Graduate Lab Final Report

Gabriel Colangelo
University at Buffalo
MAE 543

12/11/2022

Contents

INTRODUCTION

This report will discuss the derivation of an open loop transfer function for a SRV02 DC Motor, and the development of PD controller for position control of this motor. The controller was designed in MATLAB and implemented on hardware using Labview. The final performance and tuning of the position controller on hardware will also be discussed.

OPEN LOOP CONTROL SIMULATION AND IMPLEMENTATION

As shown in Lab Manual 2, the sum of the torques acting on an electric motor can be given by:

$$J_l\ddot{\theta}_l = \eta_g K_g \eta_m K_t I_m - \eta_g K_g^2 J_m \ddot{\theta}_l - B_{eq}\dot{\theta}_l \tag{1}$$

where J_l is the load shaft moment of inertia, θ_l is the load position, K_g is the gearbox ratio, η is the efficiency of the motor and the gearbox, K_t is the motor torque constant, B_{eq} is the viscous damping coefficient, and J_m is the motor moment of inertia. Substituting in $J_{eq} = J_l + \eta_g K_g^2 J_m$ and rearranging yields:

$$J_{eq}\ddot{\theta}_l + B_{eq}\dot{\theta}_l = \eta_g K_g \eta_m K_t I_m \tag{2}$$

Applying Kirchhoff's voltage law and assuming the motor inductance to be negligible, the motor armature current is given below, where K_m is the motor back EMF constant, V_m is the motor voltage, and R_m is the armature resistance.

$$I_m = \frac{V_m - K_m\dot{\theta}_m}{R_m} \tag{3}$$

Combining equations 2 and 3 gives:

$$J_{eq}\ddot{\theta}_l + B_{eq}\dot{\theta}_l = \eta_g K_g \eta_m K_t \frac{V_m - K_m\dot{\theta}_m}{R_m} \tag{4}$$

The angular velocity of the motor shaft is related to the angular velocity of the load shaft with the relation:

$$\dot{\theta}_m = K_g\dot{\theta}_l \tag{5}$$

Combining equations 4 and 5 and rearranging gives the following differential equation:

$$J_{eq}\ddot{\theta}_l + (B_{eq} + \frac{\eta_g K_g^2 \eta_m K_t K_m}{R_m})\dot{\theta}_l = \frac{\eta_g K_g \eta_m K_t}{R_m} V_m \tag{6}$$

Taking the Laplace transform of both sides yields the open loop transfer function for a DC motor relating the motor voltage to the load shaft position in the frequency domain. This transfer function is given below.

$$\frac{\theta_l(s)}{V_m(s)} = \frac{\frac{\eta_g K_g \eta_m K_t}{R_m}}{J_{eq}s^2 + s(B_{eq} + \frac{\eta_g K_g^2 \eta_m K_t K_m}{R_m})} \tag{7}$$

This transfer function takes the form of

$$\frac{\theta_l(s)}{V_m(s)} = \frac{b}{a_2 s^2 + a_1 s + a_0} \tag{8}$$

Using the motor parameters from Lab Manual 2 gives the following values

Table 1: System Parameters

b	0.1282
a_2	0.002
a_1	0.0729
a_0	0

The block diagram and implementation of the open loop motor model in Labview is shown below on the next page, along with the output for a square wave voltage input.

