

DC Motor With Inertia Disk
Model Development, Proportional Controller,
and State Feedback Controller with Full State
Estimator

© 2002-2005

Erick L. Oberstar & Oberstar Consulting

Table of Contents

| | |
|--|----|
| Table of Contents | 1 |
| Table of Figures | 1 |
| Introduction..... | 2 |
| Model Development..... | 3 |
| State Space Method: | 4 |
| Laplace Method: | 5 |
| Model Simulation and Physical System Response | 7 |
| Open Loop Frequency Responses..... | 8 |
| Open Loop Velocity Responses..... | 10 |
| Model Parameter Measurement - Locked Rotor Testing..... | 11 |
| Controller Design..... | 11 |
| Proportional Controller | 11 |
| State Feedback Design..... | 15 |
| Observer Design (State Estimator) | 16 |
| State Feedback with Estimated States..... | 17 |
| Conclusion | 20 |
| Bibliography | 21 |
| Appendix A - Matlab® Source Code..... | 22 |
| Appendix B - Proportional Controller C Source Code | 30 |

Table of Figures

| | |
|---|----|
| Figure 1 - Maxon Model 110953 Brush Motor with Inertia Disk | 2 |
| Figure 2 – Schematic Diagram of Integrated Amp, Motor, & Disk Model..... | 2 |
| Figure 3 – SS Method: Continuous Position System Bode Plot..... | 8 |
| Figure 4 - Laplace Method: Continuous Position System Bode Plot | 8 |
| Figure 5 - SS Method: Continuous Velocity System Bode Plot..... | 9 |
| Figure 6 – Laplace Method: Continuous Velocity System Bode Plot..... | 9 |
| Figure 7 - SS Method: Continuous Open Loop Velocity Step Response | 10 |
| Figure 8 - Laplace Method: Continuous Open Loop Velocity Step Response..... | 10 |
| Figure 9 - Physical System: Open Loop Velocity Step Response | 11 |
| Figure 10 - U.W. Madison Mechatronics Lab’s custom Atmel AVR embedded system..... | 12 |
| Figure 11 - Closed Loop Discrete Proportional Controller | 12 |
| Figure 12 – Discrete Closed Loop P Control Bode Plot..... | 13 |
| Figure 13 – Discrete Closed Loop P Control Step Response | 13 |
| Figure 14 - Discrete P Control Physical System Step Response | 14 |
| Figure 15 - Discrete P Control Larger Step Response Scaled To A Unit Step Response | 14 |
| Figure 16 – Discrete State Feedback Block Diagram with Reference Input | 15 |
| Figure 17 – Discrete State Feedback Controller Step Response (Assumes Full State Feedback)..... | 16 |
| Figure 18 - Discrete Observer Block Diagram with Reference Input | 16 |
| Figure 19 – Observer State Error V.S. Time | 17 |
| Figure 20 – Discrete State Feedback Controller with Full Observer (Controller-Estimator)..... | 17 |
| Figure 21 – Controller-Estimator States, Estimated States, & State Error | 18 |
| Figure 22 – Discrete Controller-Estimator Bode Plot (SR=300Hz)..... | 19 |
| Figure 23 – Discrete Controller-Estimator Step Response (SR=300Hz)..... | 19 |

Introduction

A new motor and inertia disk system has been added to the U. W. Madison, Mechanical Engineering Departments “pencil sharpener” motion control experiment platform. This system utilizes a Maxon A-Max 26 6 watt, DC Brush motor, model 110953 with an inertia disk for the load. The motor has an integrated 500 count per revolution encoder for measuring position. Before this addition to the system can be utilized in the classroom the system model for the motor and inertia disk must be developed and verified. The motor is shown in Figure 1 with key parameters shown in Table 1. A schematic diagram of the motor model is shown in Figure 2.

Figure 1 – Maxon Model 110953 Brush Motor with Inertia Disk

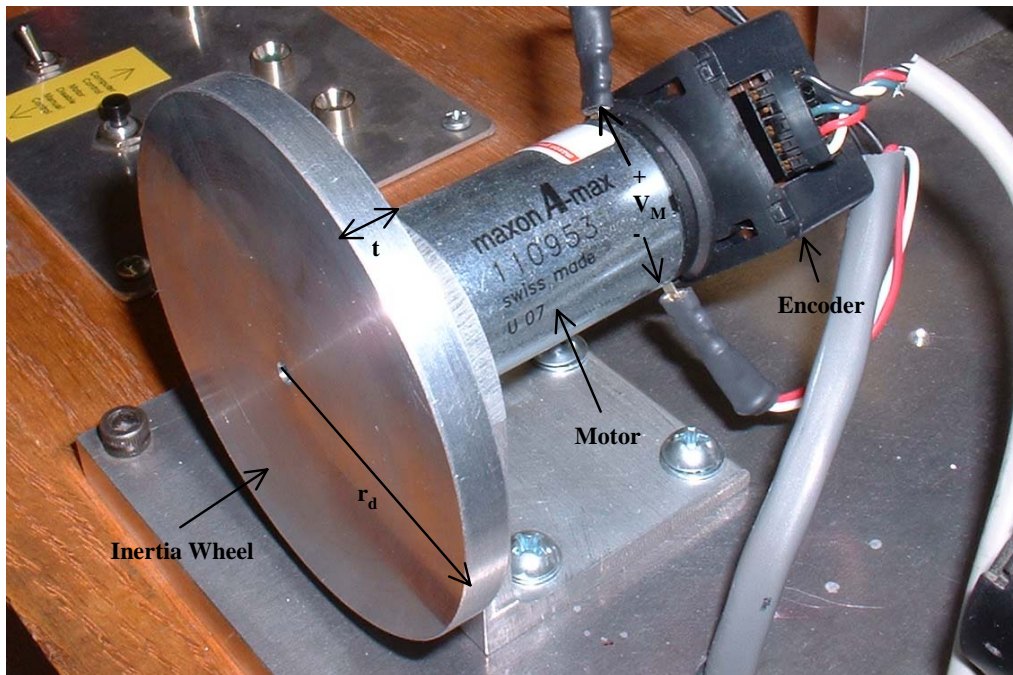


Figure 2 – Schematic Diagram of Integrated Amp, Motor, & Disk Model

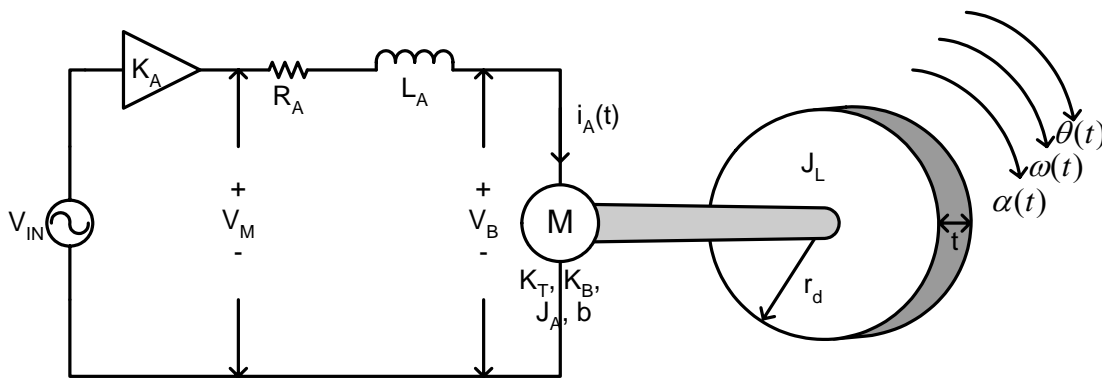


Table 1 – Maxon Model 110953 Brush Motor Parameters

| | |
|--------------------------|---|
| $V_{IN}(t)$ | Amplifier Input Voltage (Volts) |
| $K_A = 4$ | The Servo Amplifier Voltage Gain (Volts/Volts) |
| $V_M(t)$ | Motor Input Voltage (Volts) |
| $R_A = 30$ | Motor Armature Coil (Terminal) Resistance (Ohms) |
| $L_A = 0.001690$ | Motor Armature Coil (Terminal) Inductance (Henrys) |
| $i_A(t)$ | Motor Armature Coil Current (Amps) |
| $V_B(t)$ | Motor Back Electromotive Force (EMF) (Volts) |
| $K_T = 0.02830$ | Motor Torque Constant (Nm/A) |
| $K_B = 0.02830$ | Motor Back EMF (Speed) Constant (Volt-Sec/Radian) |
| $J_A = 1.06 \text{ e-}6$ | Motor Armature Inertia (Kg-m ²) |
| $b = 5.8\text{e-}6$ | Viscous Damping Coefficient (Kg-m ² /Sec) |
| J_L | Load Inertia (Kg-m ²) (Calculated from Disk Parameters) |
| $r_d = 0.0254$ | Load Disk Radius (m) |
| $t = 0.00635$ | Load Disk Thickness (m) |
| $\rho = 2702$ | Mass Density of the Disk (Aluminum) (Kg/m ³) |
| $T(t)$ | Motor Shaft Torque (Nm) |
| $\alpha(t)$ | Motor Shaft Angular Acceleration (Radians/Sec ²) |
| $\omega(t)$ | Motor Shaft Angular Velocity (Radians/Sec) |
| $\theta(t)$ | Motor Shaft Angular Position (Radians) |
| $K_{DA} = 0.0390625$ | D/A Converter Gain (Volts/Count) (10V/2 ⁸) |
| $K_{ENC} = 318.3099$ | Encoder Counter Gain (Counts/radian) (2000Counts/2 π Radians) |

Model Development

The governing equations of this system are:

Electrical System Equations - Using Kirkoff's Voltage and Current Laws (KVL/KCL)

$$V_M(t) = K_A V_{IN}(t)$$

$$V_M(t) = i_A(t)R_A + L_A \frac{di_A(t)}{dt} + V_B(t)$$

$$V_B(t) = K_B \omega(t) = K_B \frac{d\theta(t)}{dt}$$

Mechanical System Equations

$$T(t) = K_T i_A(t)$$

$$T(t) = J_T \alpha(t) + b \omega(t) = J_T \frac{d\omega(t)}{dt} + b \frac{d\theta(t)}{dt} = J_T \frac{d^2\theta(t)}{dt^2} + b \frac{d\theta(t)}{dt}$$

$$J_T = J_A + J_L \quad J_L = \frac{1}{2} M r_d^2 = \frac{1}{2} \rho \pi r_d^4 t \quad J_T = J_A + \frac{1}{2} \rho \pi r_d^4 t$$

The equation for J_T assumes a stiff coupling between the motor shaft and disk (i.e. the shaft twist is negligible)

Either state space or Laplace methods can be used to represent the systems model. Both are equivalent. However for higher dimensional multiple input multiple output (MIMO) systems, the state space techniques may become more attractive because of the ease of solving the multiple system equations simultaneously.

