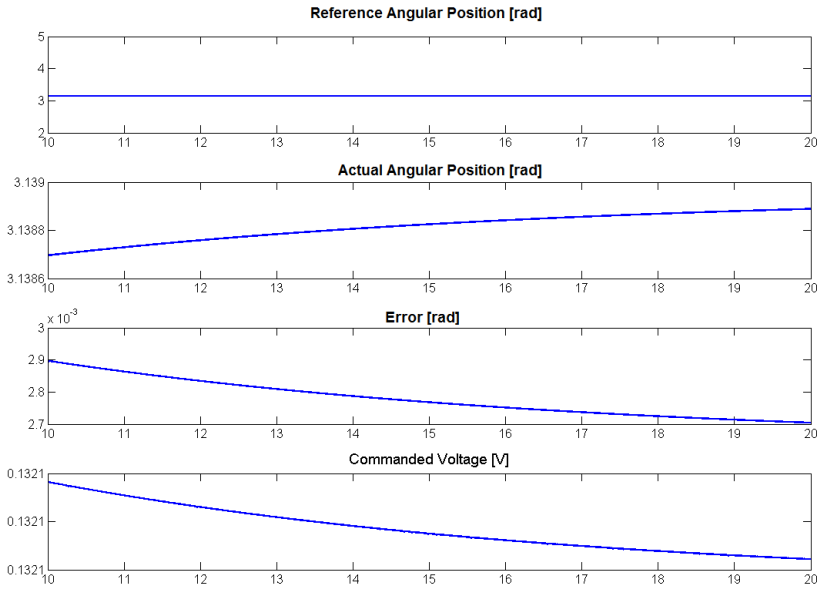

Figure 27: Step Command of π radians

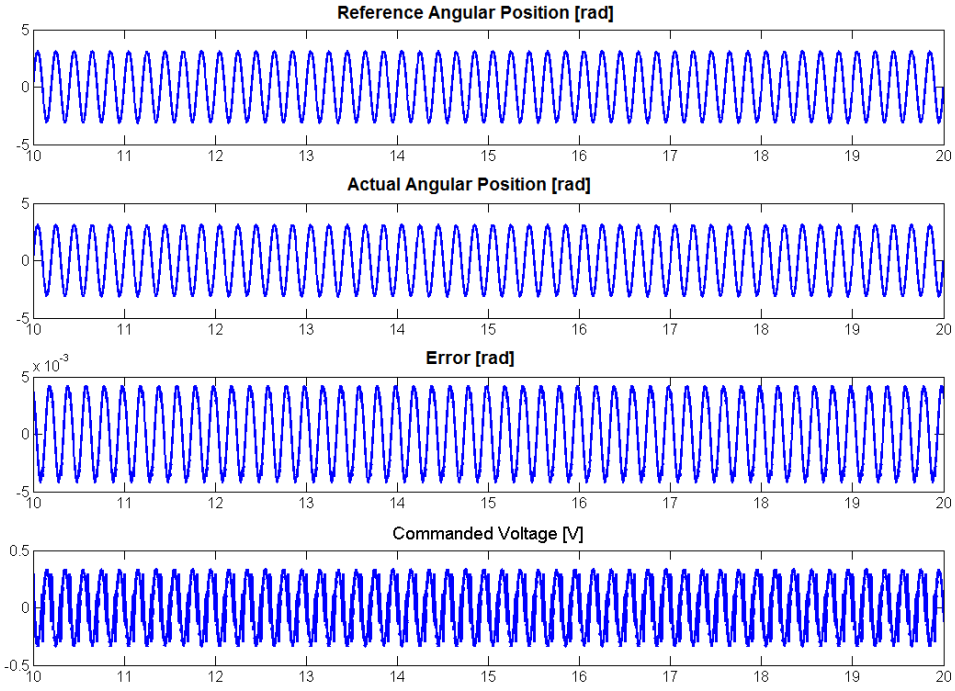

Figure 28: 5Hz Sinusoidal Command of $\pi/2$ radians/sec

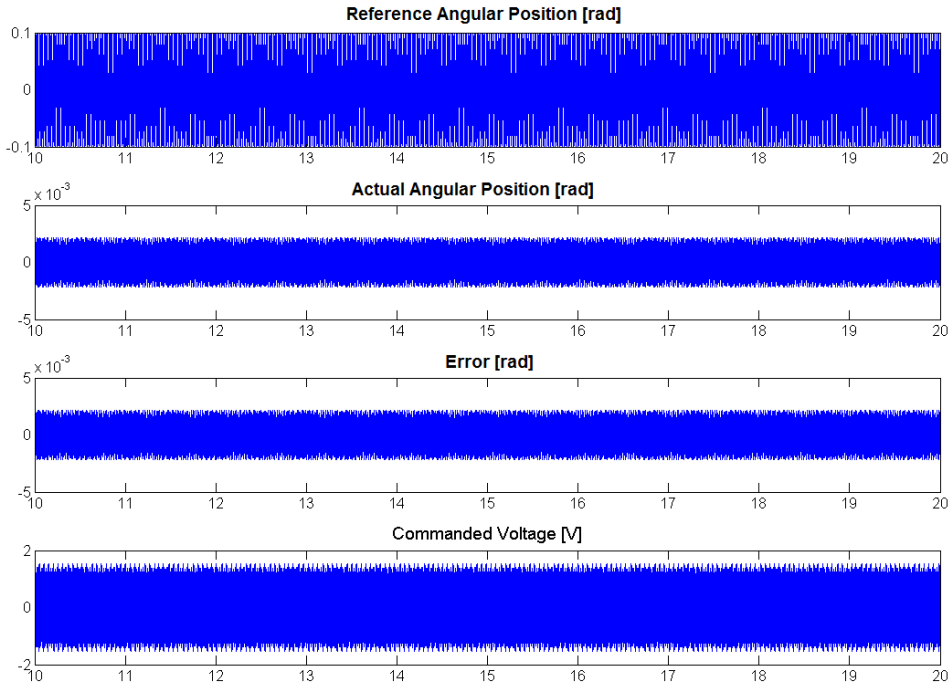

Figure 29: 60Hz Input Disturbance of 0.1 radians

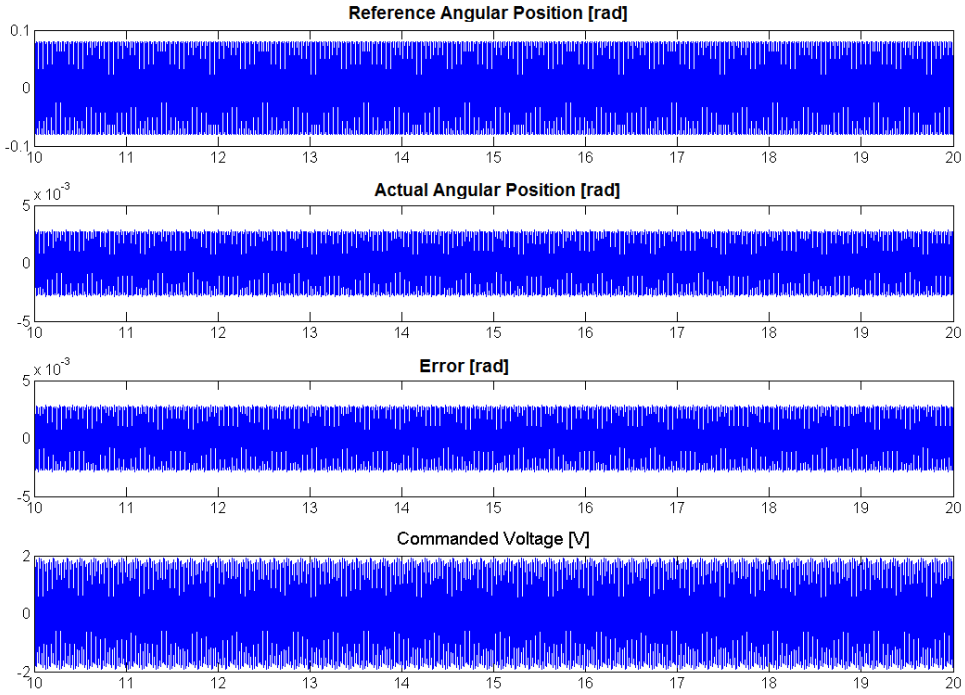

Figure 30: 60Hz Output Disturbance of 0.1 radians

4.

a.

When we implemented the controller and values from our simulation in part 3, we were unable to achieve stable output in the physical system. Therefore, we kept the same general approach - a velocity controller nested within a position controller with feed forward - but we re-tuned the system with different gain values. Because we have a velocity controller inside of our position controller, we first tuned the velocity control on its own (see section 6). Once it was working, we then implemented it in the inner loop of our position controller and tuned the position control gains. In the end we were not able to meet all of the control goals with one controller. We designed one controller with good step response and noise attenuation, and then modified the gain values to improve the frequency response separately. The results for the step response and noise attenuation are given below.

Gains	P	I	D	Feed Forward	Filter τ
Position Loop	0.1	20	0	1	6.5797
Velocity Loop	22.5	22.5	1.125	-	6.5797

Table 8: Values of position controller with nested velocity controller for step response and noise attenuation

Figure 31 shows the system's response to a position step input of π radians, and figure 32 shows a zoomed in view after it had settled. The peak error here is 0.11%, which out performs the goal of 2% error. Additional results for position steps of different magnitudes are included in section 5b. The error seen here is very similar to the steady state error of 0.10% predicted in our simulink simulation in response to a step input of π radians.