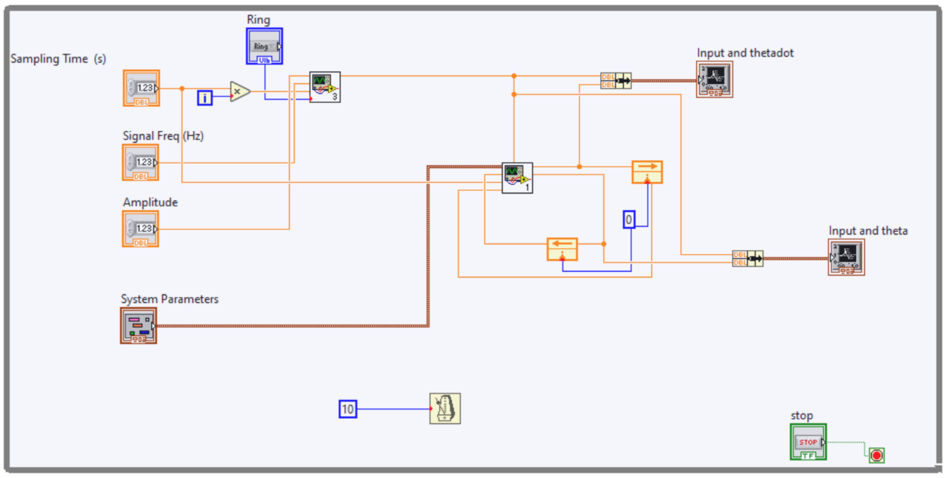

Figure 1: Labview Motor Model Block Diagram

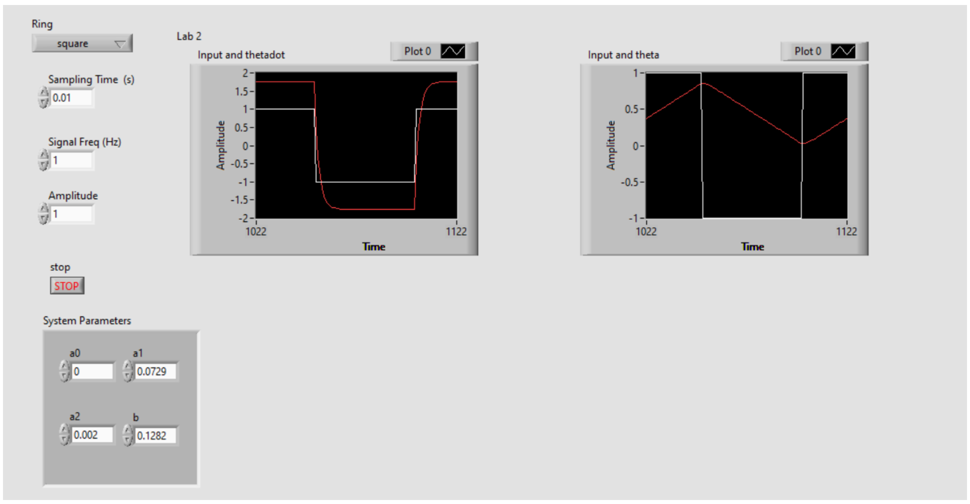

Figure 2: Labview Front Panel Sim

The implementation is as follows. The front panel takes in the sampling period, signal type, signal amplitude, and signal frequency, as well as the system parameters. The system parameters are fed into a Motor SubVI, which is just a block diagram implementation of equation 6 in terms of the system parameters. A signal generator subVI outputs a voltage signal using the input parameters from the front panel. The voltage signal from the signal generator is also sent as an input to the motor subVI, which outputs the load shaft angular acceleration, $\ddot{\theta}_l$. The load shaft acceleration is then numerically integrated using a first order finite difference implementation to obtain the load shaft velocity, which is similarly integrated to obtain the load shaft position. To perform the numerical integration, the sampling period input from the front panel is used as the Δt in the finite difference integration equation below.

$$\dot{\theta}_n = \dot{\theta}_{n-1} + \ddot{\theta}_n \Delta T \tag{9}$$

The load shaft position and velocity is then plotted on the front panel. To perform the simulation, the user simply inputs the desired signal and system parameters and chooses one of the three signal types from the drop down menu. Then the system response appears in real time on the front panel until the system is stopped. The subVI's for the signal generator and motor simulation are shown below.

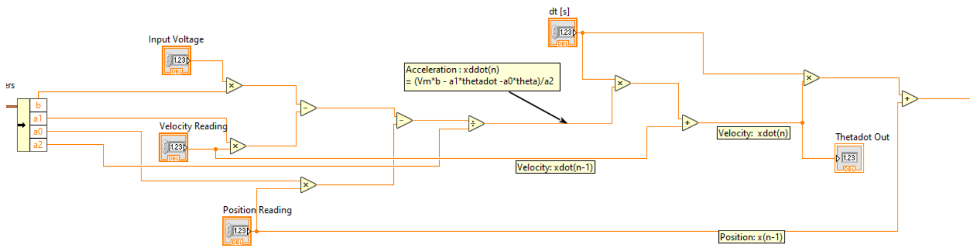

Figure 3: Motor Simulation SubVI

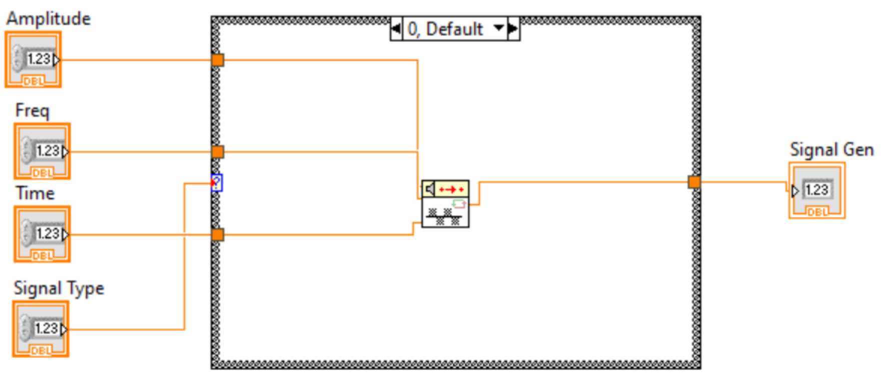

Figure 4: Signal Generator SubVI

After the motor model was implemented on Labview, the open loop motor simulation was implemented on hardware using the SRV02 motor, a NI-terminal board, and a UPM power amplifier. The block diagram for the Labview VI used for this hardware implementation is shown on the next page.

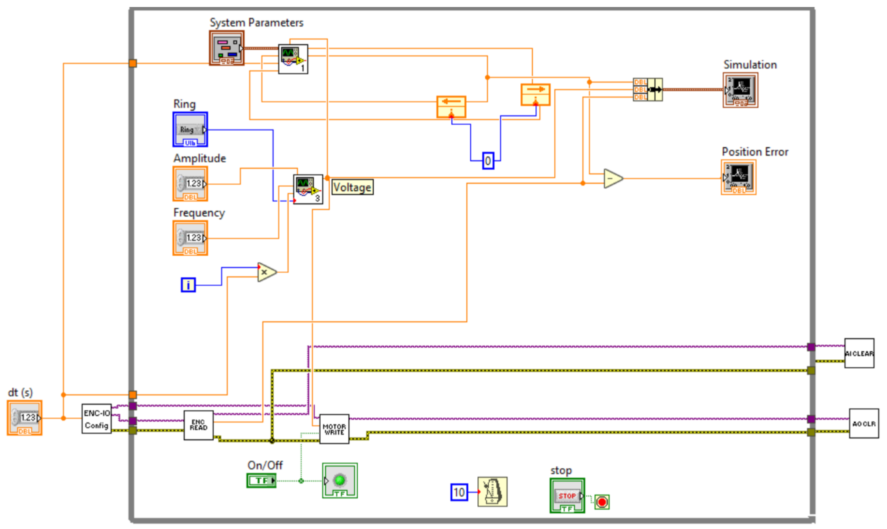

Figure 5: Open Loop Labview Implementation

The hardware implementation is as follows. The amplitude, frequency, and signal type of the input voltage is given from the front panel, along with the motor parameters. These values are fed into the signal generator which outputs a voltage signal to both the motor simulation subVI and the motor write subVI, which generates the periodic signal into the motor. The motor position from the simulation is then compared to the actual motor shaft position, which is read in from the encoder read subVI. The open loop VI generates two figures. One of the error between the simulation and actual motor position, and another of the motor simulated position, motor actual position, and the input voltage. This VI runs in real time, and will run until stopped. Below are figures for the Labview output for a sinusoidal, triangle, and square wave 1V input voltage at 0.8Hz, as well as the error between the simulated motor position and the measured motor position.