Either of these systems are continuous time representations. For implementation they are transformed into discrete time systems because the motor position control must be implemented in a real world discrete system. The system chosen in this case, is based on a custom Atmel AVR embedded system with encoder counter and analog output capabilities. The real world system adds additional blocks into the closed loop position control model. Additional gain factors resulting from unit conversions must be added into the closed loop model. The D/A converter /analog buffer circuitry provides a gain factor of

$$K_{da} = \frac{10(\text{Volts})}{2^8(\text{counts})} = \frac{10}{256} = 0.0390625(\text{Volts/Count}).$$

The continuous model provides position feedback directly in units of radians. However position feedback in the real system is provided in counts. The encoder / encoder counter combination for position feedback provides a gain of

$$K_{enc} = \frac{2000(\text{Counts})}{2\pi(\text{Radians})} = 318.3099(\text{Counts/Radian}).$$

These parameters were included in the models to make it easier to verify the plots of the physical systems response to the simulated systems.

State Space Method

The state space model takes the form: $\dot{X}(t) = AX(t) + Bu(t), y(t) = CX(t) + Du(t)$. To solve using state space methods first assign state variables. Let $x_1(t) = i_A(t), x_2(t) = \theta(t), x_3(t) = \omega(t)$. Rearrange the continuous systems equations, solving for the derivatives and the output y:

$$\dot{x}_1(t) = \frac{di_A(t)}{dt}, \dot{x}_2(t) = \frac{d\theta(t)}{dt}, \dot{x}_3(t) = \frac{d\omega(t)}{dt}.$$

This yields the following equations:

$$\frac{di_A(t)}{dt} = \frac{R_A i_A(t)}{L_A} + \frac{K_B \omega(t)}{L_A} + \frac{K_{DA} K_A u(t)}{L_A}, \quad \frac{d\theta(t)}{dt} = \omega(t), \quad \frac{d\omega(t)}{dt} = \frac{K_T i_A(t)}{J_T} + \frac{b\omega(t)}{J_T},$$

$$y(t) = K_{ENC} \theta(t)$$

These equations can be put directly into the state space form (Messner, Tilbury 1999):

$$\dot{X}(t) = AX(t) + Bu(t), y(t) = CX(t) + Du(t)$$

where:

$$\dot{X}(t) = \begin{bmatrix} \frac{di_A(t)}{dt} \\ \frac{d\theta(t)}{dt} \\ \frac{d\omega(t)}{dt} \end{bmatrix}, A = \begin{bmatrix} \frac{R_A}{L_A} & 0 & \frac{K_B}{L_A} \\ 0 & 0 & 1 \\ \frac{K_T}{J_T} & 0 & \frac{b}{J_T} \end{bmatrix}, B = \begin{bmatrix} \frac{K_A}{L_A} \\ 0 \\ 0 \end{bmatrix}, C = [0 \quad 1 \quad 0], D = 0$$

To aid in the verification of this model from the physical system the D/A converter and encoder gains are added into the continuous model. These terms appear as pure gain in the B & C matrices.

$$\dot{X}(t) = \begin{bmatrix} \frac{di_A(t)}{dt} \\ \frac{d\theta(t)}{dt} \\ \frac{d\omega(t)}{dt} \end{bmatrix}, A = \begin{bmatrix} \frac{R_A}{L_A} & 0 & \frac{K_B}{L_A} \\ 0 & 0 & 1 \\ \frac{K_T}{J_T} & 0 & \frac{b}{J_T} \end{bmatrix}, B = \begin{bmatrix} \frac{K_{DA}K_A}{L_A} \\ 0 \\ 0 \end{bmatrix}, C = [0 \quad K_{ENC} \quad 0], D = 0$$

Substituting in the parameters from Table 1, the model becomes:

$$\dot{X}(t) = \begin{bmatrix} \frac{di_A(t)}{dt} \\ \frac{d\theta(t)}{dt} \\ \frac{d\omega(t)}{dt} \end{bmatrix}, A = \begin{bmatrix} -1.775147 \times 10^4 & 0 & -16.766977 \\ 0 & 0 & 1 \\ 2.304944 \times 10^4 & 0 & 0.472391 \end{bmatrix}, B = \begin{bmatrix} 92.455621 \\ 0 \\ 0 \end{bmatrix}, C = [0 \quad 2000 \quad 0], D = 0$$

Before going any further into a controller design, the state space representation must be checked for controllability and observability. If either the controllability matrix $C_\Sigma = [B \quad AB \quad A^2B]$ or the observability matrix $O_\Sigma = [C \quad CA \quad CA^2]^T$ is rank deficient the system must be modified. If the controllability matrix is rank deficient additional inputs/control signals must be added to make the system controllable. Similarly if the observability matrix is rank deficient additional sensors must be added to the system to make it observable. For this problem both C_Σ and O_Σ are full rank. This makes it a worthwhile effort to design a controller/observer or a classical controller.

For comparison with the Laplace method, the transfer function in zero/pole/gain form was calculated in Matlab® using the commands “zpk” and “ss2tf”. The command ss2tf calculates the transfer function using the equation: $TF = \frac{num}{den} = C(sI - A)^{-1}B + D$. The command “zpk” converts the transfer function into a zero/pole/gain format. This results in the transfer function:

$$\frac{\Theta(s)}{V_{IN}(s)} = \frac{67838427}{s(s + 1.775 \times 10^4)(s + 2.65)}$$

Laplace Method

To represent the model via Laplace methods, take the Laplace Transform of all continuous system equations:

Electrical System Equations:

$$\begin{aligned} V_M(s) &= K_A V_{IN}(s) \\ V_M(s) &= I_A(s)R_A + L_A s I_A(s) + V_B(s) \\ V_B(s) &= K_B \Omega(s) = K_B s \Theta(s) \end{aligned}$$

Mechanical System Equations:

$$T(s) = K_T I_A(s)$$

$$T(s) = J_T \alpha(s) + b\omega(s) = J_T s\omega(s) + bs\theta(s) = J_T s^2\theta(s) + bs\theta(s)$$

Then collect terms & Equate V_M to V_M

$$K_A V_{IN}(s) = I_A(s)R_A + L_A s I_A(s) + V_B(s)$$

Substitute in V_B

$$K_A V_{IN}(s) = I_A(s)R_A + L_A s I_A(s) + K_B s\theta(s)$$

Substitute in Torque for Current

$$K_A V_{IN}(s) = \frac{T(s)}{K_T} R_A + L_A s \frac{T(s)}{K_T} + K_B s\theta(s)$$

Substitute in Inertia & Damping for Torque

$$K_A V_{IN}(s) = \frac{J_T s^2\theta(s) + bs\theta(s)}{K_T} R_A + L_A s \frac{J_T s^2\theta(s) + bs\theta(s)}{K_T} + K_B s\theta(s)$$

Collect Terms

$$K_A V_{IN}(s) = \left(\frac{J_T s^2 + bs}{K_T} R_A + L_A s \frac{J_T s^2 + bs}{K_T} + K_B s \right) \theta(s)$$

$$K_A V_{IN}(s) = \left(\frac{R_A}{K_T} J_T s^2 + \frac{R_A}{K_T} bs + \frac{L_A}{K_T} J_T s^3 + \frac{L_A}{K_T} bs^2 + K_B s \right) \theta(s)$$

$$K_A V_{IN}(s) = \left(\frac{L_A}{K_T} J_T s^3 + \left(\frac{R_A J_T + L_A b}{K_T} \right) s^2 + \left(\frac{R_A}{K_T} b + K_B \right) s \right) \theta(s)$$

Solving for the transfer function (Bishop, Dorf 2001)

$$\frac{\theta(s)}{V_{IN}(s)} = \frac{K_A K_T}{L_A J_T s^3 + (R_A J_T + L_A b) s^2 + (R_A b + K_B K_T) s}$$

Putting it in a factorized form (Zero, Pole, Gain form)

$$\frac{\theta(s)}{V_{IN}(s)} = \frac{K_A K_T}{s \left(s + \frac{(R_A J_T + L_A b) \pm \sqrt{(R_A J_T + L_A b)^2 - 4 L_A J_T (R_A b + K_B K_T)}}{2 L_A J_T} \right)}$$

$$\frac{\theta(s)}{V_{IN}(s)} = \frac{K_A K_T}{s \left(s + \frac{(R_A J_T + L_A b) \pm \sqrt{R_A^2 J_T^2 + 2 R_A J_T L_A b + L_A^2 b^2 - 4 L_A J_T R_A b - 4 L_A J_T K_B K_T}}{2 L_A J_T} \right)}$$

$$\frac{\theta(s)}{V_{IN}(s)} = \frac{K_A K_T}{s \left(s + \frac{(R_A J_T + L_A b) \pm \sqrt{R_A^2 J_T^2 - 2R_A J_T L_A b + L_A^2 b^2 - 4L_A J_T K_B K_T}}{2L_A J_T} \right)}$$

Because the physical system has gains associated with the D/A converter & the encoder, the open loop model is easily modified to represent the physical system by multiplying by the appropriate gains yielding:

$$\frac{\theta(s)}{V_{IN}(s)} = \frac{K_{DA} K_A K_T K_{ENC}}{L_A J_T s^3 + (R_A J_T + L_A b) s^2 + (R_A b + K_B K_T) s}$$

Substituting the physical constants in and converting to a zero/pole/gain form:

$$\frac{\theta(s)}{V_{IN}(s)} = \frac{67833423.6879}{s (s + 1.775 \times 10^4) (s + 2.65)}$$

It is important to note that this transfer function is equal to the open loop continuous transfer function generated using the state space techniques.

Model Simulation and Physical System Response

Continuous state space and transfer function models were independently created and simulated using Matlab®. From the following state space form various open loop responses were simulated:

$$\dot{X}(t) = AX(t) + Bu(t), y(t) = CX(t) + Du(t)$$

Where:

$$\dot{X}(t) = \begin{bmatrix} \frac{di_A(t)}{dt} \\ \frac{d\theta(t)}{dt} \\ \frac{d\omega(t)}{dt} \end{bmatrix}, A = \begin{bmatrix} -1.775147 \times 10^4 & 0 & -16.766977 \\ 0 & 0 & 1 \\ 2.304944 \times 10^4 & 0 & 0.472391 \end{bmatrix}, B = \begin{bmatrix} 92.455621 \\ 0 \\ 0 \end{bmatrix}, C = [0 \quad 2000 \quad 0], D = 0$$

In Zero/Pole/Gain form:
$$\frac{\theta(s)}{V_{IN}(s)} = \frac{67838427}{s (s + 1.775 \times 10^4) (s + 2.65)}$$

Where the open loop poles are at $\omega = (0, -1.775 \times 10^4, \& -2.65)$. Note that the pole at $s=0$ is unstable. The motors electrical system pole is $s=-1.775 \times 10^4$ and the mechanical pole is at $s=2.65$. The Laplace method open loop responses were also simulated and are also shown in following sections.

Open Loop Frequency Responses

Figure 3 & Figure 5 show the frequency response of the model using the state space method. Figure 4 & Figure 6 show the frequency response and open loop step response of the model using the Laplace method.