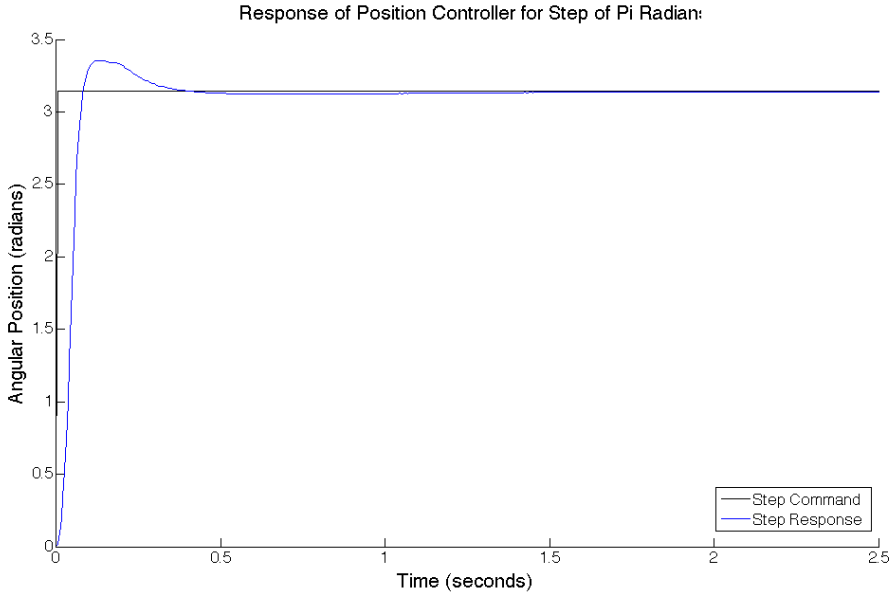

Figure 31: Step response of position controller

Figure 33 shows the noise attenuation during a step input of π radians. To simulate the noise within LabView, we added a 60 Hz sine signal with an amplitude of 1 to the output of the DAQ assistant. Figure

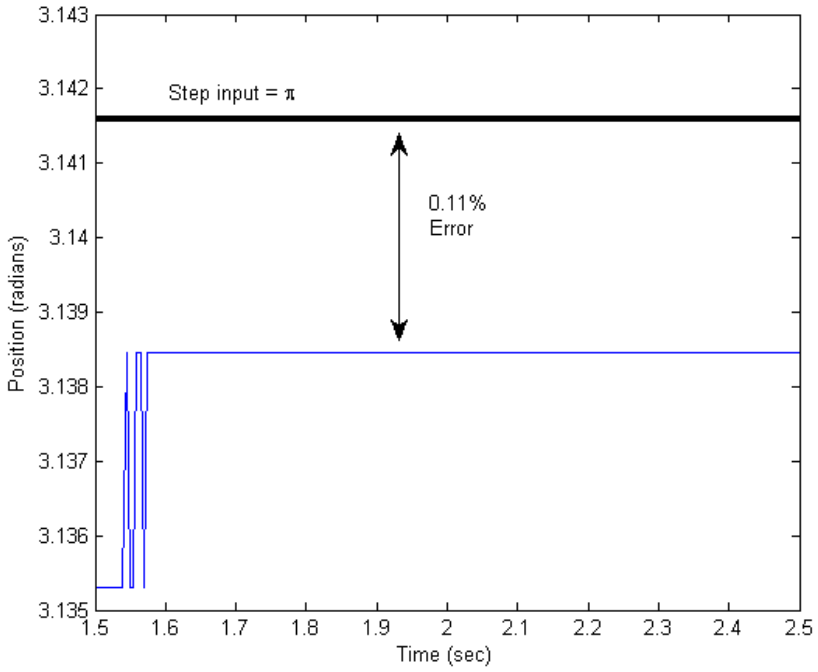

Figure 32: Steady state error for position step input of π

34 shows a zoomed in view after the step has taken place. For the step response with no noise, our steady state error was approximately 0.003, and with the noise the highest error is 0.060. Given that we were inputting a noise amplitude of one and the error only increased by 0.057, we have attenuated the noise by approximately 17.54 times. This exceeds the goal of a ten times reduction.

The position controller we were using for the step response and noise attenuation did not not perform well with sinusoidal inputs forcing us to modify gains while keeping the overall architecture the same. The relevant values are given in the table 9, and the following figures show the response at various frequencies with an amplitude of π. Figure 35 shows the error as a function of input frequency. During our tests the position output remained stable up to 5 Hz, but the output amplitude error exceeded 5% just beyond 4 Hz. This does not meet the goal of less than 5% error up to 5 Hz. The performance of the real system at 5 Hz with an output magnitude of -2.078 dB performs far worse than our simulated system which maintained a 0 dB magnitude at the same frequency

Gains	P	I	D	Feed Forward	Filter τ
Position Loop	0	20	0	1	6.5797
Velocity Loop	3	7	0.4	-	6.5797

Table 9: Values of position controller with nested velocity controller for frequency response

To determine the closed-loop bandwidth of the system, we slowly increased the frequency of the input signal until the output amplitude just crossed -3 dB (for a input of π this corresponded to an output of 2.22). We found that the bandwidth of our position control system was 5.27 Hz, as shown in figure 36.

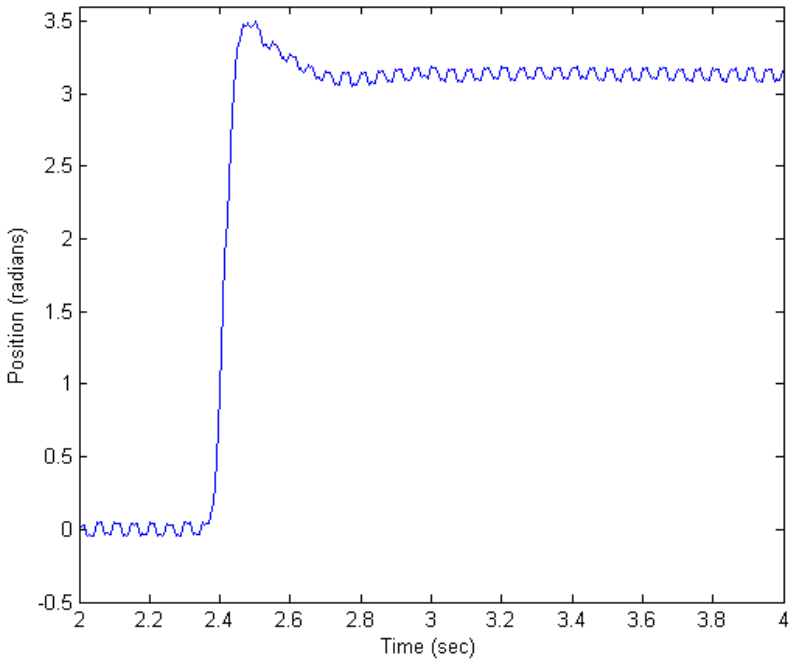

Figure 33: Step response for input of π with a 60 Hz noise signal (amplitude = 1)

Frequency (Hz)	Input Amplitude	Output Amplitude	Magnitude (dB)	%Error
1	π	3.13845	-0.008693172	-0.100033771
2	π	3.02221	-0.336504689	-3.800067888
3	π	3.079209	-0.174214105	-1.985733367
4	π	3.030152	-0.313709167	-3.547266176
5	π	2.473084	-2.078220099	-21.27929134
5.27	π	2.181877	-3.166392169	-30.54869805

Table 10: Position control data.

Even though we included many nonlinear factors in our simulation, we still faced difficulties meeting all of the performance specifications, particularly trying to do so with just one controller. In general, we found that when we adjusted gains to meet the step requirement, the frequency response usually got worse, and vice-versa. It's possible that some of the values used in our simulation are incorrect (even after testing and regression), or that there are other factors we did not account for, such as the dynamics of the servo-amp. In the end we were able to meet the steady state error for step input and noise attenuation requirements, and were very close to the frequency response requirement.

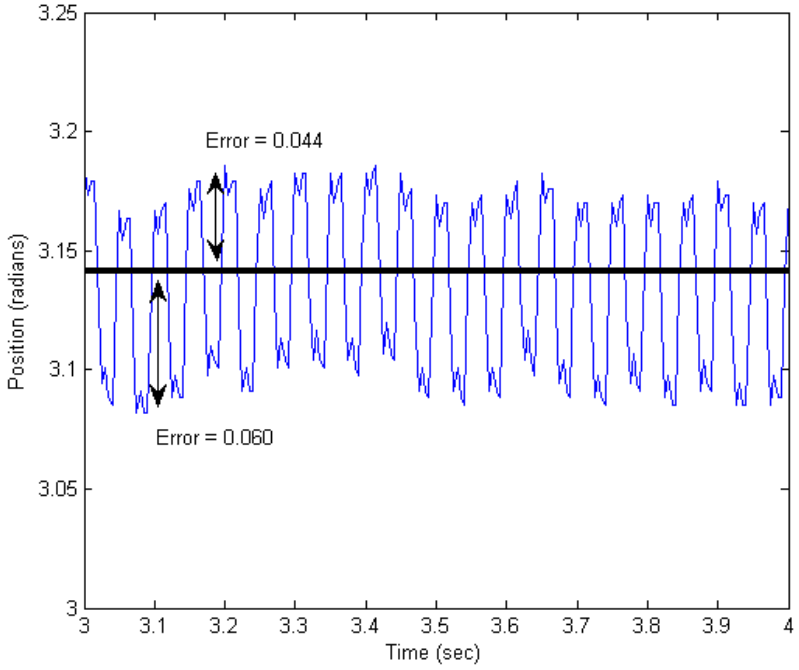

Figure 34: Steady state step response for input of π with a 60 Hz noise signal (amplitude = 1)