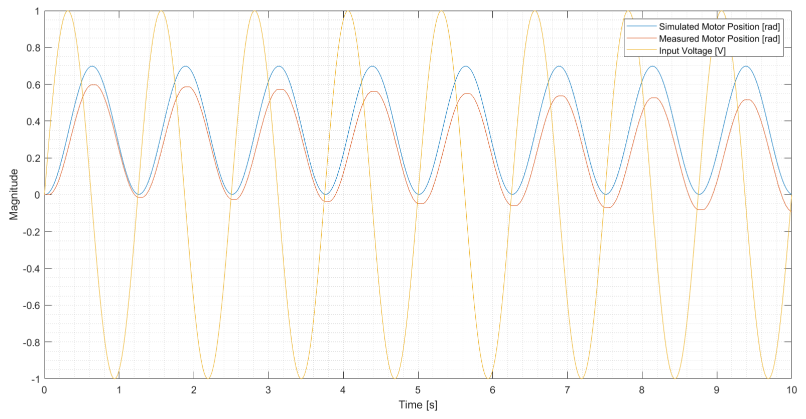

Figure 6: Open Loop Sinusoidal Input Voltage

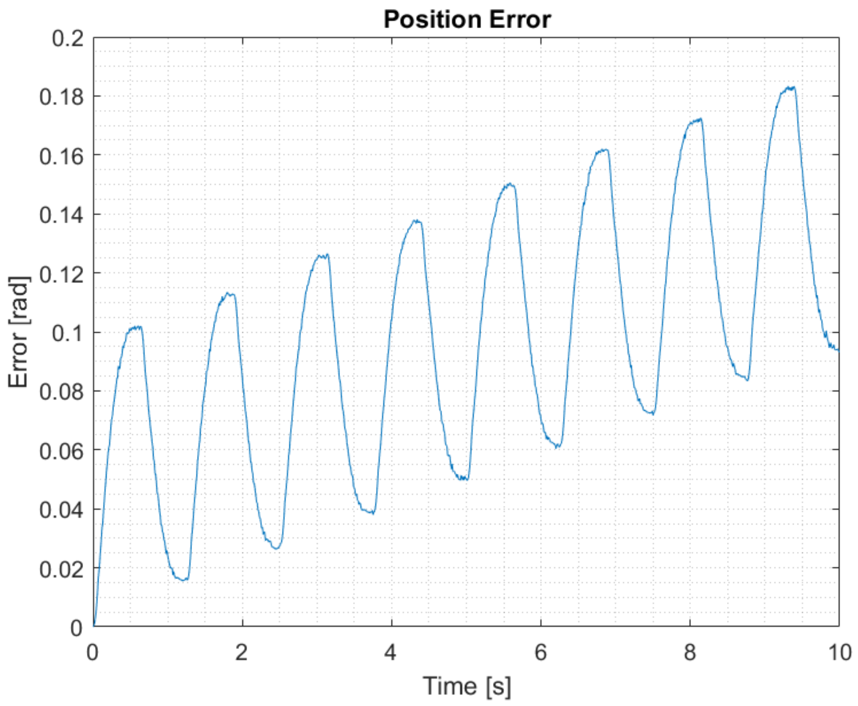

Figure 7: Motor Position Error Sinusoidal Input

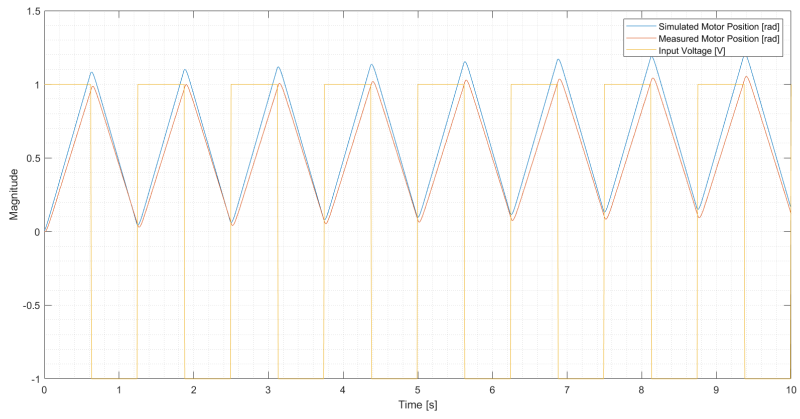

Figure 8: Open Loop Sinusoidal Square Voltage

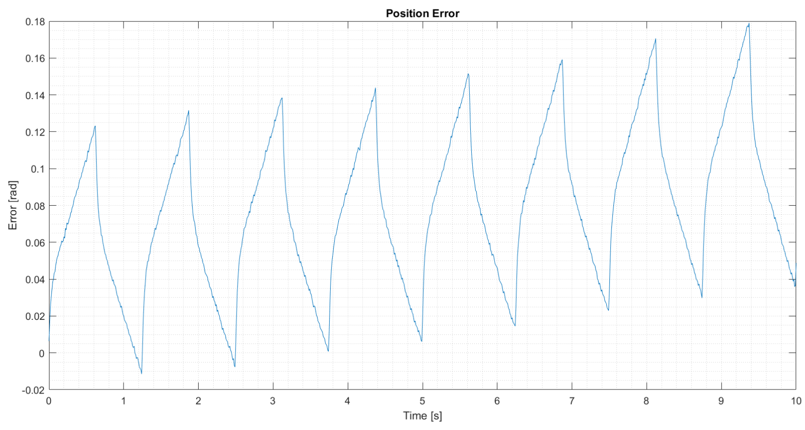

Figure 9: Motor Position Error Square Input

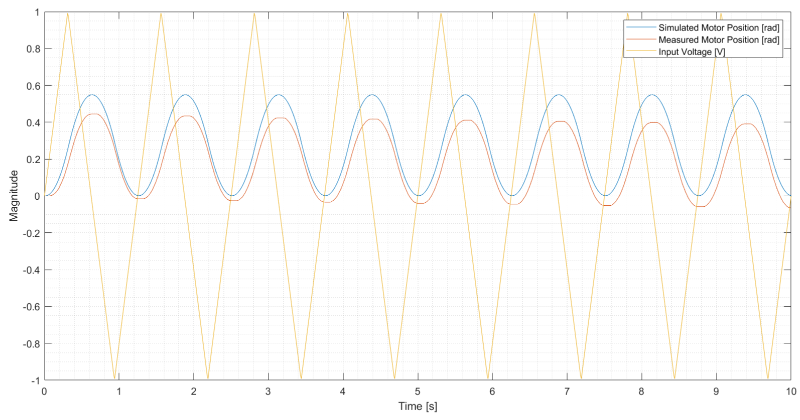

Figure 10: Open Loop Triangle Input Voltage

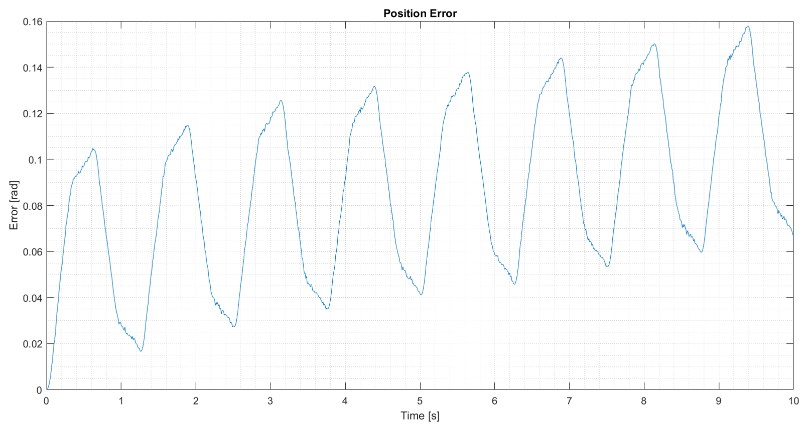

Figure 11: Motor Position Error Triangle Input

From the error plots of all three signals, it is clear that there is substantial error between the simulated motor position and the measured motor position. This difference could be due to a number of things. The simulation uses a linear motor model, and there nonlinearities in the true motor dynamics that are not accounted for. Such nonlinearities are mechanical backlash and dead-zones caused by coulomb friction. The model also assumes the motor armature inductance to be negligible, which further reduces the fidelity of the model. There also may be uncertainty in the position measurements obtained from the encoder, which can add to the position error. It is also observed that the error also increases over time, which may be due to the encoder not zeroing out after the completion of each periodic signal. The motor measured position and simulated position have the same shape and appear to be in phase, the only difference appears to be in the magnitude which is a result of the previously discussed modeling errors.

Position Control Simulation and Implementation

A PD controller was developed to perform position control on the motor load shaft. The PD controller was chosen because the plant is a type 1 system, so it will have zero steady state for a step input. The closed loop design specifications require a overshoot of less than or equal to 5% and a peak time of approximately 0.1 seconds. A second order system, like the one shown in equation 7, can be described by a transfer function parameterized by its damping ratio, ζ and natural frequency ω_n. This transfer function is given by:

$$G(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{10}$$

The minimum damping ratio and natural frequency needed to satisfy the design requirements can be found by solving the following two equations.

$$t_p = \frac{\pi}{\omega_d} = \frac{\pi}{\omega_n\sqrt{1-\zeta^2}} \tag{11}$$

$$M_p = e^{\frac{-\zeta\pi}{\sqrt{1-\zeta^2}}} \tag{12}$$