Figure 3 – SS Method: Continuous Position System Bode Plot

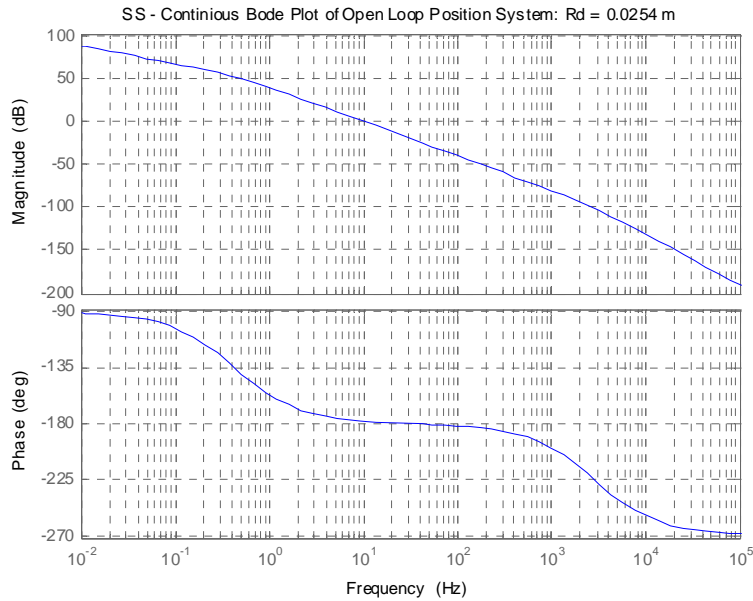


Figure 4 – Laplace Method: Continuous Position System Bode Plot

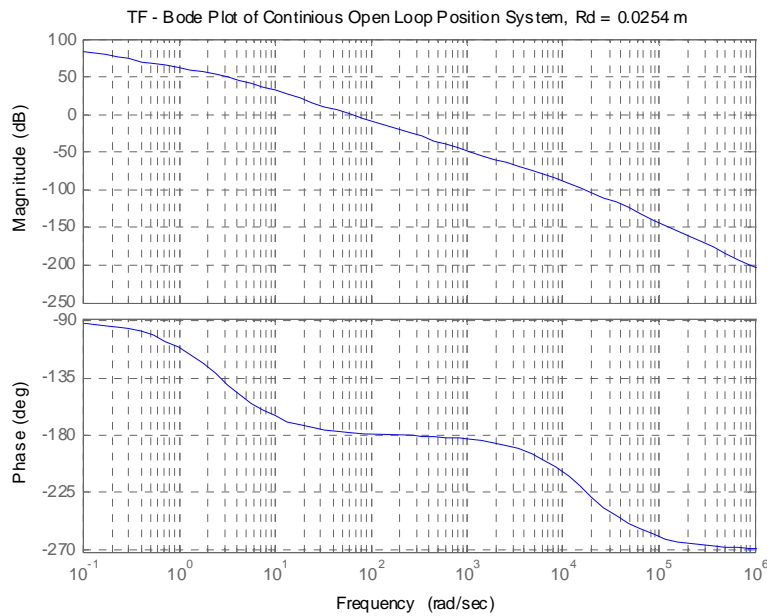


Figure 5 – SS Method: Continuous Velocity System Bode Plot

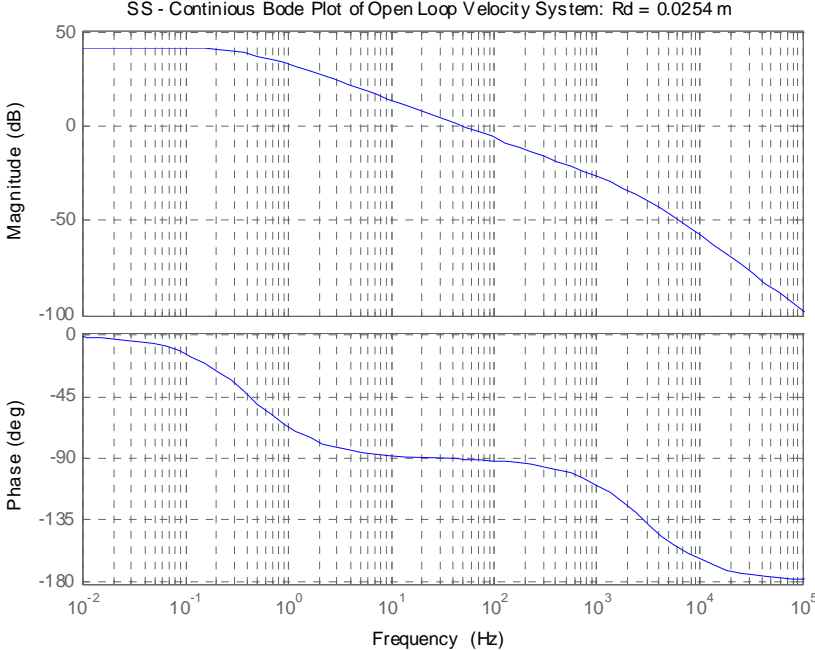
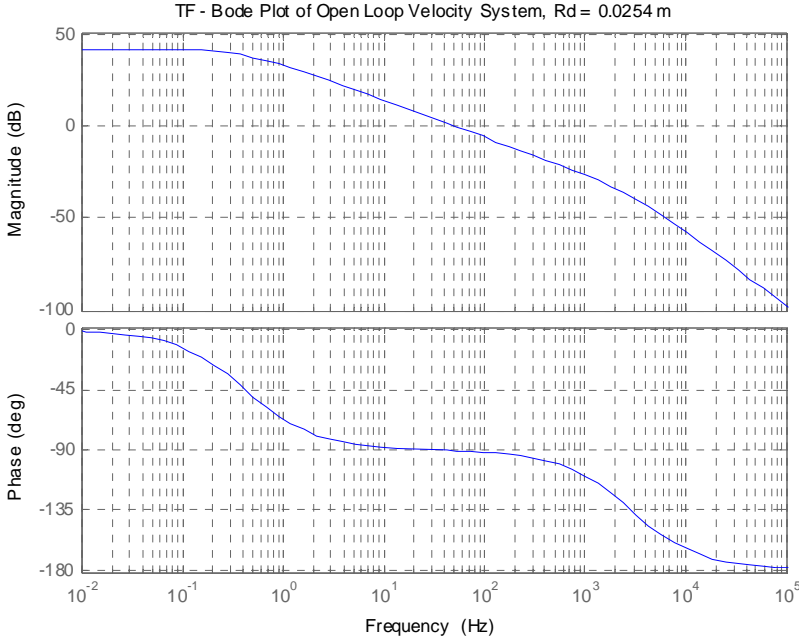


Figure 6 – Laplace Method: Continuous Velocity System Bode Plot



Open Loop Velocity Responses

Figure 7 & Figure 8 show the open loop velocity step responses for both methods. Figure 9 shows the open loop velocity response of the physical system. The measured steady state velocity shown in Figure 9 is close to the steady state open loop velocities predicted by the models in Figure 7 & Figure 8.

Figure 7 – SS Method: Continuous Open Loop Velocity Step Response

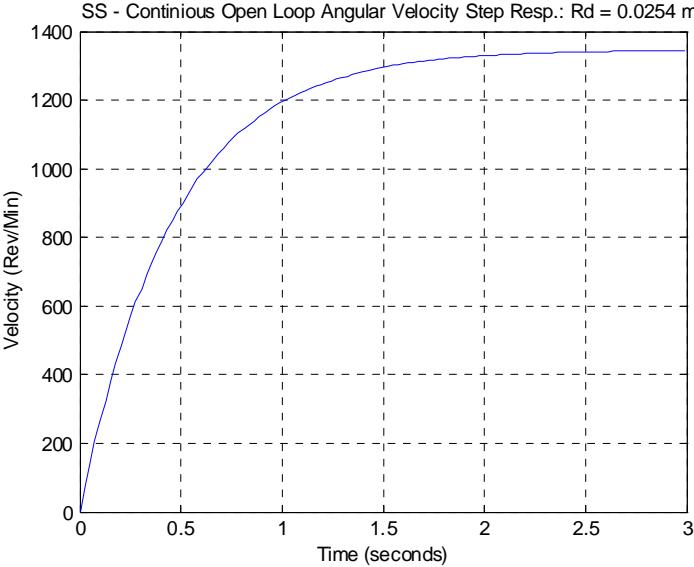


Figure 8 – Laplace Method: Continuous Open Loop Velocity Step Response

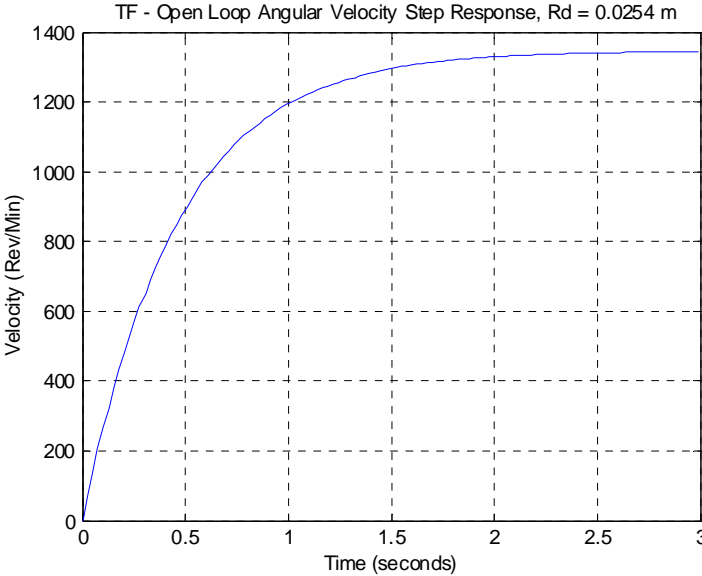
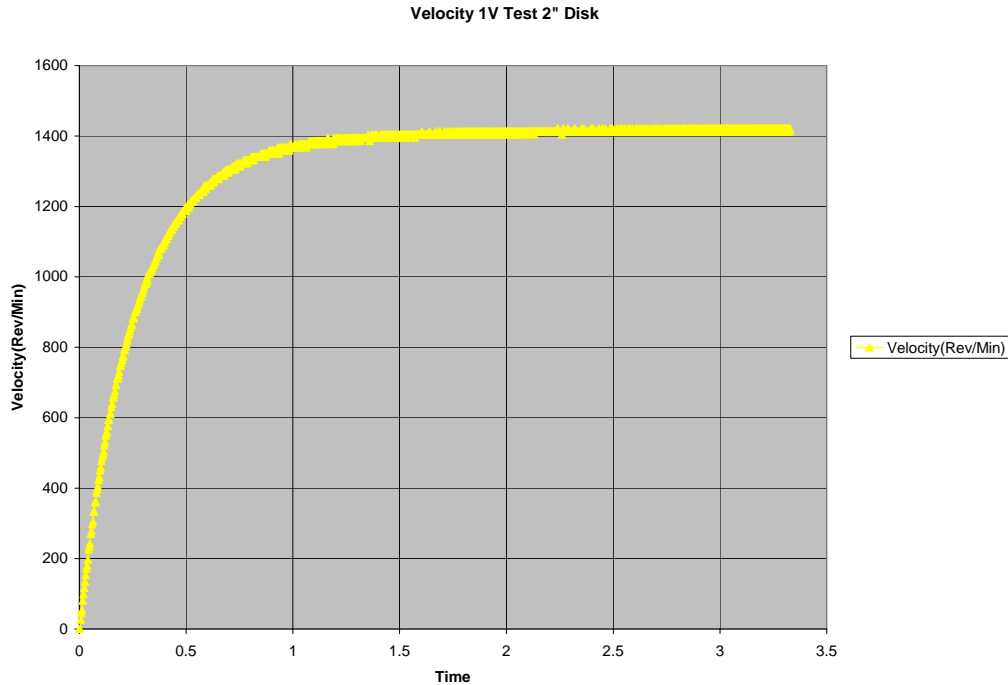