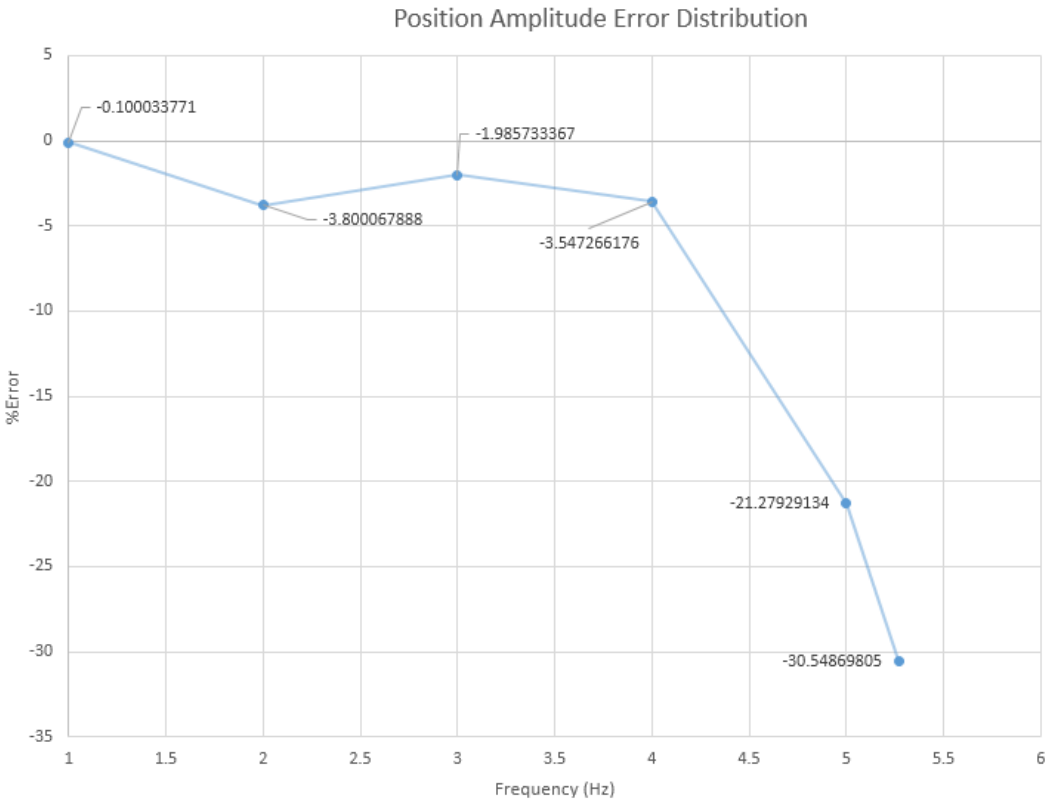

Figure 35: Position error vs. frequency for an input amplitude of π radians

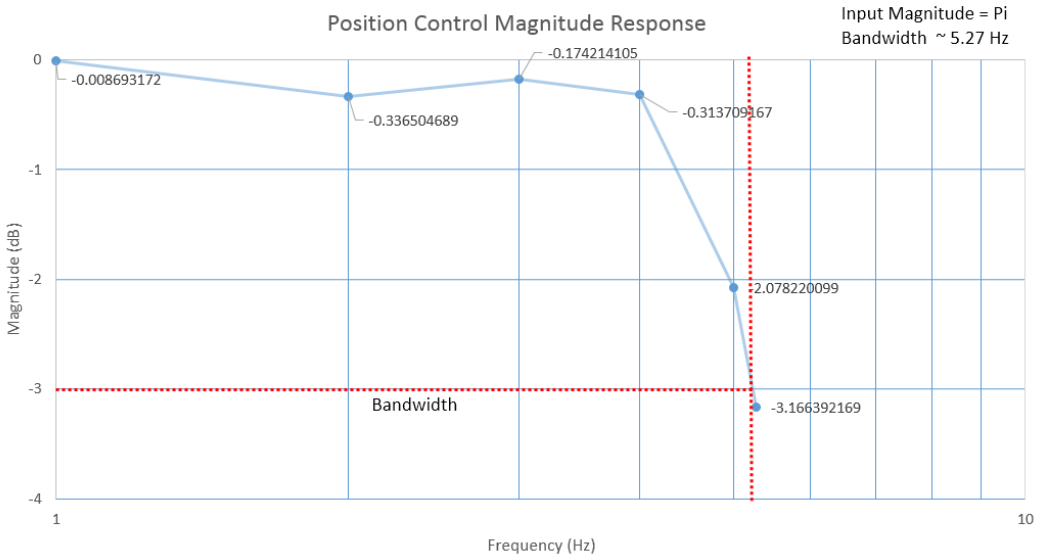

Figure 36: Output magnitude vs. frequency used to determine bandwidth of position controller

5.

Comparison of Our System Model with the Data Sheet and the Physical System

	Original Model	Original Model (with gains used in VI)	New Model (with gains used in VI)	Real Physical System
BW of Velocity Loop [Hz]	6.52	6.52	2.8	2.9
BW of Position Loop [Hz]	11.4	11.8	5.1 *(with I =1)*	5.27 *(with I =10)*

The original model was built using the values of J, b and τ_f and optimized to provide the best performance. We then proceeded to measure the actual system parameters empirically. Using the measured values as well as accounting for non-linear effects like saturation and varying friction, we made a second model - the new model. At the same time we proceeded to tune the physical system. We then incorporated the practical values used in the real system on our theoretical system for comparison of how well our model simulates the real world. As you can see from the table above, our model does a very realistic job of modeling the system.

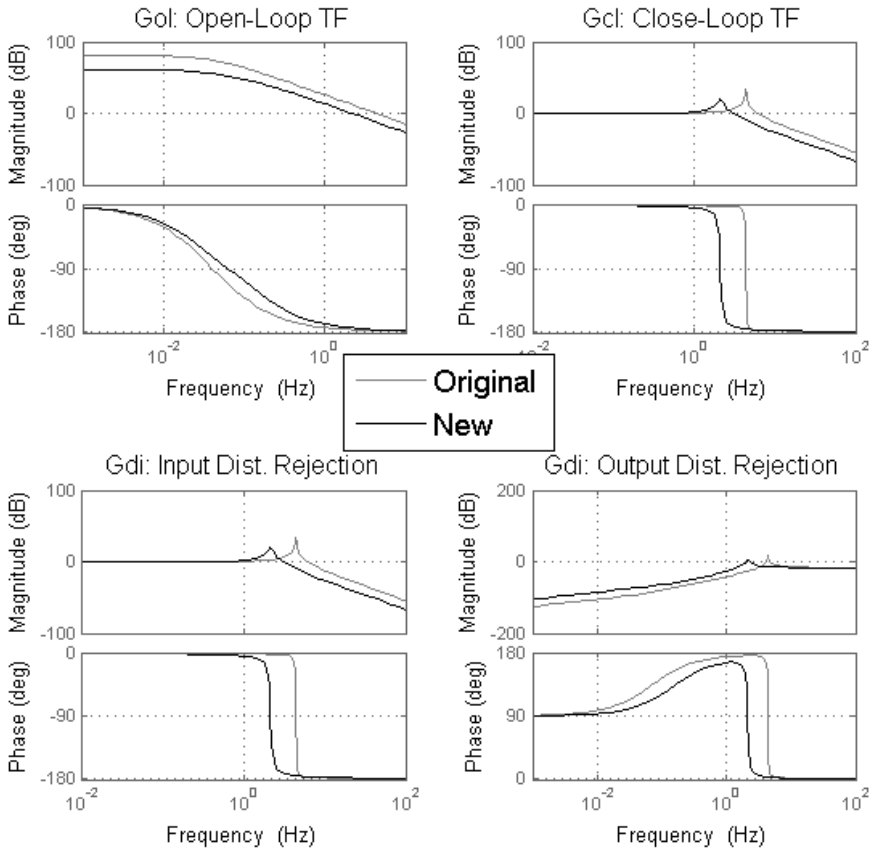

Figure 37: Comparison of the new model with the original model.

a. Coulomb Friction

The bearings in the motor are the primary source of coulomb friction in the motor encoder system. This could be especially true in the current configuration of the motor because Pittman's website only specifies that they use ball or sleeve bearings in their products. If these are only radial bearings with no thrust capability, then the motor's performance could be affected by the vertical mounting we are using. Additionally, there is friction between the brushes and commutator. In section 2b we discussed and carried out experiments and modeling to determine the friction value.

Setting aside the earlier parameter identification, there are some simple experiments that could be run to estimate the friction in the system. If we assume that the torque constant given in the data-sheet is correct and that the gain of the servo-amp is exact and known, then for a given command voltage, the motor should be capable of a specific torque output. By slowly increasing the voltage from zero until the motor just begins to turn. From this, the torque caused by static friction would be equal to the product of the torque constant and the input voltage (because there is a 1:1 relationship between command voltage and applied current).

A similar method could be used to determine sliding friction. By applying a constant voltage to the system, the motor would come to a steady-state velocity. If we the coefficient of damping is known, then the torque caused by sliding friction is equal to the motor torque (K_t * $V_{command}$) minus the torque from damping (the product of the damping coefficient and velocity). This was carried out in part 2b.

Friction was included in our Simulink model from the beginning of our modeling, as presented in part 3. At the beginning of our tests we used approximations based on the data-sheet values, but as we performed more regression analysis, we obtained new values that should be closer to the true values of the physical system.

b. Saturation

In the actual system, the effects of saturation can clearly be seen in the steady increase of rise time as well as the decreasing stability of the system reflected in the higher percent overshoot and larger steady state error. With saturation, the system is not able to move at the speed at which the controller requests, meaning that rise time will become dependent on the actual input magnitude, rather than being independent as is normally the case without saturation. The loss of stability that we see is an effect of integral wind up. Because the system takes longer to respond, our integral term has a longer period of time to accumulate error, which then has to be canceled out with positive error, leading to large overshoots and long settling times. In extreme cases this can drive the system to instability. The effects of saturation on integrator wind up can be compensated for by limiting the maximum and minimum values that the integrator can accumulate. Another method is to model the saturation effect and use this model to alter the input to the integrator such that when the driver becomes saturated, the integrator no longer receives input.

As can be seen in figure 38 our model also exhibits the same large rise time that we would expect for extremely large input signals. For an input of $\pi \times 10^5$ we see a rise time of roughly, 18 seconds, or 260 times larger than the rise time for a step of π. If there was no saturation effect, we would expect to see the exact same rise time. While our physical system appears to be going unstable for large step magnitudes, the simulation refused to become unstable even for enormous step sizes.

Our experimental system exhibits distortion due to changes in step size immediately because our aggressive feed forwards design was already using the all of the available command voltage.

Step Input Amplitude (Radians)	1π	1.5π	2π	3π	4π
Rise Time (seconds)	0.0700002	0.0800000	0.0850000	0.110000	0.125
Settling Time (seconds)	0.290000	0.320000	0.28000	0.54500	1.45500
Percent Overshoot (%)	6.79997	16.8000	31.8500	65.0331	91.0003
Steady State Error (%)	0.100034	0.594022	0.150008	0.13335	0.399713

Table 11: Step Response for a variety of input positions. Where rise time is defined as reaching 90% of the final value while settling time is the time after which the function remains within 2% of its final value