It can be determined from equations 11 and 12, that the minimum damping ratio is $\zeta = 0.69$ and the minimum natural frequency is $\omega_n = 43.4$ [rad/s]. The PD controller transfer function is given by:

$$C(s) = K_p + K_d s \tag{13}$$

Using equations 13 and 8, the closed loop transfer function was calculated to be:

$$G_{CL} = \frac{\theta_l}{\theta_{ref}} = \frac{\frac{b}{a_2}(K_p + K_d s)}{s^2 + s(\frac{a_1+bK_d}{a_2}) + \frac{b}{a_2}K_p} \tag{14}$$

Comparing the coefficients of the denominators of equations 14 and 10 yields two equations that can be used to calculate control gains K_p and K_d.

$$2\zeta\omega_n = \frac{a_1 + bK_d}{a_2} \tag{15}$$

$$\omega_n^2 = \frac{b}{a_2}K_p \tag{16}$$

This method of comparing the denominators of the transfer functions is known as pole placement. Pole placements allows one to choose where the poles of the closed loop transfer function reside. Solving equations 15 and 16 with the minimum damping ratio and natural frequency results in the control gain values of K_d **= 29.39 [V/rad]** and K_p **= 0.366 [V/rad/s]** . With these control gains, the closed loop poles are found to reside at $s_{1,2} = -29.96 \pm 31.42i$, which corresponds to a damping ratio of 0.69 and a natural frequency of 43.4 [rad/s]. A unit step input voltage was applied to the closed loop system given by equation 14 using the control gain values discussed above. The results are shown on the following page.

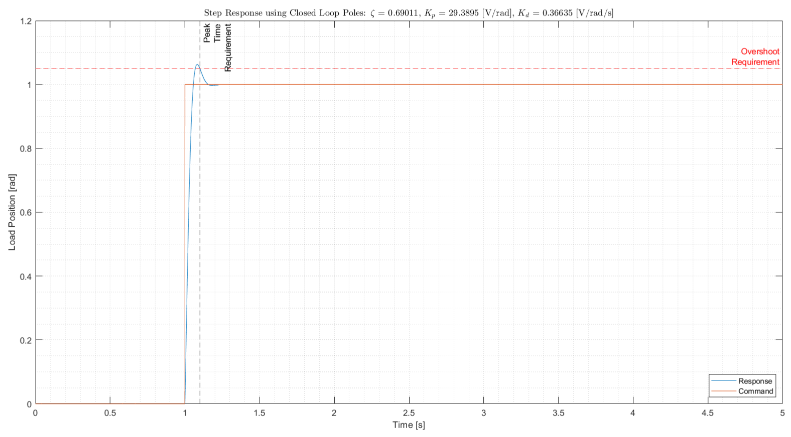

Figure 12: Closed Loop Step Response - Original Tuning

From the step response figure, it is clear that the overshoot requirement is not met as the maximum overshoot is approximately **6%**. The peak time is also around **0.08 seconds**, so slighty off of the 0.1 second requirement. The reason the simulated response differs from the expected dynamics of the pole placement tuning, is because the pole placement method assumes the numerators of equations 10 and 14 to be the same. Clearly they are not the same, as the numerator of equation 14 contains a zero at s = $\frac{-K_p}{K_d}$. The zero in the numerator prevents the closed loop transfer function from having the same form as equation 10, which means that the closed dynamics will be different and that is exactly what is observed. To satisfy the design constraints, the control gains were iteratively tuned until the desired response was achieved. The tuning that achieves this is given by $K_p = 27.94$ [V/rad] and $K_d = 0.58$ [V/rad/s]. The closed loop step response with this new tuning is shown below.