Figure 9 – Physical System: Open Loop Velocity Step Response



Model Parameter Measurement - Locked Rotor Testing

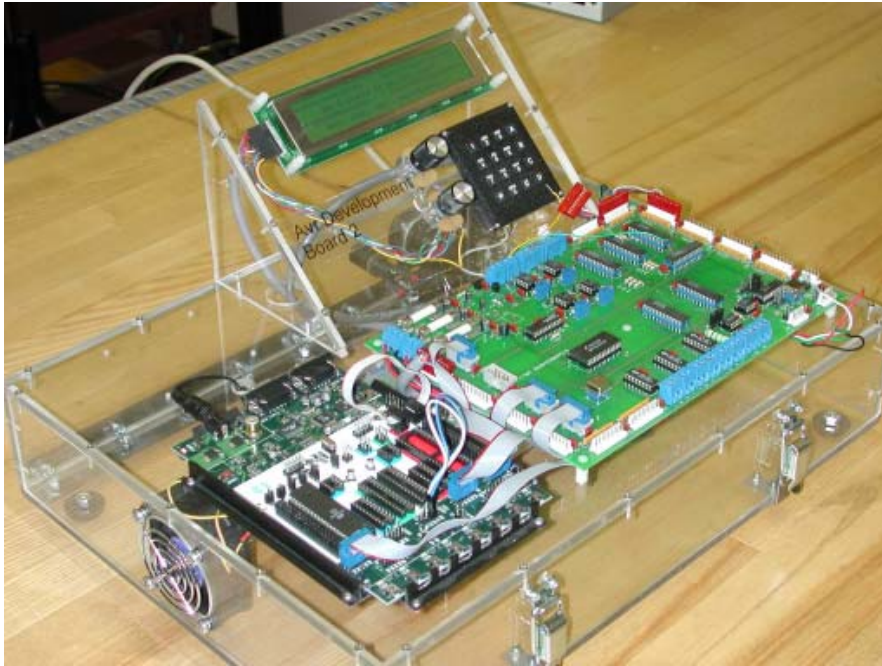
After the initial model was developed simulation predicted results did not agree well with the physical system measurements. At this point further investigation into the manufacturer provided motor parameters. To measure the electrical parameters locked rotor testing was performed. This required measurement of motor voltage and current with the rotor locked (prevented from moving). DC and AC tests were performed to obtain more accurate values for the armature resistance and inductance. This testing yielded $R_A = 30$ Ohms and $L_A=0.0017$ Henrys. The torque constant K_T , Speed Constant K_B , and armature inertia J_A were not measured due to time and equipment constraints.

Controller Design

Proportional Controller

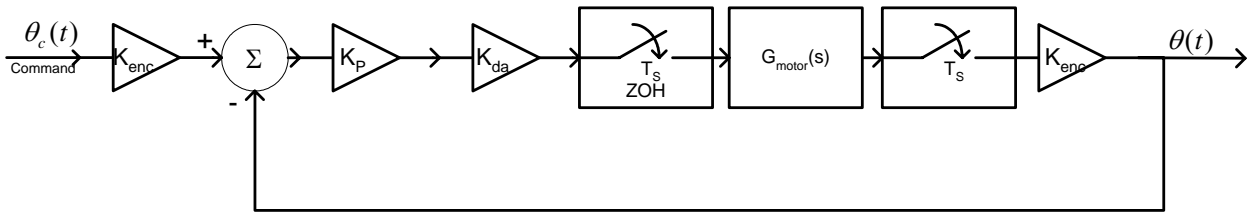
A simple closed loop system implementing a proportional controller was implemented using the U.W. Madison Mechatronics Lab's custom Atmel AVR embedded system with encoder counter and analog output capabilities. This system provides microcontroller for computation and data upload as well as access to three 12-bit encoder counters and a quad 8-bit D/A converter ($\pm 5V$). The Atmel AT90S8535 microcontroller is fast enough to close a control loop implementing several floating-point calculations while uploading position and manipulation values over a serial port at sampling rates up to 1.5 KHz. This system is shown in Figure 10.

Figure 10 – U.W. Madison Mechatronics Lab’s custom Atmel AVR embedded system



A block diagram of the closed loop discrete time proportional controller is shown in Figure 11. It is important to note that the D/A converter and Encoder gains are already included in the motor model. $G_{motor}(s)$ in the block diagram is the pure motor transfer function.

Figure 11 – Closed Loop Discrete Proportional Controller



The original continuous open loop system was discretized using the Matlab® command “c2d”. The discretized model was then integrated into the closed loop system. The poles of the open loop discrete motor system with SR=300Hz (sampling rate), are $z=(1.0000, 0.9912, 0.0000)$. Note that the pole at $z=1$ is not stable. The discrete P control system for a $K_P = 0.01$ with SR = 300 Hz has poles at $z = (-0.0000, 0.9955 + 0.0201j, 0.9955 - 0.0201j)$ (All of which are stable since the magnitude of the roots are less than 1). The discrete closed loop proportional controller Bode plot is shown in Figure 12. The discrete closed loop proportional controller step response is shown in Figure 13. This step response shows a rise time of ~0.2 seconds using a 10%-90% criteria, a maximum overshoot of 50%, and as settling time of ~3 seconds (Kuo 1995).

Figure 12 – Discrete Closed Loop P Control Bode Plot

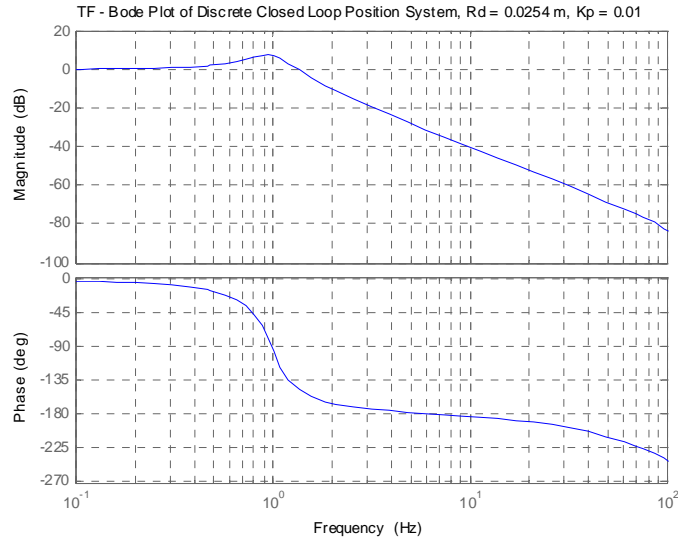
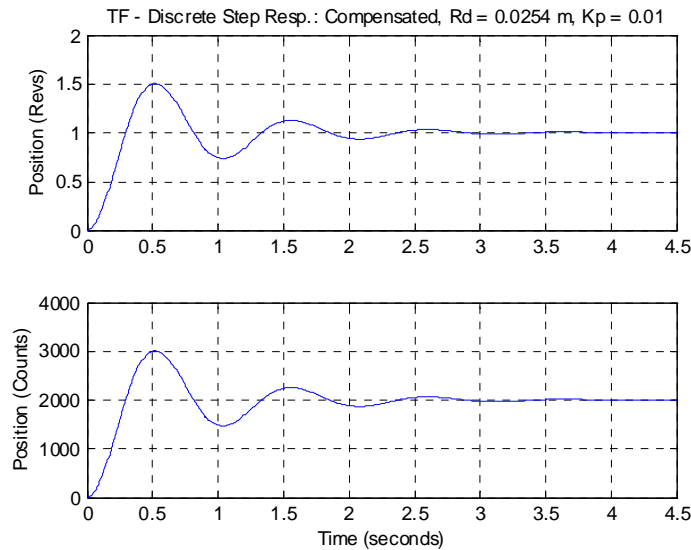
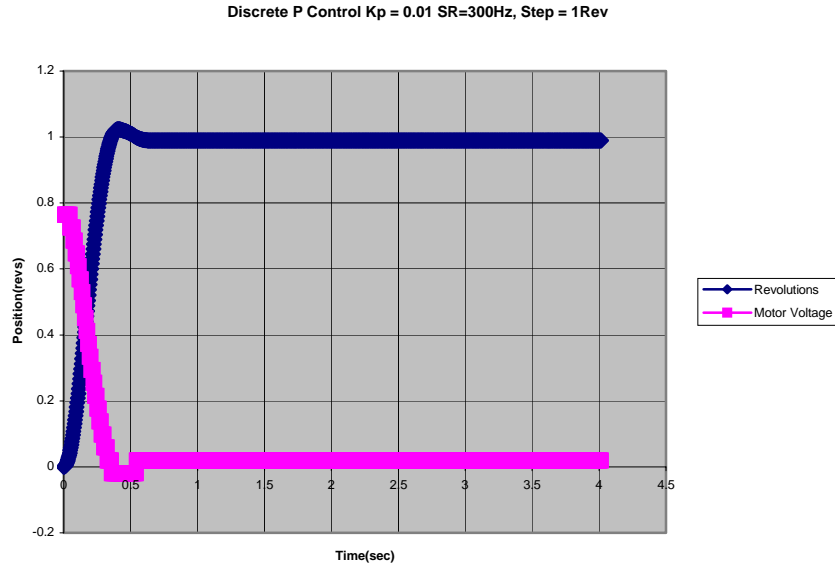


Figure 13 – Discrete Closed Loop P Control Step Response



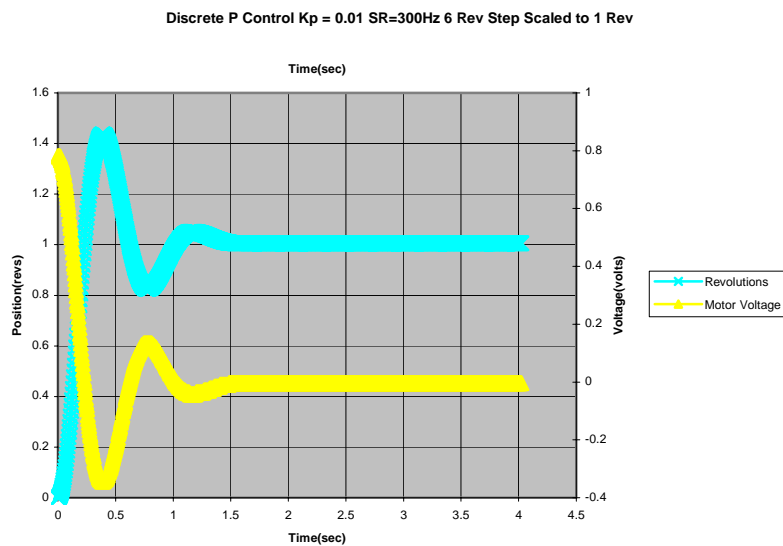
The proportional controller was physically implemented on U.W. Madison Mechatronics Lab's custom Atmel AVR embedded system and tested. The step response from the physical system is shown in Figure 14. This is obviously different than the simulation results. The rise time is close to predicted but maximum overshoot and settling time are significantly different. This is because coulomb friction was neglected in the motors model. For motors in this power range, coulomb friction has a significant impact on the response.