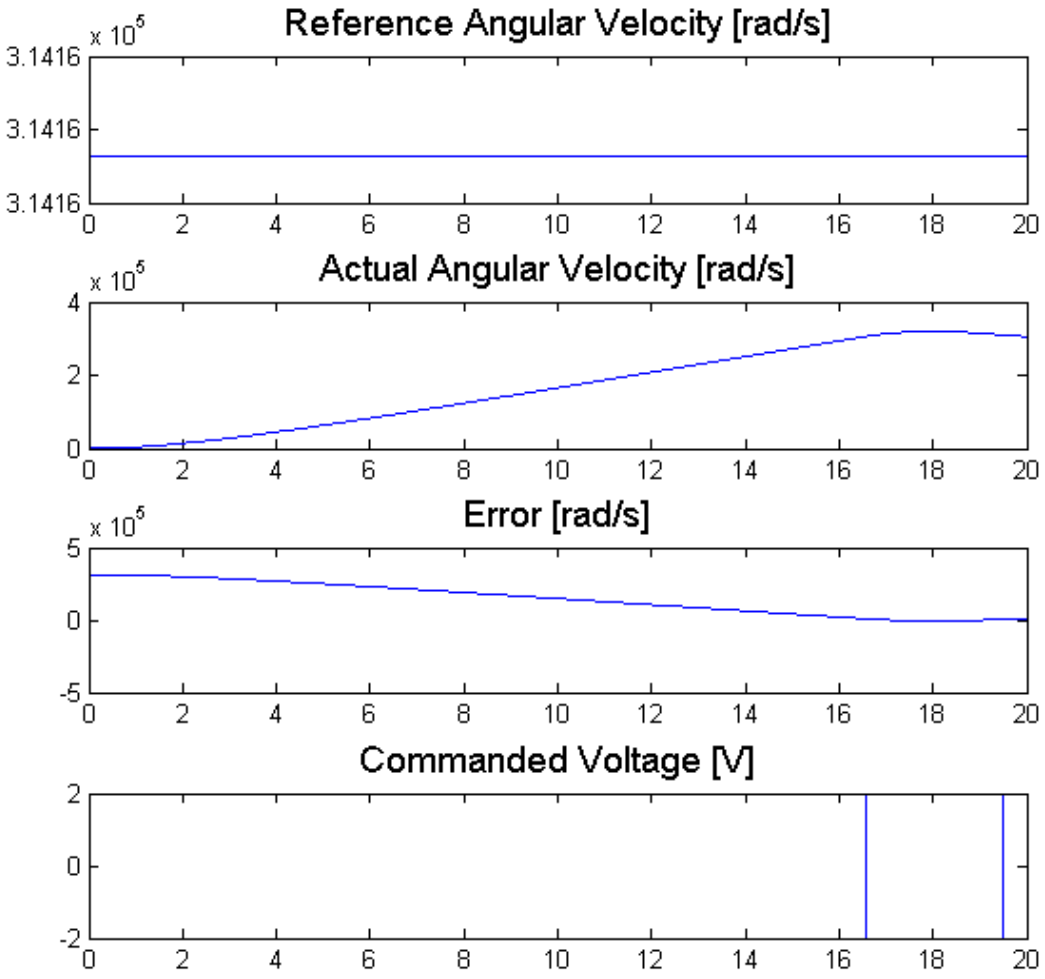

Figure 38: Tracking Ability for Extremely Large amplitude of step command

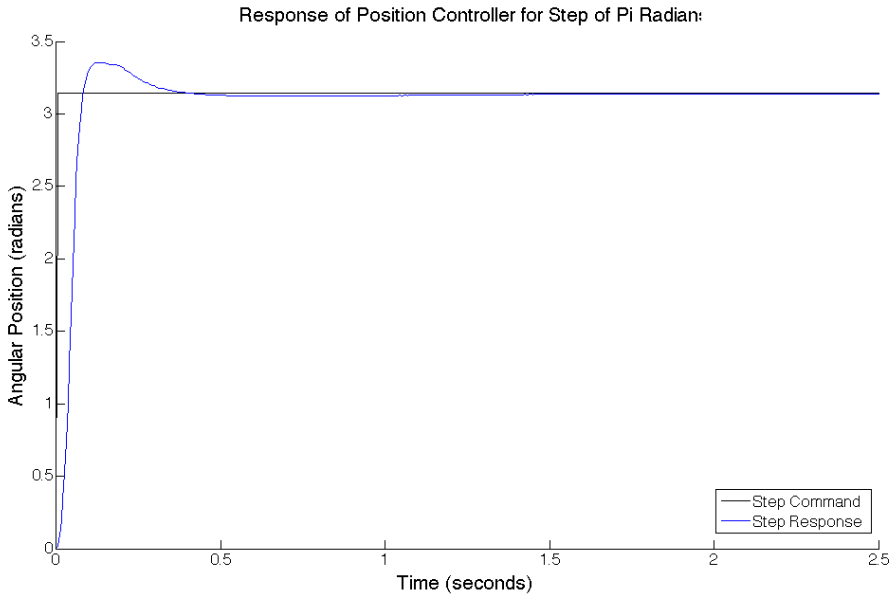

Figure 39: Step response of position controller

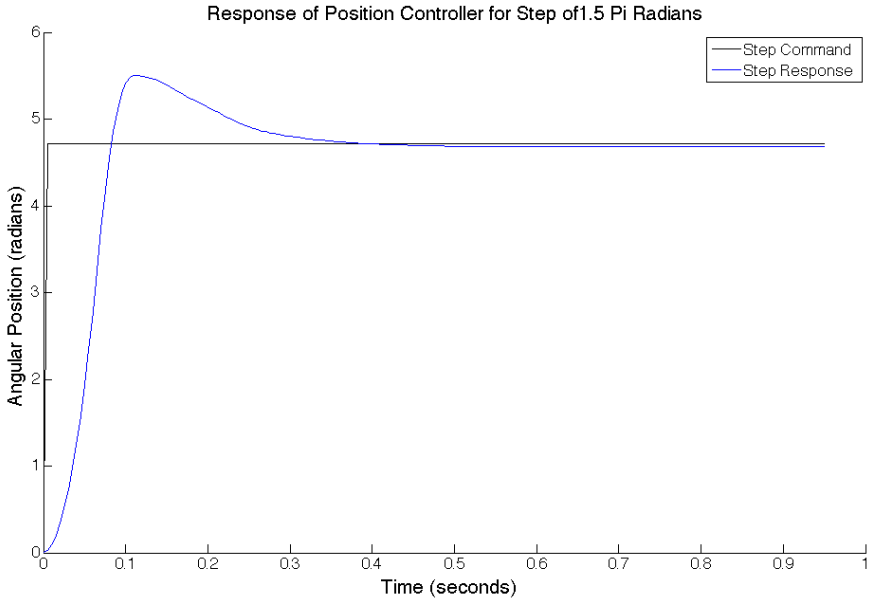

Figure 40: Step response of position controller

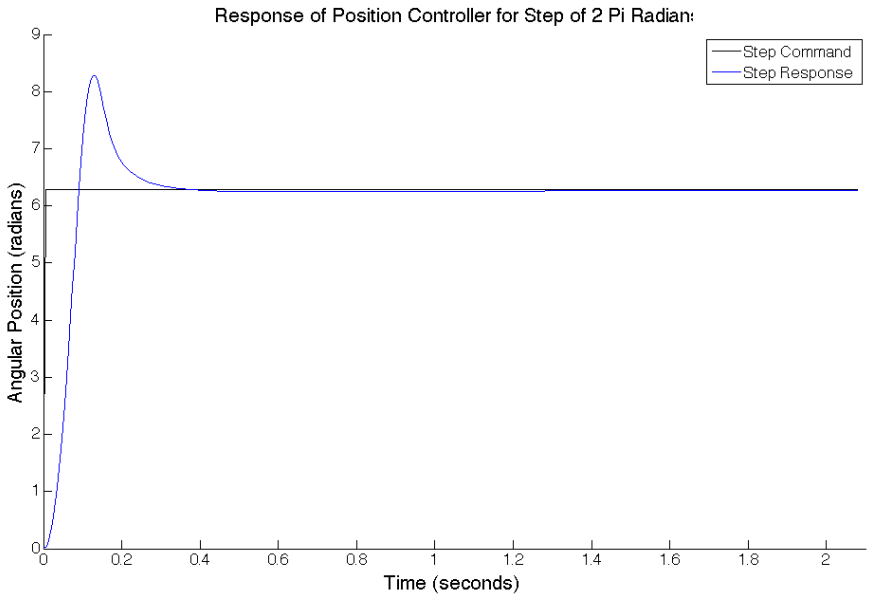

Figure 41: Step response of position controller

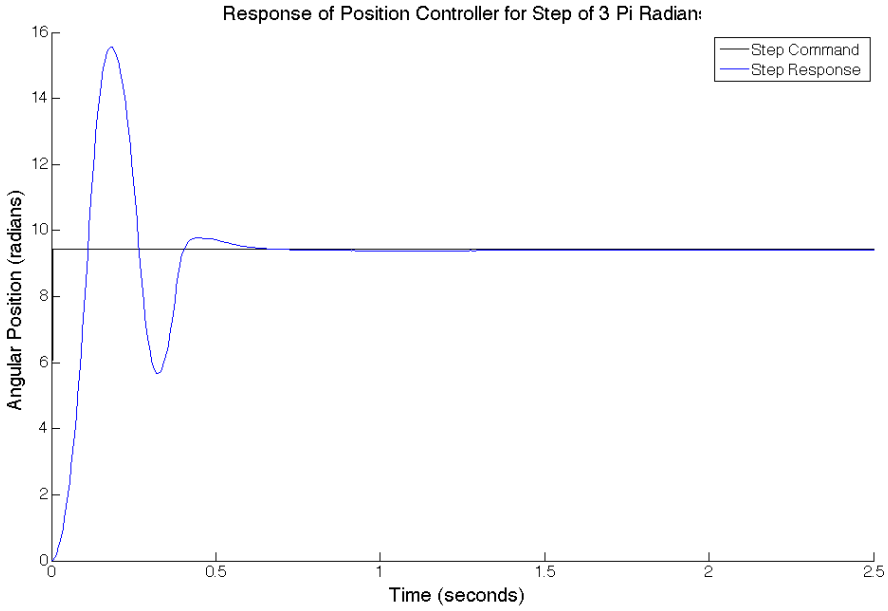

Figure 42: Step response of position controller

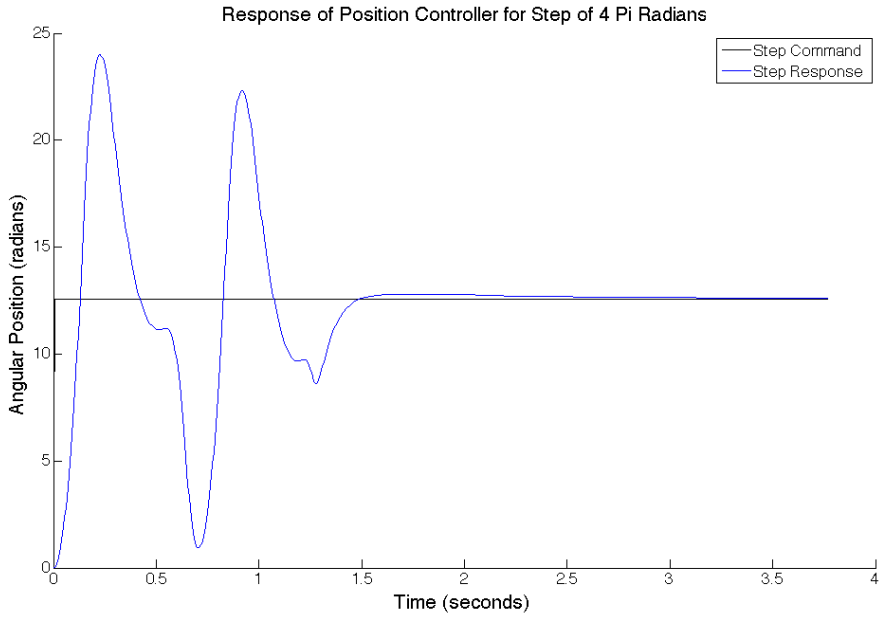

Figure 43: Step response of position controller

c. Quantization

The real behavior of the optical encoder was already modeled as the quantizer with an interval $2\pi/2000$ in our model. The higher resolution of the sensor, the closer the system is to a linear unity gain block, providing less measured error. The tracking error of sinusoidal input exceeded 5% as the resolution reached 75 step/rev, while the error of step input went beyond 2% when the resolution neared 325 steps/rev.