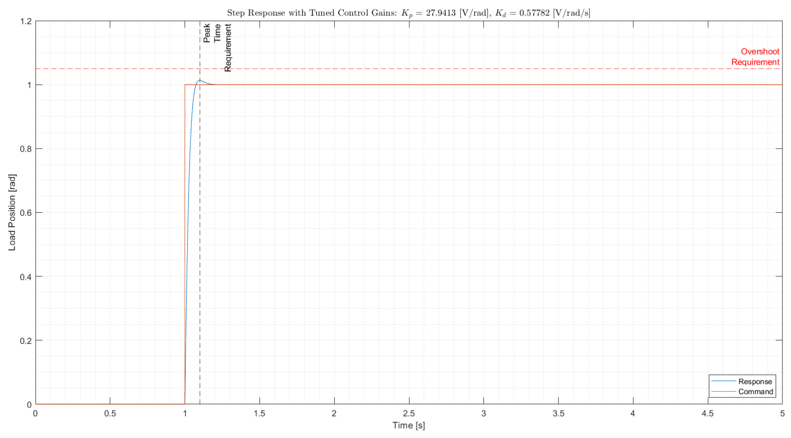

Figure 13: Closed Loop Step Response - New Tuning

The previously simulations were run using Simulink. The closed loop Simulink model is shown below. All relevant MATLAB code is shown in the appendix.

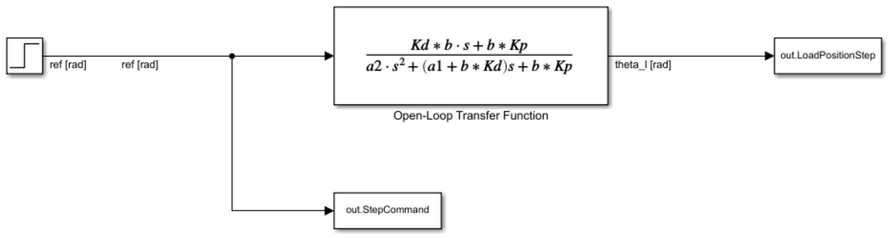
ref [rad]
ref [rad]
$\frac{Kd*b\cdot s+b*Kp}{a2\cdot s^2+(a1+b*Kd)s+b*Kp}$
Open-Loop Transfer Function
theta_l [rad]
out.LoadPositionStep
out.StepCommand

Figure 14: Closed Loop Simulink Model

With a suitable tuning found in simulation, the PD position controller was now implemented on hardware using Labview. The block diagram for the main Labview VI is shown below.

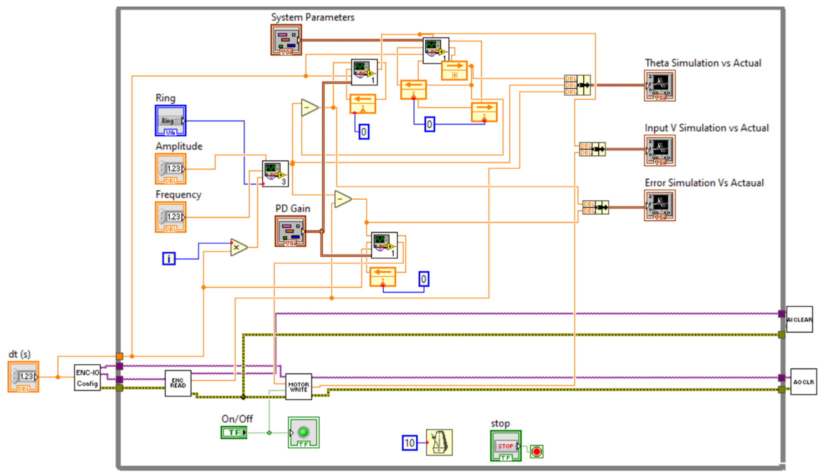

Figure 15: Position Control Main Labview VI

The PD controller gains are pulled from the front panel of the VI along with the signal and system parameters in a similar fashion to what was discussed prior. The PD controller is actually implemented with a filtered derivative rather than a pure derivative to help minimize high frequency noise. A low pass filter VI was used to implement the filtered derivative in the PD controller. Two PD controller subVI's were used, one acting on the motor simulation VI and the other acting on the actual hardware. The desired reference position for both the simulation and physical motor was generated using the signal generator VI. The simulation position output and the encoder position were subtracted from the reference position to generate the error for the motor simulation subVI and the actual motor error signal. The error is fed into its respective PD controller to generate a voltage signal which will drive the motor. The VI displays three figures. The first being the simulated vs actual position. The second figure is the simulated input voltage vs the actual input voltage, which is read from the motor write subVI. The last figure shows the actual vs simulated error signal into each PD controller. Running the VI with the tuned set of control gains on hardware produces the following results with a 0.1 rad square wave reference input at 0.8Hz.