Figure 14 – Discrete P Control Physical System Step Response



Generating a larger step in the physical system and scaling the response back to a unit step input generates a plot closer to the predicted response. Figure 15 shows a step size of six revolutions that was scaled back down to a unit step (by dividing the position and output voltages by the step size - six). In this case its rise time is still close to the predicted. However, the maximum overshoot is significantly closer and the settling time is much closer to predicted values. The coulomb friction has less impact, the larger initial step of the “scaled response” is. For this physical system, a step of larger than six revolutions with a gain of $K_p=0.01$ would lead to the output voltage clipping and the system becoming more nonlinear. Therefore it was not possible to obtain larger step sizes for this system.

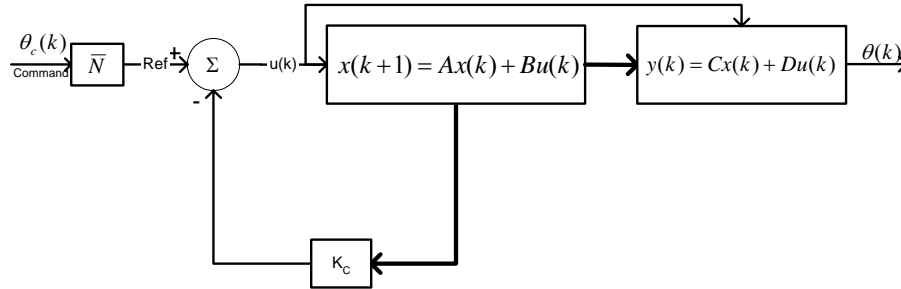
Figure 15 – Discrete P Control Larger Step Response Scaled To A Unit Step Response



State Feedback Design

A discrete state feedback controller was designed to achieve a similar response to the discrete proportional controller. Initially the continuous state space model was converted to a discrete model using the Matlab® command “c2d” again along with the same SR=300Hz. A block diagram showing the structure of a state feedback system is shown in Figure 16.

Figure 16 – Discrete State Feedback Block Diagram with Reference Input

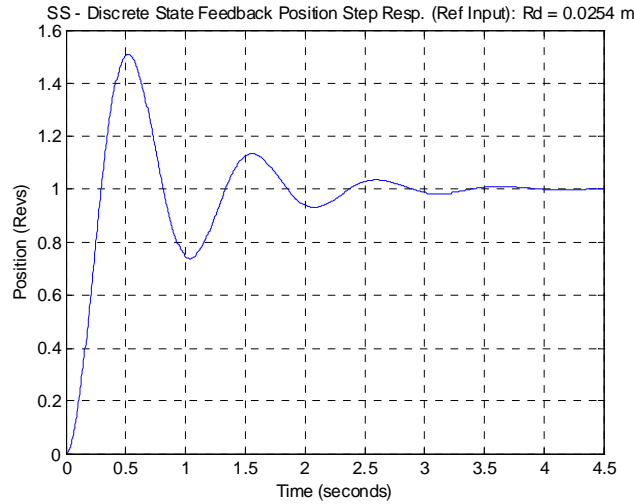


To facilitate comparison, the discrete state feedback controller was assigned the same eigen values as the discrete proportional controller with $K_P = 0.01$ and SR=300Hz. The discrete state feedback controller has its eigen values assigned by solving $\det(sI - A + BK)$ for the characteristic polynomial and equating its coefficients in terms of K_i to the coefficients of the desired characteristic polynomial (Chen 1999). This is done in Matlab® by use of the “place” function. For the desired eigen values of $(-0.0000, 0.9955 + 0.0201j, 0.9955 - 0.0201j)$, the feedback gain K_C was calculated using “place” to be:

$$K_C = [-0.000000000000045 \quad 3.18309886184065 \quad -0.000000000000024]$$

The correct eigen value assignment was verified by calculating $eig(A - BK_C)$. For the eigen value assignment it is assumed full state feedback is available. The reference input step response for the state feedback controller shown in Figure 17 is nearly identical to the step response for the proportional controller. This is the expected result since the poles of the state space system were assigned to the poles of the discrete proportional controller. It is necessary to use the steady state states as “reference” step input since actual output is not fed back and compared to a reference. Only the state multiplied by the feedback gain K_C is fed back. Therefore the appropriate input for the step is the steady state states.

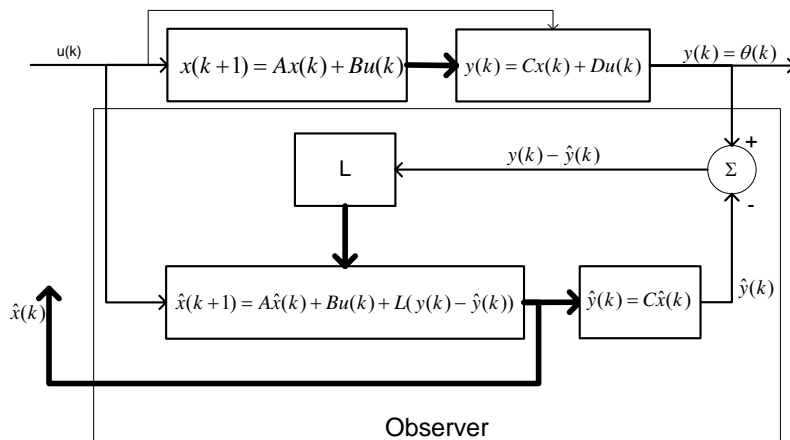
Figure 17 – Discrete State Feedback Controller Step Response (Assumes Full State Feedback)



Observer Design (State Estimator)

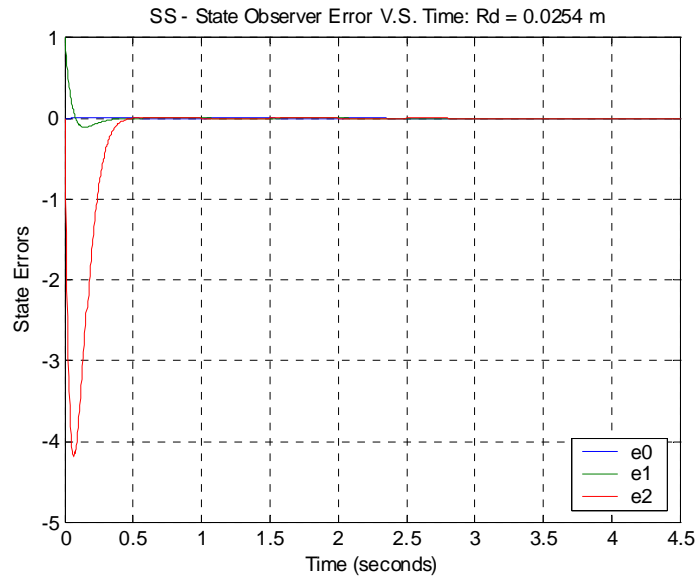
Since the C matrix (output) of the state space model is not all ones and hence does not allow direct monitoring of the internal states of the system, an Observer (State Estimator) is needed to provide estimates of the actual state for use in the state feedback eigen value assignment. An observer (state estimator) is merely another copy of the system that computes an estimate of the states and output compares the estimated output to the real output. The since the observer is only generating estimates of the state ($\hat{x}(k)$), it is important that the error in the state estimate go to zero faster than the actual systems dynamics need to converge to their steady state values. A block diagram showing the structure of a state feedback system with the full observer is shown in Figure 18. The dynamics of the observer are governed by the eigen values of $eig(A - LC)$. The observer eigen values can be arbitrarily assigned just as the eigen values of the controller. This is done using the Matlab® command “place” again, but replacing the matrix B by the matrix C and taking the transposes of each matrix. This works because of the duality between controllability and observability (Messner, Tilbury 1999).

Figure 18 – Discrete Observer Block Diagram with Reference Input



The observer must have “faster” poles (i.e. smaller magnitude) than the discrete state feedback controller poles at $z=(-0.0000, 0.9955 + 0.0201j, 0.9955 - 0.0201j)$. The observer poles were chosen to be at $z=(0.9550 + 0.0201j, 0.9550 - 0.0201j, 0.0000)$. The observer feedback gain matrix was calculated in the previously mentioned fashion to be $L = [0.00004641413231 \quad 0.00025506708338 \quad 0.00161522319476]$. The state error for the observer was plotted and is shown in Figure 19. From the plot the state estimate error converges to zero within ~ 0.5 seconds for the assigned observer eigen values.

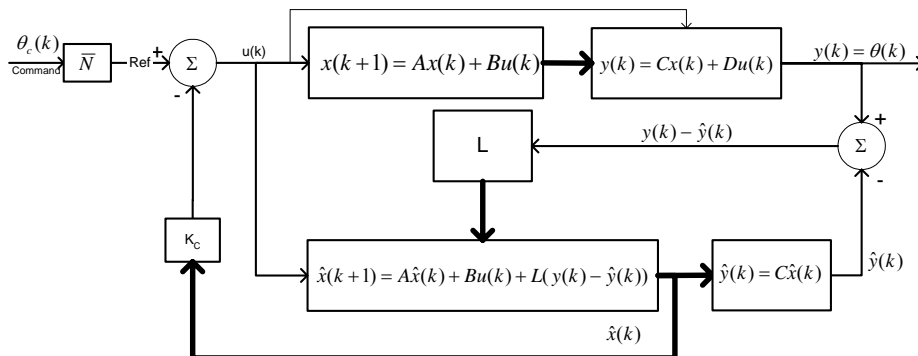
Figure 19 – Observer State Error V.S. Time



State Feedback with Estimated States

After independently designing the state feedback controller and the observer independently the system was lumped together using the estimated states from the observer for the state feedback controller by replacing $u(k) = r(k) - K_c x(k)$ with $u(k) = r(k) - K_c \hat{x}(k)$ (where $r(k)$ is the reference input after scaling by \bar{N}). A block diagram of the controller-estimator configuration is shown in Figure 20 (Chen 1999).

Figure 20 – Discrete State Feedback Controller with Full Observer (Controller-Estimator)



The controller-estimator system can be represented as (Chen 1999):

$$\begin{bmatrix} \dot{x}(k) \\ \dot{e}(k) \end{bmatrix} = \begin{bmatrix} A - BKc & BKc \\ 0 & A - LC \end{bmatrix} \begin{bmatrix} x(k) \\ e(k) \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} r(k)$$

This lumped system was simulated in Matlab®. A plot showing the estimated states, the actual states, and the state error resulting from simulation of the controller-estimator system is shown in Figure 21. At the beginning of the simulation in Figure 21, the state and state estimates are not identical but quickly converge together and then die to zero. Additionally the Bode plot and step response of the discrete the controller-estimator system at SR=300Hz is shown in Figure 22 and Figure 23 respectively. It is worth noting that the controller-estimator Bode and step responses are the same as the state feedback controller and the proportional controller.