Most parasitic effects have been considered for our model. For quantization, we find it causes significant trouble in our position control. In feedback control systems, the sum of loop sensitivity and complementary sensitivity is one. Since the sensed error which resulted from quantization is based on the complementary sensitivity, more power is required to achieve the same closed-loop response; otherwise, the tracking error cannot be lowered in a desired manner.

The effects of quantization on velocity estimation cannot be overstated. Without a proper model of the system velocity estimation requires some sort of filtering to average over previous time steps. But this introduces a time delay in the velocity estimation and can even cause it to miss quick transients entirely.

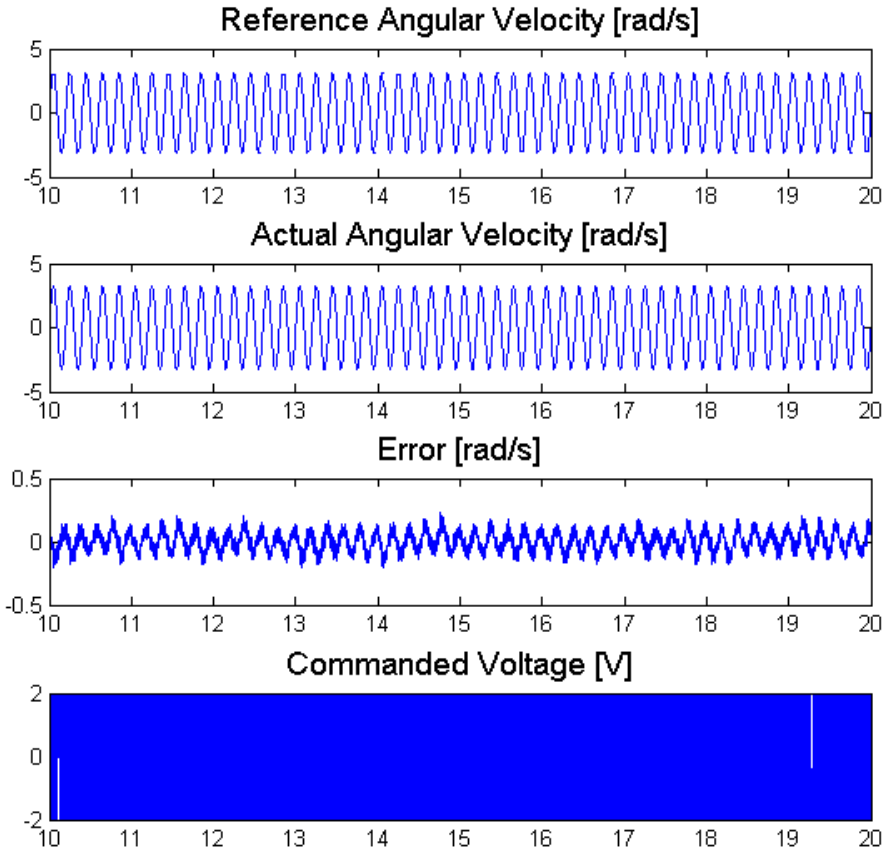

Figure 44: Sinusoidal Tracking Error $> 5\%$ when Resolution $= 75$ steps/rev

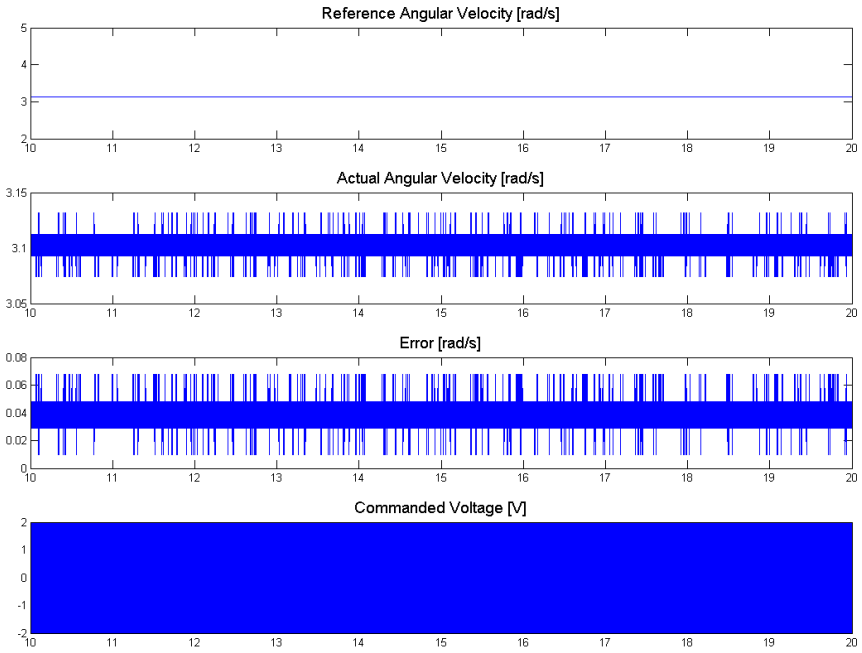

Figure 45: Step Tracking Error $> 2\%$ when Resolution $= 325$ steps/rev

6.

As was discussed in section 4, work on the velocity controller for the real system was performed before the position controller, because we needed to nest the velocity loop within the position loop. The gains we used in the simulation were not able to stabilize the system with the performance we wanted, so we re-tuned the gain values. As with position control, we were not able to use one controller to meet all of the requirements: we designed one controller that could meet the step and noise requirements, and then modified the gain values to improve the frequency response separately. The results for the step response and noise attenuation are given below. The gains used here are the same as in the step position controller's nested velocity loop.

Gains	P	I	D	Filter τ
Velocity Controller	22.5	22.5	1.125	6.5797

Table 12: Values of velocity controller for step response and noise attenuation

Figure 46 shows the system's response to a velocity step input of $2{\cdot}\pi$ radians, and figure 47 shows a zoomed in view after it had settled. The peak error here is approximately 0.16% (and on average is much less), which greatly exceeds the goal of 2% error. In section 3 we showed that our simulation had a steady state error of 0.08%. While our real controller had twice as much error, both still greatly exceeded the performance goal. Additional results for velocity steps of different magnitudes are also shown in the following tables and figures.

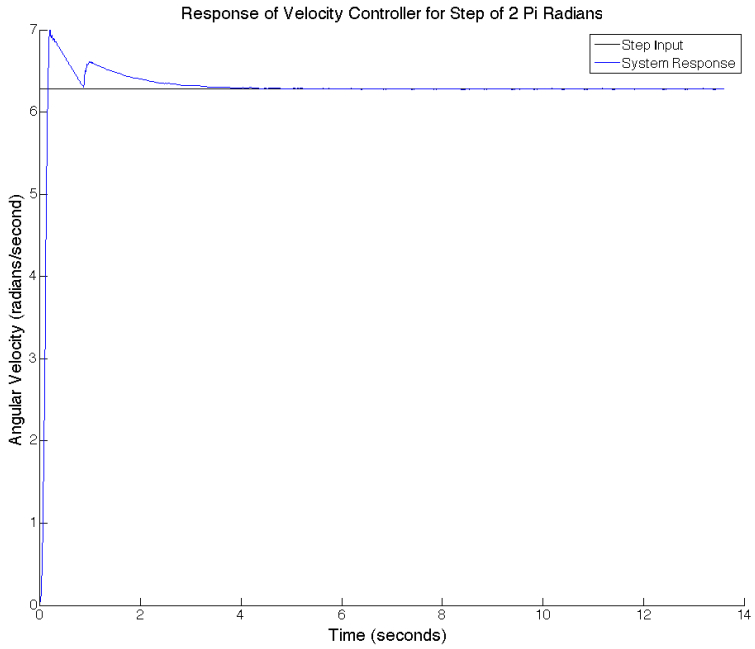

Figure 46: Velocity Controller Step Response

Figure 53 shows the noise attenuation during a step input of $2{\cdot}\pi$ radians/sec. To simulate the noise within LabView, we added a 60 Hz sine signal with an amplitude of 1 to the output of the DAQ assistant, just as with the position controller. Figure 54 shows a zoomed in view after the step has taken place. For the

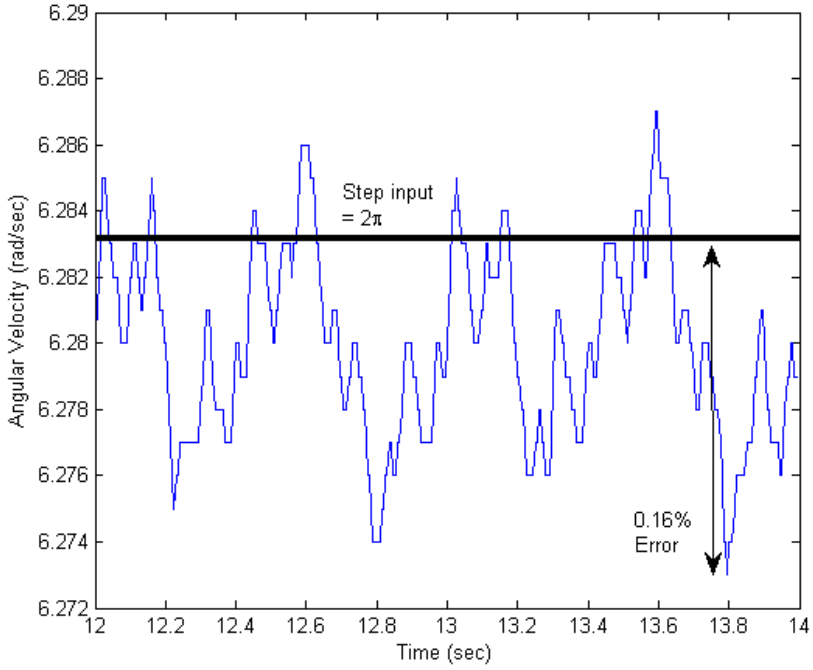

Figure 47: Steady state error for step input of $2{\cdot}\pi$ rad/sec

Step Input Amplitude (Radians)	$0.5 \cdot \pi$	$1 \cdot \pi$	$1.5 \cdot \pi$	$2 \cdot \pi$	$5 \cdot \pi$	$10 \cdot \pi$
Rise Time (seconds)	0.11	0.115	0.125	0.14	0.23	0.385
Settling Time (seconds)	1.5700	1.6700	1.95500	1.9700	2.3050	2.8550
Percent Overshoot (%)	4.3415	6.1564	7.9919	11.3925	21.091	27.1553
Steady State Error (%)	0.11779	0.041862	0.053597	0.058333	0.050267	0.050634

step response with no noise, our peak steady state error was 0.010, and with the noise the highest error is 0.20. Given that we were inputting a noise amplitude of one and the error only increased by 0.19, we have attenuated the noise by approximately 5.26 times. This meets the goal of a five times reduction. Our simulation had an error with noise of 2.05%, which corresponds to an error of 0.16. The real controller was very close to matching the simulation's performance.