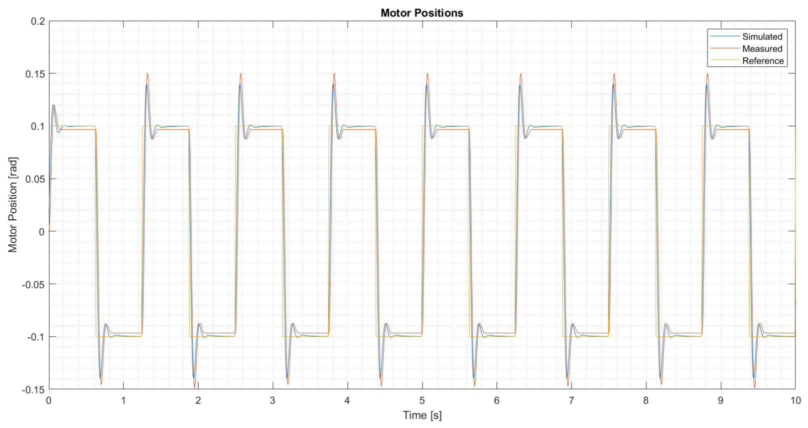

Figure 16: First Tuning on Hardware

Clearly this response does not meet the design specifications. The response has nearly 50% overshoot and has non zero steady state error. This response does not match the simulated output. This is likely due to the same physical issues that plagued the open loop response in the previous section. Mechanical backlash, deadzones, noise in the voltage signal, and other model simplifications aren't accounted for in the simulation. It should also be noted that the Labview simulated response isn't exactly the same as the MATLAB simulation. This is because the PD controller was implemented with a pure derivative in MATLAB while the controller was implemented as a filtered derivative in Labview. To meet the design specifications, the control gains had to once again be iteratively fine tuned. The final tuned control gains implemented on hardware were chosen to be K_p = **10 [V/rad]** and K_d = **0.1 [V/rad/s]**. These control gains were vastly different than both set of gains found in the MATLAB simulations. This was because our model was too low fidelity to produce good control gains that could overcome the physical non-linearities and disturbances. Using the final tuned control gains, the following responses were generated for 0.4 Hz signals of square, sinusoidal, and triangle inputs of 0.5 radians. The control input voltage is also shown.