Figure 21 – Controller-Estimator States, Estimated States, & State Error

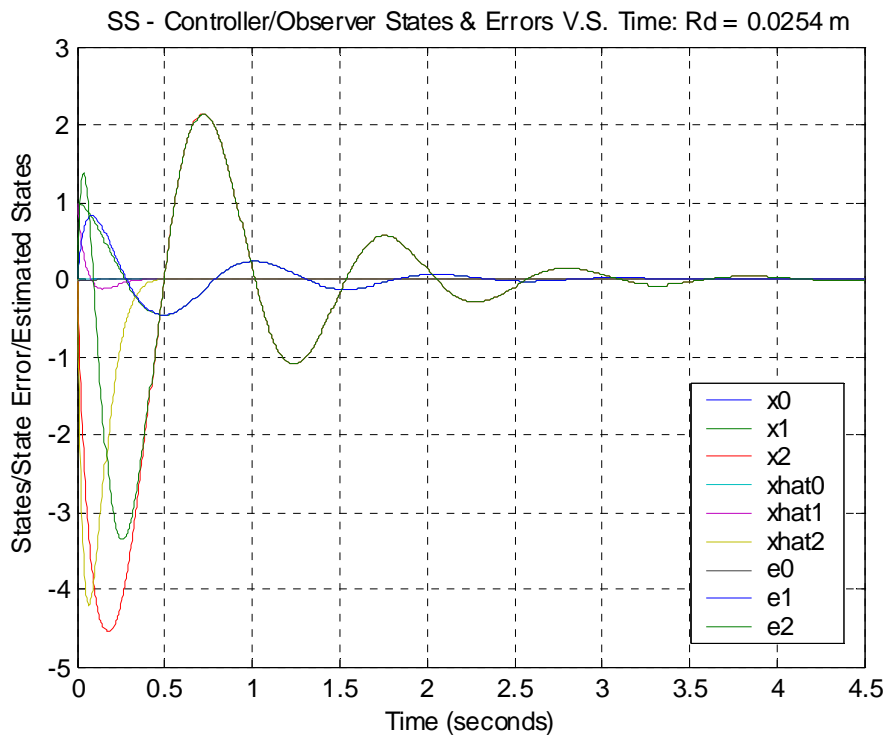


Figure 22 – Discrete Controller-Estimator Bode Plot (SR=300Hz)

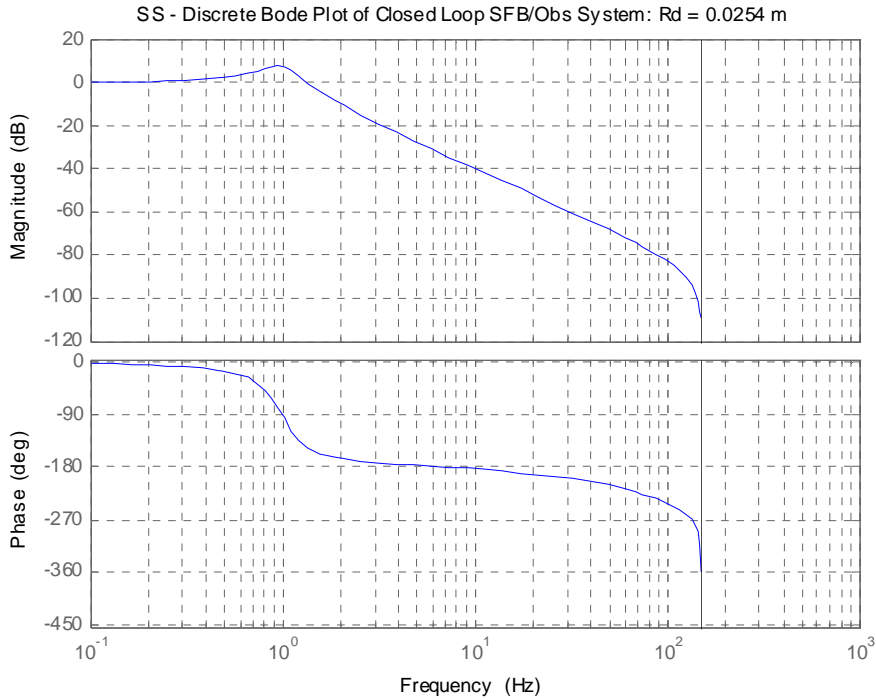
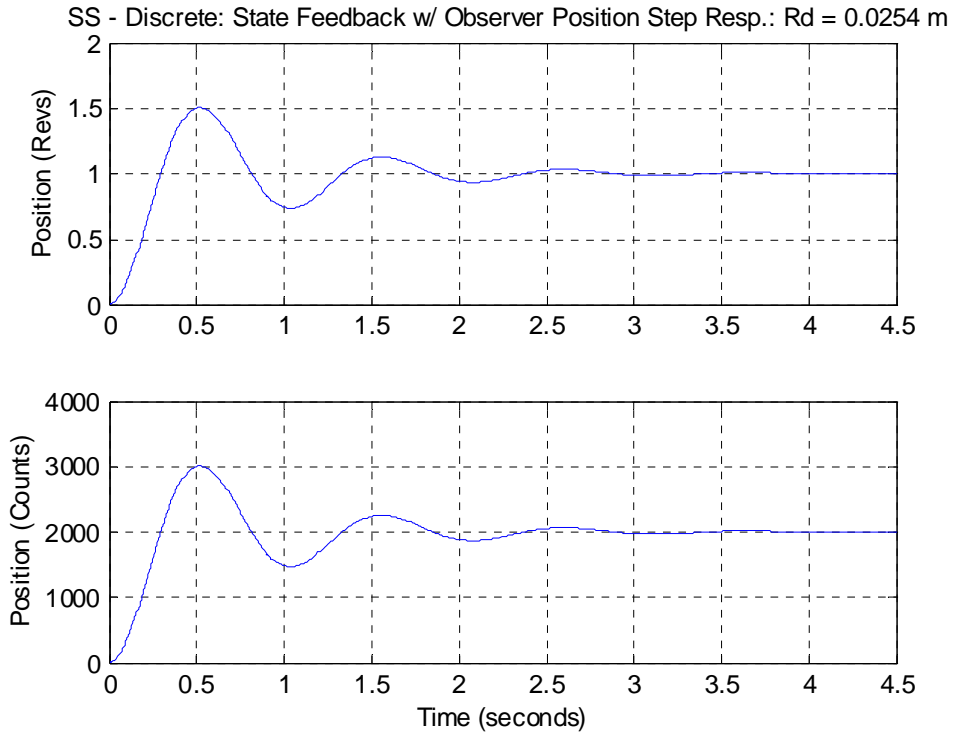


Figure 23 – Discrete Controller-Estimator Step Response (SR=300Hz)



Conclusion

A model was successfully developed for a simple motor – inertia disk system. Discrepancies between the models predicted response and the response of the physical system were observed. These discrepancies can be attributed to several causes.

First is the variance of the physical systems parameters (specifically the motors) from the manufacturers specifications. For example the motor resistance is specified to be 17.9Ω and was measured via locked rotor testing to be $\sim 30\Omega$. Due to equipment and time limitations other motor parameters such as the back-emf constant K_B , the torque constant K_T , and the armature inertia J_A , could not be measured and likely vary from the manufacturers specifications as well.

The second cause of discrepancies resulted from neglecting a significant nonlinearity in the system - coulomb friction. For motors of this power level coulomb friction has a significant contribution in the response when doing position control. Its effect is especially noticeable for small step sizes. If the system is stepped larger distances for a given gain, the predicted linear system dynamics become large enough to see in the measured physical response. The step sizes are limited in the physical system by the output voltage range of the D/A converter. If a step size is too large for a given gain, the voltage output saturates. While the voltage is saturated in this case the response is essentially the velocity system step response. I.e. a constant voltage is applied to the motor and it accelerates to the corresponding velocity.

A proportional controller was successfully implemented on the physical system and simulated in Matlab®. However the effects of coulomb friction dramatically affected the physical systems measured response. This effect was mitigated to some effect by using a larger position step command and scaling the measured response. One lesson readily apparent is that proportional position control; either using a simple proportional controller, or a state space method of motor position for this motor and other small motors may not suffice. This is because the response is non-linear (coulomb friction). To get good performance from the controller and will require a more sophisticated approach such as the addition of active damping or error integration into the controller. Additionally a method for switching between linearized models could be developed.

A state space model, state feedback controller, and state feedback with full observer were designed and simulated in Matlab®. The results from the state space approach agreed well with the Laplace approach simulation results. For this particular system, a simple third order SISO system, the state space approach is computational overkill. Identical results can be obtained with a simple proportional controller with a far less computational load to the processor the system is implemented on (due to the overhead of the matrix operations on the sparse A matrix - $\sim 50\%$ of the entries are zero).

Overall the project was successful in demonstrating some non-linearities in low power DC motors, as well as the practice of model development, and state space controller/observer design.

Bibliography

Kuo, Benjamin C. (1995). Automatic Control Systems Seventh Edition. Englewood Cliffs, New Jersey, Prentice-Hall Inc.

Bollinger, John G., Duffie, Neil A. (1988). Computer Control of Machines and Processes. New York, Addison-Wesley Publishing Company

Messner, William, Tilbury, Dawn, (1999). Control Tutorials for MATLAB® and Simulink: A Web-Based Approach: 1/e. <http://www.engin.umich.edu/group/ctm/>, Regents of the University of Michigan (2002)

Chen, Chi-Tsong (1999). Linear System Theory and Design Third Edition. New York, New York, Oxford University Press

Bishop, Robert H., Dorf, Richard C. (2001). Modern Control Systems Ninth Edition. Upper Saddle River, New Jersey, Prentice-Hall Inc.