Step Input Amplitude (Radians)	$2 \cdot \pi$	$2 \cdot \pi$ with Noise
Rise Time (seconds)	0.14	0.135
Settling Time (seconds)	1.9700	-
Percent Overshoot (%)	11.3925	29.6000
Steady State Error (%)	0.058328	0.054649

The velocity controller we were using for the step response and noise attenuation did not not perform well with sinusoidal inputs, therefore we modified to gains to try to improve the frequency response. The relevant values are given in the table 13, and the following figures show the response at various frequencies with an amplitude of $\pi/2$. Figure 55 shows the error as a function of input frequency. During our tests the velocity

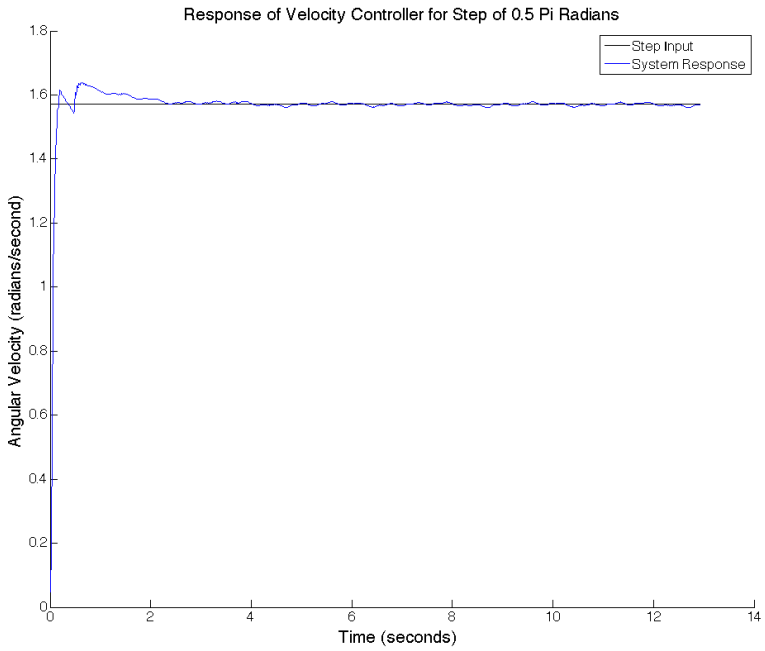

Figure 48: Velocity Controller Step Response

output remained stable up to 5 Hz, but the output amplitude error increased greatly. The error exceeded 5% just beyond 2.2 Hz. This does not meet the goal of less than 5% error up to 5 Hz. In the simulation we were able to achieve 4.46% error up to 5 Hz. Therefore, our real controller was well below the performance of the simulation.

Gains	P	I	D	Filter τ
Velocity Loop	2	9	0.3	6.5797

Table 13: Values of velocity controller for frequency response

$$\text{Predicted Bandwidth } = 2.8Hz$$

$$\text{Experimental Bandwidth } = 2.9Hz$$

To determine the closed-loop bandwidth of the system, we slowly increased the frequency of the input signal until the output amplitude just crossed -3 dB (for a input of $\pi/2$ this corresponded to an output of 1.11), just as we did in section 4 to determine the bandwidth of the position controller. We found that the bandwidth of our position control system was 2.9 Hz, as shown in figure 56. This is very close to the bandwidth value predicted by our simulation.

Just like with the position controller, we faced a lot of trouble meeting all of the performance specifications with just one controller. Again, we found that when we adjusted gains to meet the step requirement, the frequency response usually got worse, and vice-versa. As we mentioned in section 4 ,it's possible that some of the values used in our simulation are incorrect, or that there are other factors we did not account for. In the end we were able to meet the steady state error for step input and noise attenuation requirements, but

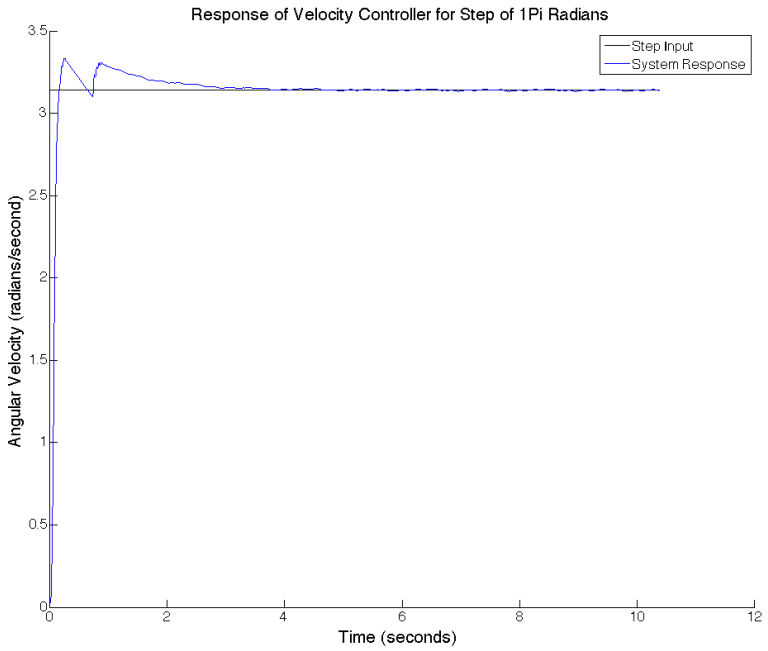

Figure 49: Velocity Controller Step Response

Frequency (Hz)	Input Amplitude	Output Amplitude	Magnitude (dB)	%Error
1	$\pi/2$	1.649	0.422015572	4.978600463
2	$\pi/2$	1.632143	0.332766594	3.905450513
2.9	$\pi/2$	1.11778	-2.955270847	-28.83991508
3	$\pi/2$	1.06516	-3.374100561	-32.18980833
4	$\pi/2$	0.581875	-8.625803574	-62.956687
5	$\pi/2$	0.47833	-10.32784514	-69.54856643

Table 14: Numerical results of velocity control.

were not very close to the frequency response requirement.