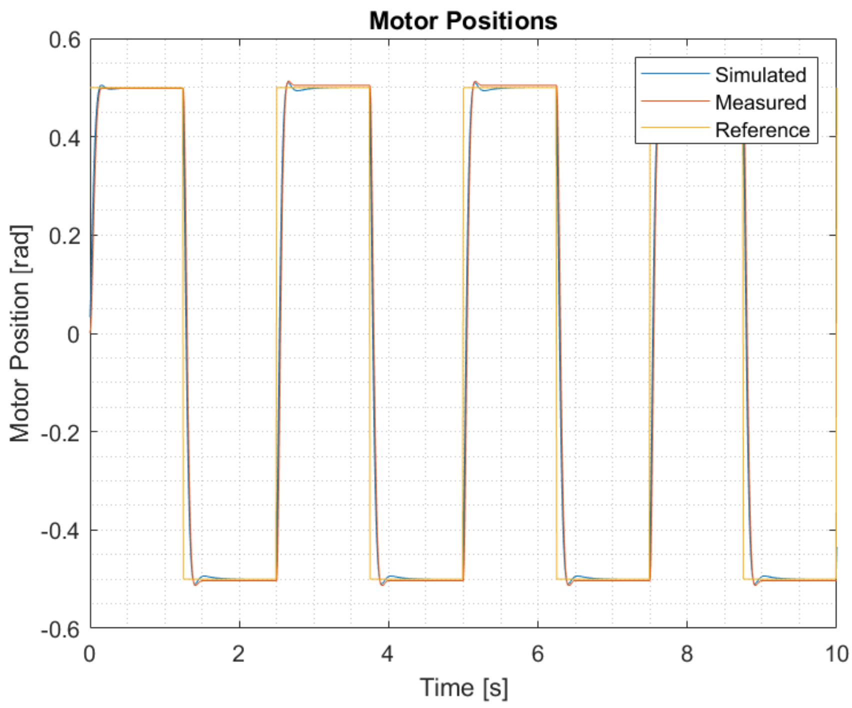

Figure 17: Fine Tuned Square Wave

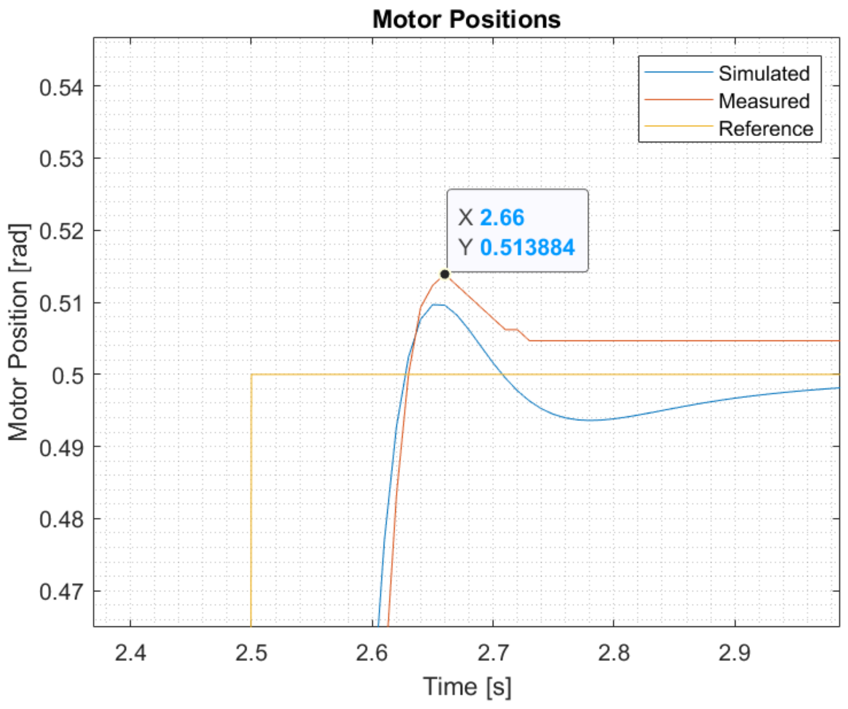

Figure 18: Fine Tuned Square Wave - Zoomed In

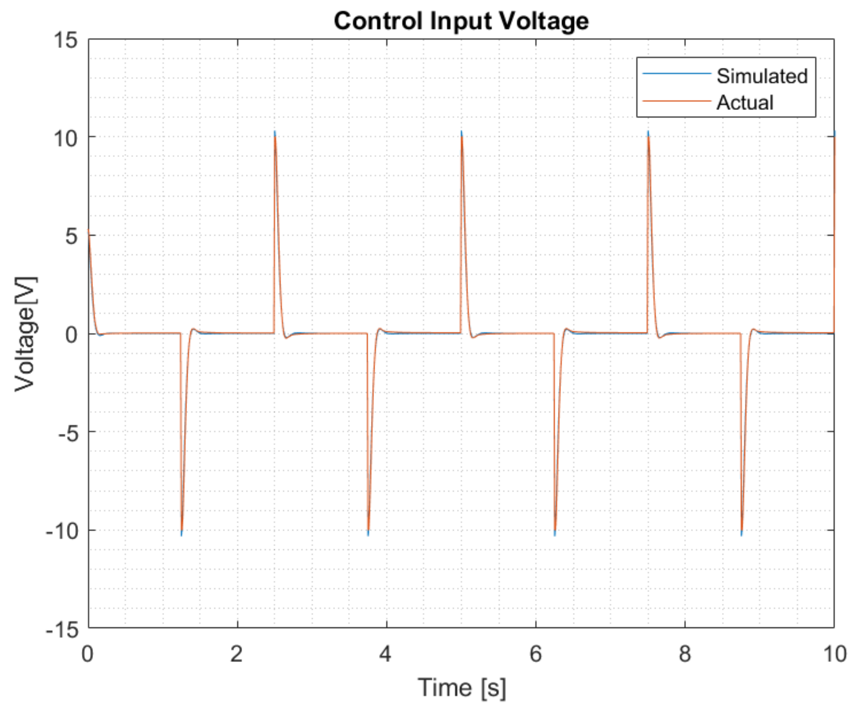

Figure 19: Square Wave Input Voltage

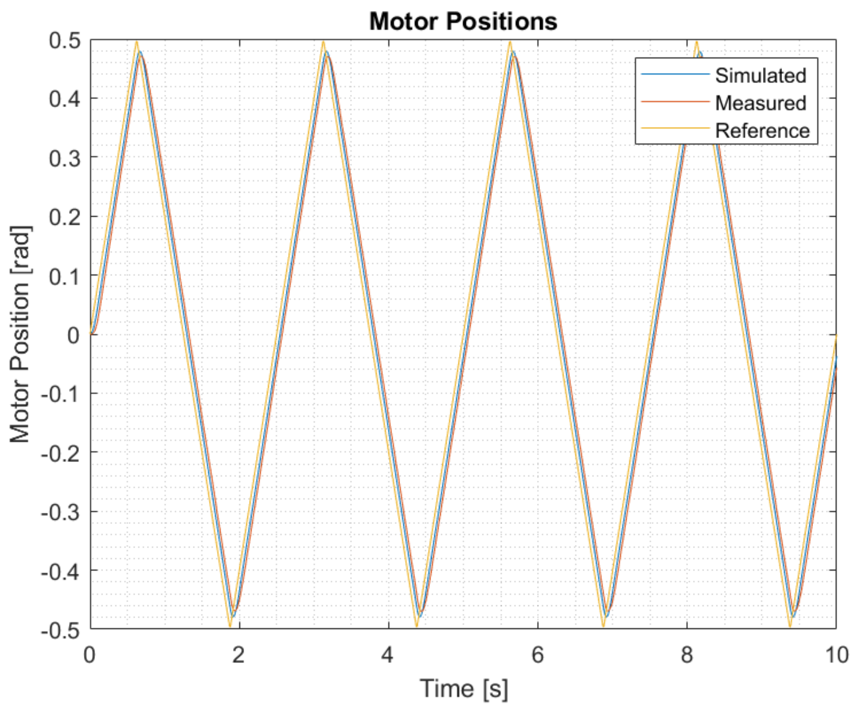

Figure 20: Fine Tuned Triangle Wave

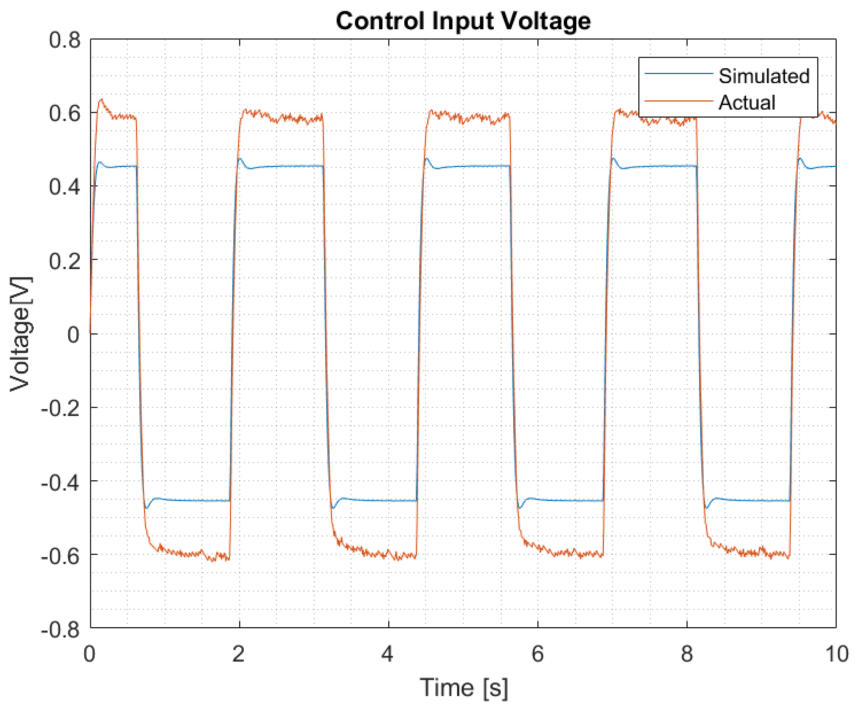

Figure 21: Triangle Wave Input Voltage

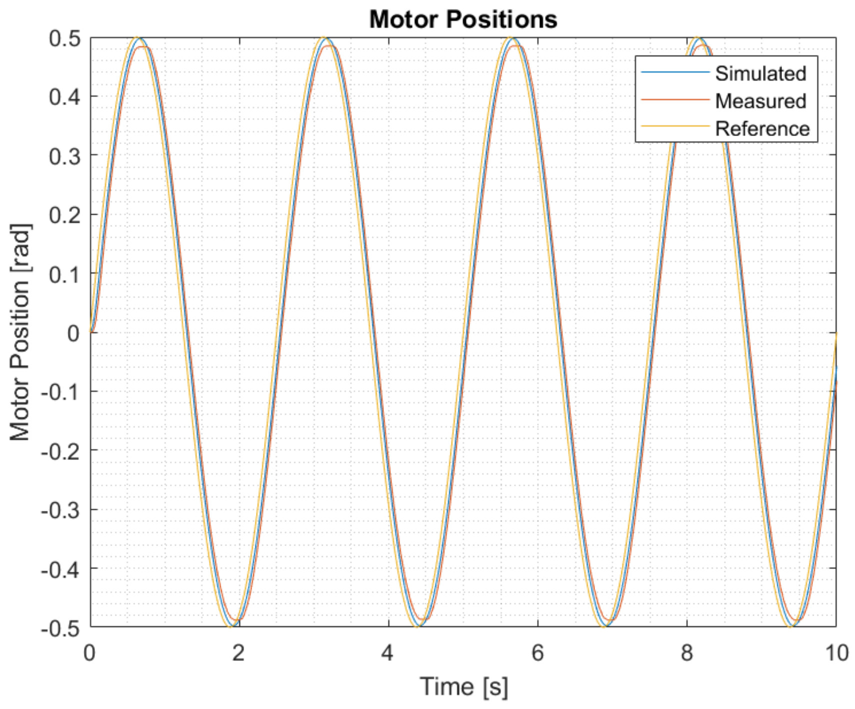

Figure 22: Fine Tuned Sine Wave

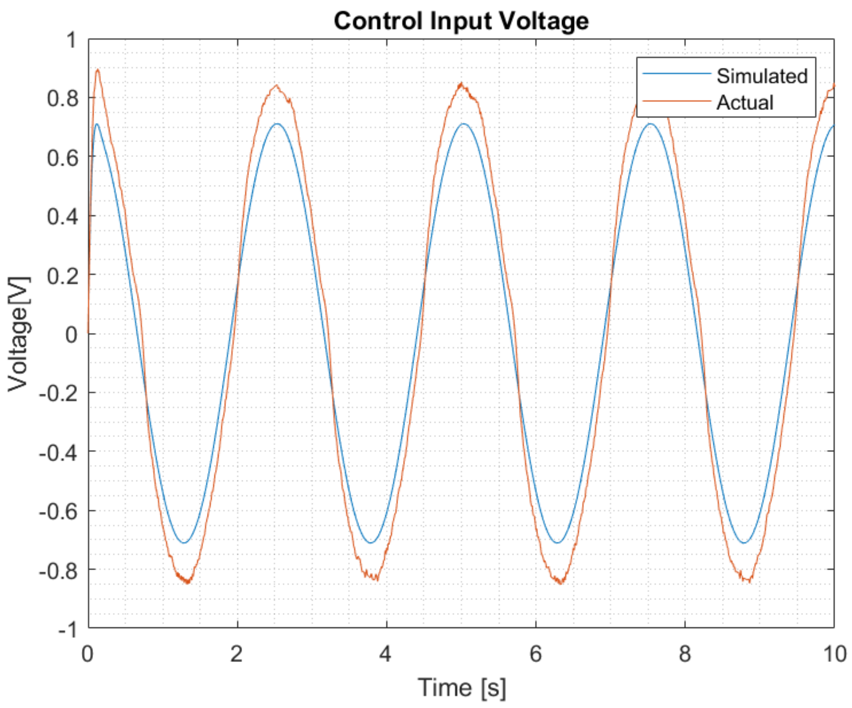

Figure 23: Sine Wave Input Voltage

From square wave response figure above, it is observed that the position response with the fined tuned controller has an **overshoot of roughly 2% with a peak time of approximately 0.15 seconds. Therefore, we can say that our controller has achieved the control objectives**. If an additional design requirement of zero steady state error had to be satisfied, a new controller would need to be developed. From the triangle and sinusoidal wave responses, it is seen that there is constant error between the reference and measured signals. The triangle and sine wave reference signals can be constructed from ramp inputs and parabolic inputs respectively. The open loop transfer function for the system can be found by multiplying equation 8 by equation 13, where $a_0 = 0$. Carrying through this multiplication shows that this system is clearly a type 1 system, with a single pole at the origin, as shown below.

$$G_{OL} = G(s)C(s) = \frac{b(K_p + K_d s)}{a_2 s^2 + a_1 s} \tag{17}$$

Type 1 systems have zero steady state error for a step input, but have non-zero steady state error for a ramp input and unbounded error for a parabolic input. Therefore, for a design requirement of zero steady state error, a PD controller is not sufficient. For zero steady state error in response to a ramp input and bounded errors for a parabolic input, the open loop transfer function must