Appendix A - Matlab® Source Code

From ssmodel_a.m:

```
clear all; close all;
echo on
% This M-File simulates a State Space Model for a DC Brush motor & Inertia Disk
% Written by Erick L. Oberstar – Oberstar Consulting
% erick@superkits.net
%
% Fall 2002

% Create a state space model for the Maxom motor system with various inertial wheels.

format long

% Physics Constants
rho_Al = 2.702 * 1000 % Density of aluminum g/cm^3 * 1kg/1000g * 1.0e6cm^3*1m^3 = (kg/m^3)

% Servo Amplifier Parameters
Ka = 4 % Voltage Gain set by potentiometer on servo amplifier brick

% Maxon A-Max 26 Model 110953
% Motor Parameters

Kt = 28.3 / 1000 % Motor Torque Constant mNm/A / 1000 = Nm/A
Kb = 1/(337 * (2*pi/60))% Motor Back EMF (Speed) Constant 1/((rpm/V) * (2*pi/60)) = (volt-sec)/radian
Ja = 10.6 * 1.0e-007 % Motor Armature Intertia (g*cm^2) *(1m^2/10000cm^2)*(1kg/1000g) = kg-m^2
b = 5.8e-006 % Motors Viscous damping Coefficient (kg-m2/sec)
% b = 1.8e-005 % Motors Viscous damping Coefficient (kg-m2/sec)
% b = 0 % Motors Viscous damping Coefficient (kg-m2/sec)
% Ra = 17.9 % Motor Armature Resistance (Terminal Resistance) (Ohms)
% Ra = 18.3 % Actual Motor Armature Resistance (Terminal Resistance) (Ohms) (Measured)
Ra = 30 % Actual Motor Armature Resistance (Terminal Resistance) (Ohms) (Measured)
La = 1.69 / 1000 % Motor Armature Inductance (Terminal Inductance) mH / 1000 = H (Henrys)

% Load Parameters
rho = rho_Al % Density of load disk (Aluminum)(kg/m^3)
h = 0.25 * 2.54/100 % Thickness of Load Disk (in * 2.54cm/in * 1m/100cm)= m
% rd = (3.0/2) * 2.54/100 % 3" Radius of Load Disk (in * 2.54cm/1in * 1m/100cm)= m
% rd = (2.5/2) * 2.54/100 % 2.5" Radius of Load Disk (in * 2.54cm/1in * 1m/100cm)= m
rd = (2.0/2) * 2.54/100 % 2" Radius of Load Disk (in * 2.54cm/1in * 1m/100cm)= m
% rd = 0 % No disk
Jl = (rho*pi*h*rd^4)/2 % Load Inertia (1/2 Mass*radius^2) (kg/m^3*m*m^4 = kg-m^2)

% System Parameters
Jt = Ja + Jl % System Inertia - Assume a simple rigidly coupled motor &
% load (no shaft dynamics)
CPR = 2000 % Counts Per Armature Revolution Quadrature Decoded
K_enc = CPR/(2*pi) % Encoder Gain Counts/Radian
% K_enc = CPR % Encoder Gain Counts/Rev

DA_SPN = 10 % Voltage Span of D/A Converter
DA_RES = 8 % Bits of DAC Resolution
K_da = DA_SPN/(2^DA_RES)% D/A converter gain

%Controller Parameters
SR = 300 % Sampling rate for Discrete system (Hz)
Ts = 1/SR % Sampling period for Discrete system (Sec)
stepsize = 10 % # of Revolutions for Step

% Since D/A converter & encoder are treated as gain factors they
% can just be included in the continious state space model.
A_theta_s = [-Ra/La 0 -Kb/La; 0 0 1; Kt/Jt 0 -b/Jt]
B_theta_s = [K_da*Ka/La; 0; 0]
```

```

C_theta_s = [0 K_enc 0]
D_theta_s = 0

echo off
% Check to see if the system is controllable and observable
% if not - I'd be wasting my time.
Ctrl_theta_s = ctrb(A_theta_s,B_theta_s)
if rank(Ctrl_theta_s) == min(size(Ctrl_theta_s))
    disp('Controllable');
else
    disp('Uncontrollable');
end
Obs_theta_s = obsv(A_theta_s,C_theta_s)
if rank(Obs_theta_s) == min(size(Obs_theta_s))
    disp('Observable');
else
    disp('Unobservable');
end

% Just check to make sure the state space model is the same as the tf model
% generated by the Laplace method
[num_theta_s,den_theta_s] = ss2tf(A_theta_s, B_theta_s, C_theta_s, D_theta_s)
TF_theta_s = tf(num_theta_s,den_theta_s)
[z p k] = ss2zp(A_theta_s, B_theta_s, C_theta_s, D_theta_s)
ZPK_theta_s = zpk(z,p,k)

% Make a continuous SS object
sys_theta_s = ss(A_theta_s, B_theta_s, C_theta_s, D_theta_s)

echo on

% The Velocity system for the motor and disk only (no dac/enc gains)
A_omega_s = [-Ra/La -Kb/La; Kt/Jt -b/Jt]
B_omega_s = [Ka/La; 0]
C_omega_s = [0 1]
D_omega_s = 0

echo off

[num_omega_s,den_omega_s] = ss2tf(A_omega_s, B_omega_s, C_omega_s, D_omega_s)

%sys_omega_s = tf(num_omega_s,den_omega_s)
sys_omega_s = ss(A_omega_s, B_omega_s, C_omega_s, D_omega_s)

% Calculate the systems frequency response
i = 0;
i=i+1;
figure(i)
%freqs(nums,dens);
bode(sys_theta_s);
grid on
titlestring = ['SS - Continious Bode Plot of Open Loop Position System: Rd = ',num2str(rd),' m'];
title(titlestring);

i=i+1;
figure(i)
[theta,t] = step(sys_theta_s/(K_da*(2*pi))); % multiply by 1/(K_da*2*pi) make step 1V Step with units Counts
% subplot(2,1,1),
plot(t,theta) % Theta is in units of counts
grid on
xlabel('Time (seconds)')
ylabel('Position (Counts)')
titlestring = ['SS - Continious Open Loop Position System Step Resp.: Rd = ',num2str(rd),' m'];
title(titlestring);

i=i+1;
figure(i)
rlocus(sys_theta_s);
titlestring = ['SS - Continious Root Locus of Open Loop Position System: Rd = ',num2str(rd),' m'];

```

```

title(titlestring);
sgrid(.5,0)
sigrid(10)

% Velocity form of Transfer function
zeros_omega_s = roots(num_omega_s)
poles_omega_s = roots(den_omega_s)
%[Av,Bv,Cv,Dv] = tf2ss(numvs,denvs)

% Calculate the systems frequency response
i=i+1;
figure(i)
bode(sys_omega_s)
grid on
titlestring = ['SS - Continious Bode Plot of Open Loop Velocity System: Rd = ',num2str(rd),' m'];
title(titlestring);

i=i+1;
figure(i)
%zgrid('new')
rlocus(sys_omega_s);
titlestring = ['SS - Continious Root Locus of Open Loop Velocity System: Rd = ',num2str(rd),' m'];
title(titlestring);

i=i+1;
figure(i)
[w,t] = step(sys_omega_s);
w = w*60/(2*pi);
plot(t,w)
xlabel('Time (seconds)')
ylabel('Velocity (Rev/Min)')
titlestring = ['SS - Continious Open Loop Angular Velocity Step Resp.: Rd = ',num2str(rd),' m'];
title(titlestring)
grid on

% Covert the Continious time model to the discrete time model
[A_theta_z,B_theta_z,C_theta_z,D_theta_z] = c2dm(A_theta_s,B_theta_s,C_theta_s,D_theta_s,Ts,'zoh') % Position Model
% Fix B, D & C matrices to include D/A Gain and Encoder Gain
% if below comment out these terms were added into the continious model
% B_theta_z = K_da *2000/2/pi* B_theta_z
% D_theta_z = K_da * D_theta_z
% C_theta_z = K_enc * C_theta_z

%[A_omega_d, B_omega_d, C_omega_d, D_omega_d] = c2dm(A_omega_s, B_omega_s, C_omega_s, D_omega_s, Ts, 'zoh') % Velocity Model

if rank(ctrb(A_theta_z,B_theta_z))==size(ctrb(A_theta_z,B_theta_z))
    disp('SS - The Open Loop Discrete Position State Space form IS Controllable')
else
    disp('SS - The Open Loop Discrete Position State Space form is NOT Controllable')
end
if rank(obsv(A_theta_z,C_theta_z))==size(obsv(A_theta_z,C_theta_z))
    disp('SS - The Open Loop Discrete Position State Space form IS Observable')
else
    disp('SS - The Open Loop Discrete Position State Space form is NOT Observable')
end

NumPts = 3000;
% [x1]=dstep(A_theta_z,2*pi*B_theta_z,C_theta_z,2*pi*D_theta_z,1,NumPts); % multiply B&D by 2pi rad to make step 1 full rev & not just 1 rad
[x1]=dstep(A_theta_z,B_theta_z,C_theta_z,D_theta_z,1,NumPts); % multiply B&D by 2pi rad to make step 1 full rev & not just 1 rad

x2 = x1./(2*pi); % Convert Radians to Revs

t=0:Ts:(NumPts-1)*Ts;
i=i+1;
figure(i)
subplot(2,1,1),

```

```

stairs(t,x1)
grid on
xlabel('Time (seconds)')
ylabel('Position (Radians)')
titlestring = ['SS - Discrete Open Loop Position Stairstep Resp.: Rd = ',num2str(rd),' m'];
title(titlestring)

%i=i+1;
%figure(i)
subplot(2,1,2),
stairs(t,x2)
grid on
xlabel('Time (seconds)')
ylabel('Position (Revs)')
titlestring = ['SS - Discrete Open Loop Position Stairstep Resp.: Rd = ',num2str(rd),' m'];
title(titlestring)

% Perform pole assignment by negative state feedback
% Poles from simplemodel.m with 2"disk & Kp = 0.01
p1 = 0.99550079422763 + 0.02006305837086i
p2 = 0.99550079422763 - 0.02006305837086i
p3 = -0.00000012130233

% Do pole assignment using state feedback to achieve same respons as with
% the P controller
Kc=place(A_theta_z,B_theta_z,[p1,p2,p3]);
A_theta_z_bar = A_theta_z - B_theta_z*Kc;

i=i+1;
figure(i)
% Introduce the Reference Input
% Recall that we don't compare the output to the reference; instead we measure all the states,
% multiply by the gain vector Kc, and then subtract this result from the reference. There is no
% reason to expect that K*x will be equal to the desired output. To eliminate this problem, we
% can scale the reference input to make it equal to K*x_steadystate. This scale factor is often
% called Nbar - Nbar is calculated by using the function rscale(A,B,C,D,K)
% from the Controls Tutorials for Matlab® web page at
% http://www.engin.umich.edu/group/ctm/extras/rscale.html

Nbar=rscale(A_theta_z,B_theta_z,C_theta_z,D_theta_z,Kc)

% Calculate the step response of the state feedback system
[y,t]=step(ss(A_theta_z - B_theta_z*Kc, Nbar*B_theta_z, C_theta_z, D_theta_z,Ts));
% plot(t,y/2/pi)
plot(t,y)
grid on
xlabel('Time (seconds)')
% ylabel('Position (Radians)')
ylabel('Position (Revs)')
titlestring = ['SS - Discrete State Feedback Position Step Resp. (Ref Input): Rd = ',num2str(rd),' m'];
title(titlestring);

CLsyszSS = ss(A_theta_z_bar, B_theta_z, C_theta_z, D_theta_z,Ts)
eig(A_theta_z - B_theta_z*Kc) %verify eigs were assigned as desired

% Observer Design

% Perform pole assignment for observer by negative state feedback methods
% Very Fast Observer Poles (relative to State Feedback pole assignments)
% These cause the state estimation error to go to zero rapidly
op1 = 0.5 + 0.0201i;
op2 = 0.5 - 0.0201i;
op3 = 0;

% Slower Observer Poles - only a little faster than the State Feedback pole assignments
% These cause the state estimation error to go to zero slower but before
% the state goes to zero
op1 = 0.9550079422763 + 0.02006305837086i
op2 = 0.9550079422763 - 0.02006305837086i