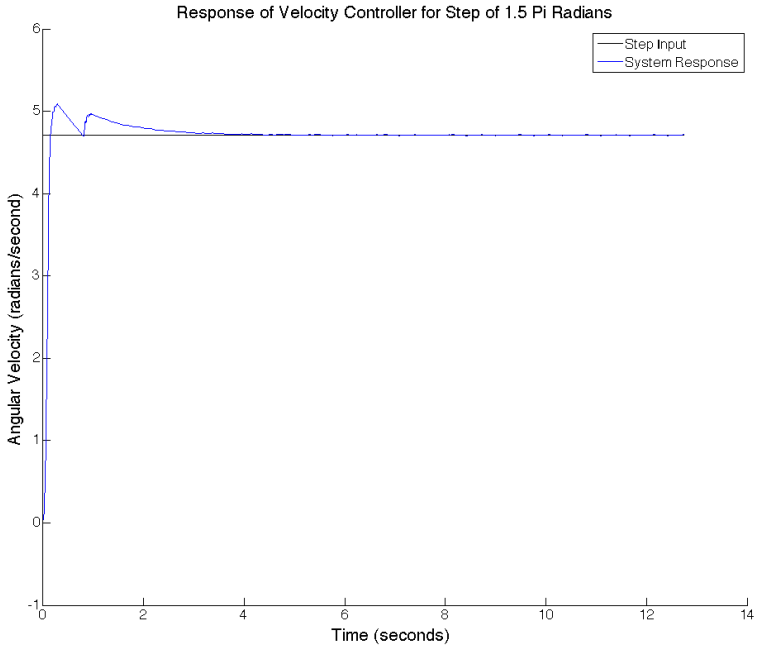

Figure 50: Velocity Controller Step Response

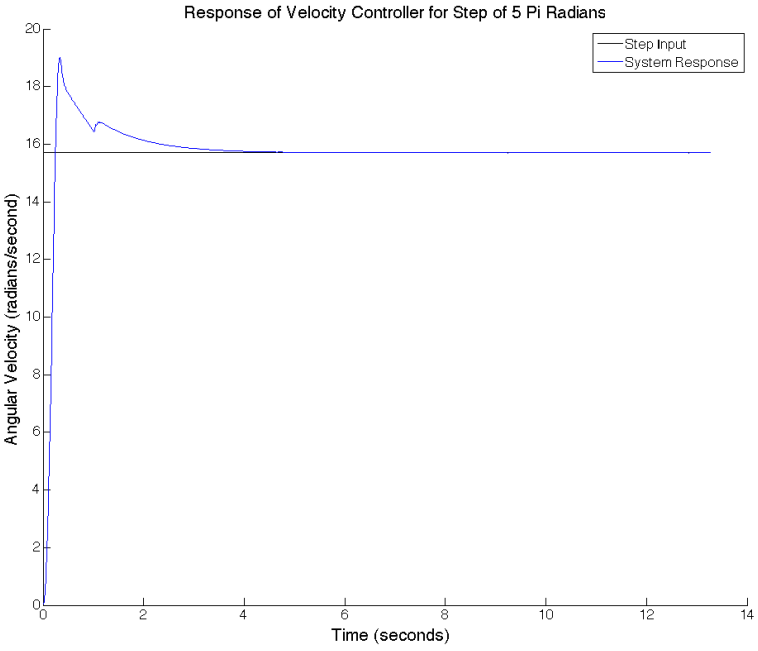

Figure 51: Velocity Controller Step Response

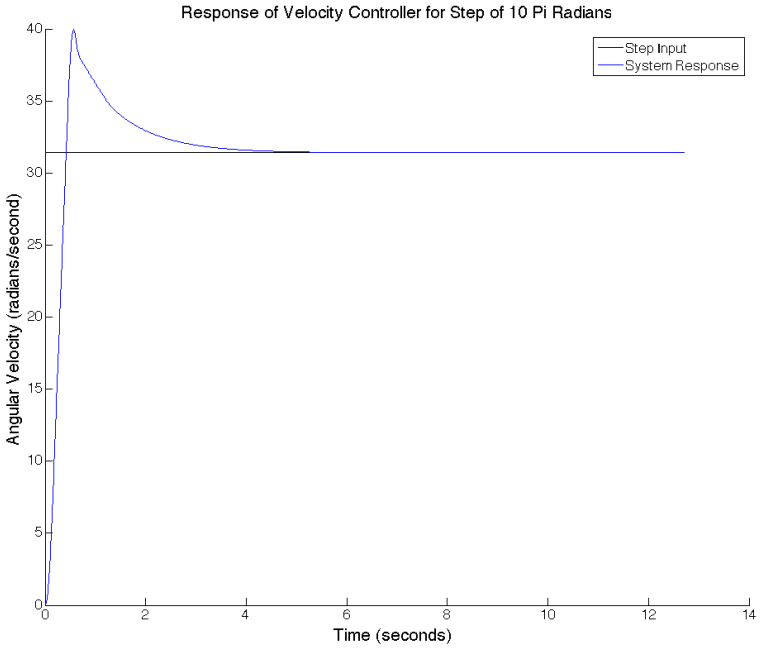

Figure 52: Velocity Controller Step Response

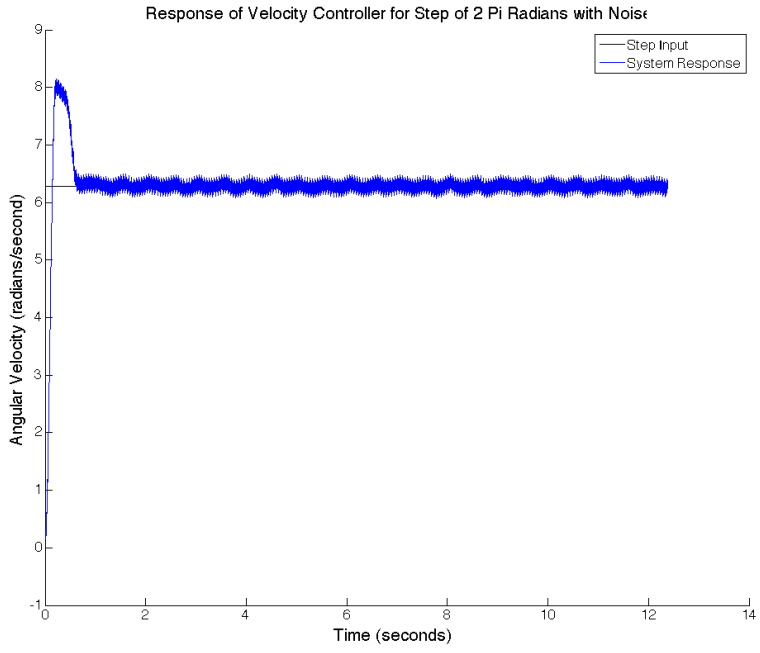

Figure 53: Velocity step response for input of $2 \cdot \pi$ with a 60 Hz noise signal (amplitude = 1)

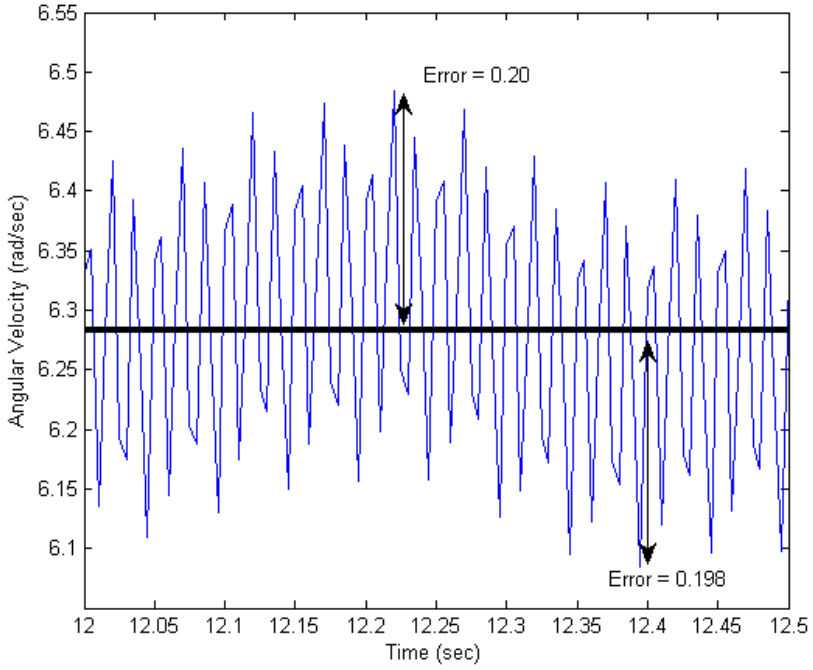

Figure 54: Steady state velocity step response for input of $2 \cdot \pi$ with a 60 Hz noise signal (amplitude = 1)

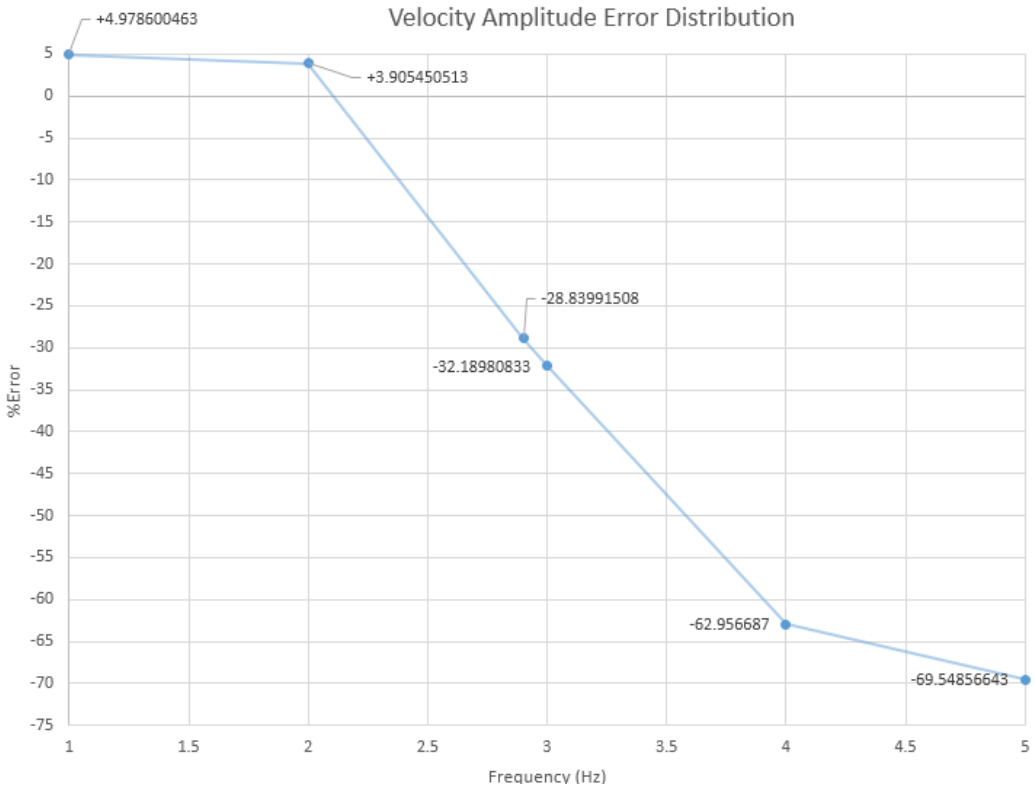

Figure 55: Velocity error vs. frequency for an input amplitude of $\pi/2$ rad/sec

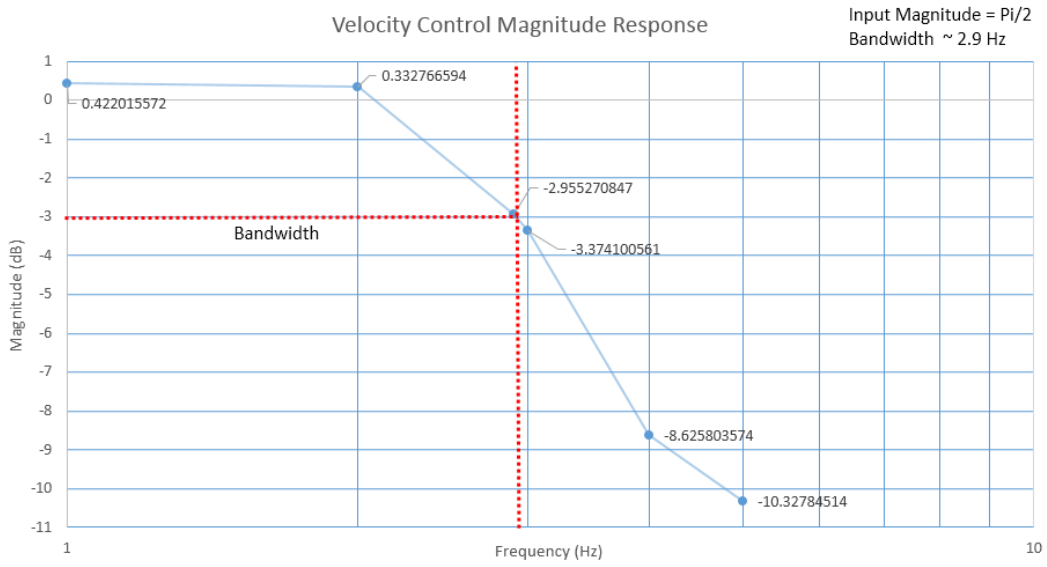

Figure 56: Output magnitude vs. frequency used to determine bandwidth of velocity controller