be type 2, i.e. have two poles at the origin. To achieve this, a PID controller can be used as the integrator will place an additional pole at the origin. The transfer function for the theoretical PID controller and the corresponding open loop transfer function is shown below, where it is clearly shown that the PID controller makes this a type 2 system.

$$C(s) = K_p + K_d s + \frac{K_i}{s} \tag{18}$$

$$G_{OL} = G(s)C(s) = \frac{b(K_d s^2 + K_p s + K_i)}{s(a_2 s^2 + a_1 s)} \tag{19}$$

Conclusions

Throughout this work, it was repeatedly found that mathematical modeling gave a good baseline of what to expect from hardware, but the results were nowhere near perfect. The control gains found using models had to be fine tuned repeatedly to achieve the desired system performance on hardware. The control gains that were initially determined using pole placement methods, didn't behave as expected as it wasn't a perfect second order system. This was discovered using simulation, and luckily not on hardware as this could've damaged the motor potentially. With a better set of gains found, the gains were then tuned on the hardware to makeup for the model deficiencies. The open loop hardware performance also didn't line up perfectly with what was seen on hardware, likely due to a variety of physical disturbances and unmodeled pehnomena in the plant. If one was to design a controller using solely a model, the model would need to be extremely high fidelity and well vetted.Even so this is not recommend as it is always a good thing to verify your controller on hardware, assuming it poses no risk to the hardware. The biggest issues faced throughout testing was almost always hardware related. In testing, issues with the power supply and DAQ board plagued the experiment but the TA was very helpful in resolving these issues. All relevant MATLAB code, Simulink models and Labview VI's are available in the attached zip file.