Appendix : Matlab Code

enchancedParameterEstimation.m

```
function [stateC] = enchancedParameterEstimation(Tin,Thetain,Iin, state0, lockedPositions)
%Oh god the state so many terms
% where state0 = [J; Kt; Bf; Br; Tff; Tfr]
% Bf and Tff are the viscous and coulomb damping forces applied when
% rotating or attempting to rotate in the positive direction
% Br and Tfr for the reverse direction
% lockedPositions is a vector that contains non zero elements in positions
% that will not be optimized

global state LPos numUPos staticRatio staticThreshold Tobs Iobs Thetaobs numPoints
assert((length(state) == 6),'Invalid state length');
assert((length(state) == length(lockedPositions)),'should have been the same length');

% ode45 sort of works, but is very slow, static friction really gives it a
% hard time

numPoints = 2*10^4;
state = state0;
LPos = lockedPositions;

% how many state variables are we actually going to regress on
numUPos = length(state);
for i = 1:length(LPos)
    if LPos(i)
       numUPos = numUPos - 1;
    end
end

staticRatio = 1.0;  % static friction assumed to be 10% larger than sliding friction to avoid stutterin
staticThreshold = 1e-6; % below this we aren't moving
Tobs = Tin; Thetaobs = Thetain; Iobs = Iin;

xstart = [0; 0];
stateC = state0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initial plot to make sure the shooting works correctly
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[T, X] = shootMotor(xstart,stateC);
figure
hold on;
plot(T,X(:,1),'b');
plot(Tobs,Thetaobs,'k');
title('Initial Estimate curve');
xlabel('Time (seconds)')
ylabel('Angle (radians)')
hold off;

dPtol = 1e-6;
```

```
dP = inf;
iter = 0;
maxiter = 1000;
lastE = inf;
lastS = stateC;

while ((max(abs(dP)) >= dPtol) && (iter < maxiter))
    disp(iter)
    E = computeError(xstart,stateC);

    % call it quits if the error increased, w
    magE = norm(E)
    if magE > lastE
        stateC = lastS;
        break
    end
    lastE = magE;
    lastS = stateC;

    J = jacobianShootMotor(xstart,stateC);

    ds = J \ E;

    % going 100% of the dP is causing trouble, going less distance should
    % help
    stateC = updateState(stateC,0.1 .* ds);

    iter = iter + 1;
end

E = computeError(xstart,stateC);

disp('magnitude of final error =')
norm(E)

%%%%%%%%%%%%%%%%%%
% Make some plots
%%%%%%%%%%%%%%%%%%
[T, X] = shootMotor(xstart,stateC);

figure
hold on;
plot(T,X(:,1),'b');
plot(Tobs,Thetaobs,'k');
title('Fit curve');
xlabel('Time (seconds)')
ylabel('Angle (radians)')
hold off;

end

function [J] = jacobianShootMotor(xstart,state0)
```

```
% compute the jacobian of the error vector in the neighborhood of the
% values P0

global Tobs numUPos LPos

[T0,X0] = shootMotor(xstart,state0);

% matrix for storing perturbed values
pX1 = zeros(length(Tobs),numUPos);
deltaP = zeros(numUPos,1);

dP = 1e-2;

p = 1;
for s = 1:length(state0)

    if ~LPos(s)
        stateP = state0;
        deltaP(p) = dP * abs(state0(s));
        %deltaP(p) = 1e-6;
        stateP(s) = stateP(s) + deltaP(p) ;
        [pT, pX] = shootMotor(xstart,stateP);
        pX1(:,p) = pX(:,1); % we only care about storing the positions
        % don't discretize yet, because that would mess up the jacobian
        p = p + 1;
    end
end

J = zeros(length(T0),numUPos);
for i = 1:length(T0)
   for j = 1:numUPos
       J(i,j) = (pX1(i,j) - X0(i)) / deltaP(j);
   end
end

end

function [E] = computeError(xstart,state)
global Thetaobs

[T, X] = shootMotor(xstart,state); % don't need to interpolate, now runs exactly at the sampled points

E = Thetaobs - X(:,1);
end

function [T, X] = shootMotor(xstart,state_)

global state Tobs numPoints

state = state_;

tstart = Tobs(1);
```

```
tend = Tobs(end);

%options = odeset('RelTol',1e-7,'AbsTol',1e-7);
%[T, X] = ode45(@motorEQMotion, Tobs,xstart,options);

T = linspace(tstart,tend, numPoints);
[X] = ode5(@motorEQMotion, T, xstart);
Xint = interpolate(T,X,Tobs);

X = Xint;
T = Tobs;
end

function [snew] = updateState(s, ds)
% can't just do s = s + ds because the actual state and the number of
% parameters we are regressing aren't equal

global LPos

snew = s;
j = 1;
for i = 1:length(s) % this is the full length state
    if ~LPos(i) % if not locked then can update
        snew(i) = snew(i) + ds(j);
        j = j + 1;
    end
end
end

function [xdot] = motorEQMotion(t, x)
% x = [theta; thetadot]

thetadot = x(2); % theta = x(1);

global state staticThreshold
J = state(1);

% need to choose the right value for B based on which direction we are
% turning in
if (thetadot > staticThreshold)
    B = state(3); % Bf = state(3);
elseif thetadot < - staticThreshold
    B = state(4); % Br = state(4);
else
    B = 0; % to avoid spurious forces when we aren't moving
end

T = motorTorque(motorCurrent(t));

thetadotdot = T / J - B * thetadot / J + coulombFriction(T, thetadot) / J;

xdot = [thetadot; thetadotdot];
```

```
end

function [It] = motorCurrent(t)
global Tobs Iobs
It = flatInterpolate(Tobs,Iobs,t);
end

function [torque] = motorTorque(Im)
global state  %Kt = state(2);
torque = state(2) * Im;
end

function [torque] = coulombFriction(motorTorque, thetadot)
% coulombic friction cannont exceed the magnitude of motor torque and
% opposes attempted motion
% models friction as different in each direction
% forces are not equal in each direction
global state staticThreshold staticRatio
Tff = state(5); Tfr = state(6);

if (abs(thetadot) < staticThreshold) % if not moving, friction opposes attempted motion
    % if motortorque does not exceed static friction
    if ((motorTorque < Tff) && (motorTorque >= 0)) || ((motorTorque > -Tfr) && (motorTorque < 0))
        FF = abs(motorTorque);
    elseif (motorTorque > Tff)
        FF = Tff;
    else
        FF = Tfr;
    end
    torque = - staticRatio * FF * sign(motorTorque);

else % if we are moving oppose the actual motion with kinetic friciton
    if thetadot > staticRatio
        torque = -Tff;
    else
        torque = Tfr;
    end
end

end

function [Xout] = interpolate(Tin,Xin,Tout)
% linear interpolation of the desired Tout values

Xout = zeros(length(Tout),1);

i = 1; % keep track of positon in the original
for o = 1:length(Tout)
    while (i <= length(Tin)) && (Tout(o) > Tin(i))
        i = i + 1;
    end
    if (i == 1) % if we asked for a T before the first T
```

```
        dt = Tin(2) - Tin(1);
        slope = (Xin(2) - Xin(1)) / dt;
        dt = Tout(o) - Tin(1);
        Xout(o) = Xin(1) + slope * dt;
    elseif (i > length(Tin))
        % linearly extrapolate - subject to noise, be careful
        dt = Tin(end) - Tin(end - 1);
        slope = (Xin(end) - Xin(end - 1)) / dt;
        dt = Tout(o) - Tin(end);
        Xout(o) = Xin(end) + slope * dt;
    else
        dt = Tin(i) - Tin(i - 1);
        slope = (Xin(i) - Xin(i - 1)) / dt;
        dt = Tout(o) - Tin(i - 1);
        Xout(o) = Xin(i - 1) + slope * dt;
    end
end

end
```

flatInterpolate.m

```
function [Xt] = flatInterpolate(T,X,t)
% not linear interpolation, but rather just returns the previous value
% makes sense for when interpolating the current values

index = inf;
for i = 1:length(T)
    if t < T(i)
        index = i;
        break;
    end
end

if (index == inf)
    Xt = X(end);
elseif (index == 1)
    Xt = 0; % assume initial value of 0
else
    Xt = X(index - 1);
end

end
```

testParameterEstimation.m

```
function [Pest] = testParameterEstimation()
% uses parameterEstimation matlab function to fit the data

J = 0.000041; % initial guesses
Kt = 0.042;
Bforwards = 5.4288e-4 * Kt; % 10*5.4288e-4 * Kt
```

```
Breverse = 6.728054e-4 * Kt; % 0.9*6.728054e-4 * Kt
Tfforwards = 1.75*0.4637 * Kt; %1.75*0.4637 * Kt
Tfreverse = 0.6071722 * Kt; %

% Bforwards = 0.000023725941605;
% Breverse = 0.000020265029631;
% Tfforwards = 0.034057546192782;
% Tfreverse = 0.025282905174523;
%
% J = 0.000043471112655;
% Kt =  0.042000000000000;
% Bforwards =  0.000002; % negative values are no good -0.000011421939988
% Breverse = 0.000002;
% Tfforwards = 0.033926022489524;
% Tfreverse = 0.025288849161596;
%
% J = 0.000046420828236;
% Kt =    0.044062508073865;
% Bforwards =    0.000001678546329;
% Breverse =   0.000002112293049;
% Tfforwards =   0.034540796108616;
% Tfreverse =   0.026025786119668;

% best so far with magnitude of error of 40.2317373
%  0.000044142993118
%   0.042016985993061
%  -0.000018376326419
%   0.000000158849427
%   0.033878935596711
%   0.025319881527519

% best numbers!!! magnitude of error of 27.649 on short data set with
% 2*10^4 points
J = 0.000035780751317;
Kt = 0.032448801388174;
Bforwards = 0.000032339885634;
Breverse = 0.000041519670783;
Tfforwards = 0.022294359068982;
Tfreverse = 0.015632909278982;

% error of 23.4941109 on short data set with 2*10^4 points
% 0.000035051428588
%    0.031449871029185
%    0.000010858577896
%    0.000048592997081
%    0.022438919281204
%    0.014309603079420

% initial error with best numbers above 8.425662612875667e+02
% on medium came up with this 3.259151910806145e+02
% 0.000028710305642
%    0.031941181768441
```

```
%    0.000075360245042
%    0.000024930146369
%    0.019532312877692
%    0.016530560525051

VAL = xlsread('shortSinusoidalInput.xls');

T = VAL(:,2); % delta Time
Theta = VAL(:,3); % need negative
Voltage = VAL(:,4);

% current is related to voltage
I = Voltage;

% % plot the experimental data
% figure
% hold on
% plot(T,Theta,'b');
% plot(T,I,'r');
% xlabel('Time (seconds)');
% legend('Angular Position','Motor Current');
% hold off

J = 6.06724e-5;
B = 4.05562e-5;
Kt = 0.01902;
Bforwards = B;
Breverse = B;
Tf = 0.000145312;
Tfforwards = Tf;
Tfreverse = Tf;

%Pest = parameterSweep(T',Theta',I,P0)
%Pest = nonuniformParameterEstimation(0,T,Theta,I,P0,);
state0 = [J; Kt; Bforwards; Breverse; Tfforwards; Tfreverse];
locked = [ 1; 1;     1    ;     1   ;     1    ;       1     ];
Pest = enchancedParameterEstimation(T,Theta,I,state0,locked)
end

function [X, L] = trimData(X)
L = length(X);
for i = 1:length(X)
    if isnan(X(i))
        L = i - 1;
        break;
    end
end
X = X(1:L);
end
```