```

```

op3 = -0.00000012130233

% Place the observer poles in a similar manner to state feedback assignment
L = place(A_theta_z,C_theta_z,[op1 op2 op3]);
eig(A_theta_z-L*C_theta_z) % check to see if they were assigned properly

% The full system with state feedback using the is:
% This form is from the Controls Tutorials for Matlab® website:
% http://www.engin.umich.edu/group/ctm/state/state.html
% Atz = [A_theta_z - B_theta_z*Kc   zeros(size(A_theta_z))
%       L*C_theta_z - B_theta_z*Kc   A_theta_z - L*C_theta_z];
% Btz = [B_theta_z*Nbar
%       B_theta_z*Nbar];
% Ctz = [ C_theta_z   zeros(size(C_theta_z))];
% Dtz = D_theta_z

% This representation from Linear System Theory and Design
% by CHI-TSONG CHEN, State University of New York, Stony Brook P.254
dimA = length(A_theta_z); % A_theta_z is square so this is ok
Atz = [A_theta_z - B_theta_z*Kc   -B_theta_z*Kc
       zeros(dimA,dimA)         A_theta_z - L*C_theta_z];
Btz = [B_theta_z*Nbar
       zeros(size(B_theta_z))];
Ctz = [C_theta_z zeros(size(C_theta_z))];
Dtz = D_theta_z

FullSys = ss(Atz,Btz,Ctz,Dtz,Ts)

% Compute Full State Feedback / Observer system Step response and compare to transfer function method
i=i+1;
figure(i)
[y,t] = step(FullSys); % Units are Revolutions
y_counts = y*CPR;
subplot(2,1,1), plot(t,y)
grid on
% xlabel('Time (seconds)')
ylabel('Position (Revs)')
titlestring = ['SS - Discrete State Feedback w/ Observer Position Step Resp.: Rd = ',num2str(rd),' m'];
title(titlestring);

subplot(2,1,2), plot(t,y_counts)
grid on
xlabel('Time (seconds)')
ylabel('Position (Counts)')
% titlestring = ['SS - State Feedback w/ Observer Position Step Resp.: Rd = ',num2str(rd),' m'];
% title(titlestring);

% Simulate State, State Estimation & State Estimation Error Convergence
i=i+1;
figure(i)
x0 = [0 1 0];

legend_pos = 0;
% 0 = Automatic "best" placement (least conflict with data)
% 1 = Upper right-hand corner (default)
% 2 = Upper left-hand corner
% 3 = Lower left-hand corner
% 4 = Lower right-hand corner
% -1 = To the right of the plot

% Simulate State Convergence
FullSys2 = ss(Atz,Btz/Nbar,Ctz,Dtz,Ts)
[Y,T,X] = lsim(FullSys2,zeros(size(t)),t,[x0 x0]);
plot(t,X(:,1:3))
grid on
xlabel('Time (seconds)')
ylabel('States')
legend('x0','x1','x2',legend_pos)
titlestring = ['SS - State Convergence V.S. Time: Rd = ',num2str(rd),' m'];
title(titlestring);

```

```

% Simulate State Error Convergence
i=i+1;
figure(i)
plot(t,X(:,(4:6)))
grid on
xlabel('Time (seconds)')
ylabel('State Errors')
legend('e0','e1','e2',legend_pos)
titlestring = ['SS - State Observer Error V.S. Time: Rd = ',num2str(rd),' m'];
title(titlestring);

% Simulate State Estimation Convergence
i=i+1;
figure(i)
plot(t,X(:,(1:3))-X(:,(4:6)))
xlabel('Time (seconds)')
ylabel('Estimated States')
legend('xhat0','xhat1','xhat2',legend_pos)
grid on
titlestring = ['SS - Observer Estimated States V.S. Time: Rd = ',num2str(rd),' m'];
title(titlestring);

% All State, State Estimation & State Estimation Error Convergences on same
% plot
i=i+1;
figure(i)
plot(t,[X(X(:,(1:3))-X(:,(4:6))))])
grid on
xlabel('Time (seconds)')
ylabel('States/State Error/Estimated States')
legend('x0','x1','x2','xhat0','xhat1','xhat2','e0','e1','e2',legend_pos)
titlestring = ['SS - Controller/Observer States & Errors V.S. Time: Rd = ',num2str(rd),' m'];
title(titlestring);

% Calculate the systems frequency response
i = 0;
i=i+1;
figure(i)
%freqs(nums,dens);
bode(FullSys);
grid on
titlestring = ['SS - Discrete Bode Plot of Closed Loop SFB/Obs System: Rd = ',num2str(rd),' m'];
title(titlestring);

```

From simplemodel_b.m:

```

clear all; close all
echo on
% This M-File simulates a Transfer Function Model for a DC Brush motor & Inertia Disk
% Written by Erick L. Oberstar – Oberstar Consulting
% erick@superkits.net
%
% Fall 2002

% Create a simple transfer function model for the Maxom motor system with various inertial wheels.

format long

% Physics Constants
rho_Al = 2.702 * 1000 % Density of aluminum g/cm^3 * 1kg/1000g * 1.0e6cm^3*1m^3 = (kg/m^3)

% Servo Amplifier Parameters
Ka = 4 % Voltage Gain set by potentiometer on servo amplifier brick

% Maxon A-Max 26 Model 110953
% Motor Parameters

```

```

Kt = 28.3 / 1000    % Motor Torque Constant mNm/A / 1000 = Nm/A
Kb = 1/(337 * (2*pi/60))% Motor Back EMF (Speed) Constant 1/((rpm/V) * (2*pi/60)) = (volt-sec)/radian
Ja = 10.6 * 1.0e-007 % Motor Armature Intertia (g*cm^2) *(1m^2/10000cm^2)*(1kg/1000g) = kg-m^2
b = 5.8e-006      % Motors Viscous damping Coefficient (kg-m2/sec)
% b = 1.8e-005    % Motors Viscous damping Coefficient (kg-m2/sec)
% b = 0           % Motors Viscous damping Coefficient (kg-m2/sec)
% Ra = 17.9      % Motor Armature Resistance (Terminal Resistance) (Ohms)
% Ra = 18.3      % Actual Motor Armature Resistance (Terminal Resistance) (Ohms) (Measured)
Ra = 30          % Actual Motor Armature Resistance (Terminal Resistance) (Ohms) (Measured)
La = 1.69 / 1000 % Motor Armature Inductance (Terminal Inductance) mH / 1000 = H (Henrys)

```

```
% Load Parameters
```

```

rho = rho_Al      % Density of load disk (Aluminum)(kg/m^3)
h = 0.25 * 2.54/100 % Thickness of Load Disk (in * 2.54cm/in * 1m/100cm) = m
% rd = (3.0/2) * 2.54/100 % 3" Radius of Load Disk (in * 2.54cm/in * 1m/100cm) = m
% rd = (2.5/2) * 2.54/100 % 2.5" Radius of Load Disk (in * 2.54cm/in * 1m/100cm) = m
rd = (2.0/2) * 2.54/100 % 2" Radius of Load Disk (in * 2.54cm/in * 1m/100cm) = m
% rd = 0         % No disk
Jl = (rho*pi*h*rd^4)/2 % Load Inertia (1/2 Mass*radius^2) (kg/m^3*m*m^4 = kg-m^2)

```

```
% System Parameters
```

```

Jt = Ja + Jl      % System Inertia - Assume a simple rigidly coupled motor &
                  % load (no shaft dynamics)
CPR = 2000        % Counts Per Armature Revolution Quadrature Decoded
K_enc = CPR/(2*pi) % Encoder Gain Counts/Radian
DA_SPN = 10       % Voltage Span of D/A Converter
DA_RES = 8        % Bits of DAC Resolution
K_da = DA_SPN/(2^DA_RES)% D/A converter gain

```

```
%Controller Parameters
```

```

Kp = .01         % PID continuous time Proportional Gain
Ki = 0           % PID continuous time Integral Gain
Kd = 0           % PID continuous time Differential Gain
stepsize = 1     % # of Revolutions for Step
SR = 300         % Sampling rate for Discrete system (Hz)
Ts = 1/SR        % Sampling period for Discrete system (Sec)

```

```
% Other Parameters
```

```
i = 0;          % Figure counter initialization
```

```

% The Transfer function (output position) / (input voltage)
% Since D/A converter & encoder are treated as gain factors they
% can just be included in the continuous tf model.
nums = [0 0 0 K_da*Ka*Kt*K_enc]
dens = [(La*Jt) (Ra*Jt + La*b) (Ra*b+Kb*Kt) 0]
% Units on transfer function are Radians/Volt

```

```

syss = tf(nums,dens)
Poles=pole(syss)
[z p k] = tf2zp(nums,dens)
syss_zpk = zpk(z,p,k)

```

```
% Calculate the systems frequency response
```

```

i=i+1;
figure(i)
bode(syss);
grid on
titlestring = ['TF - Bode Plot of Continious Open Loop Position System, Rd = ',num2str(rd),' m'];
title(titlestring);

```

```
% Calculate the systems root locus
```

```

i=i+1;
figure(i)
rlocus(syss);
titlestring = ['TF - Root Locus, Continious Open Loop Position System, Rd = ',num2str(rd),' m'];
title(titlestring);
sgrid(.5,0)
sigrid(10)

```

```

% Calculate open loop step response
% without D/A Gain as it was programmed for a constant output voltage
[theta,t] = step(stepsize*sys/K_da); % stepsize here is volts Position is in counts since K_enc left in
% theta_counts = CPR/(2*pi)*theta;
i=i+1;
figure(i)
plot(t,theta);
% grid on
% plot(t,theta_counts)
grid on
xlabel('Time (seconds)')
% ylabel('Position (Radians)')
ylabel('Position (Counts)')
titlestring = ['TF - Open Loop Position Step Response, Rd = ',num2str(rd),' m, Kp = ',num2str(Kp)];
title(titlestring);

```

```

% Velocity form of Transfer function
% Just simulate the Motor itself - so don't include D/A Gain & Encoder Gain
numvs = [0 0 Kt*Ka]
denvs = [(La*Jt) (Ra*Jt + La*b) (Ra*b+Kb*Kt)]
sysv = tf(numvs,denvs)

```

```

Polesv = pole(sysv)/(2*pi) % to turn from radian roots frequency roots

```

```

% Calculate the systems frequency response
i=i+1;
figure(i)
bode(sysv)
grid on
titlestring = ['TF - Bode Plot of Open Loop Velocity System, Rd = ',num2str(rd),' m'];
title(titlestring);

```

```

% Calculate Root Locus of the open loop velocity system
i=i+1;
figure(i)
sgrid
% subplot(2,1,1),
rlocus(sysv);
titlestring = ['TF - Root Locus of Open Loop Velocity System, Rd = ',num2str(rd),' m'];
title(titlestring);
sgrid(.5,0)
sigrid(10)

```

```

i=i+1;
figure(i)
[w,t] = step(stepsize*sysv);
w = w*60/(2*pi); % Revs /min
% w = w/(2*pi); % revs /sec
% subplot(2,1,2),
plot(t,w)
xlabel('Time (seconds)')
ylabel('Velocity (Rev/Min)')
titlestring = ['TF - Open Loop Angular Velocity Step Response, Rd = ',num2str(rd),' m'];
title(titlestring)
grid on

```

```

% Covert the Continious time model to the discrete time model

```

```

sysz = c2d(sys,Ts,'zoh');
Polez = pole(sysz)

```

```

% Connect the Porportional Controller, D/A, Encoder & Motor models in series
OLsysz = Kp * sysz

```

```

% Calculate the Closed Loop Transfer function

```

```

CLsysz = feedback(OLsysz,1)

% Calculate the Step Response
i=i+1;
figure(i)
[y1,t] = step(stepsize*CLsysz);

%[y1,x] = dstep(CLsysz);
subplot(2,1,1),plot(t,y1)
grid on
% xlabel('Time (seconds)')
ylabel('Position (Revs)')
titlestring = ['TF - Discrete Step Resp.: Compensated, Rd = ',num2str(rd),' m, Kp = ',num2str(Kp)];
title(titlestring);

%y2 = y1*CPR./(2*pi); % Convert to Revs
y2 = y1*CPR;

subplot(2,1,2),plot(t,y2)
grid on
xlabel('Time (seconds)')
ylabel('Position (Counts)')
% titlestring = ['TF - Discrete Step Resp.: Compensated, Rd = ',num2str(rd),' m, Kp = ',num2str(Kp)];
% title(titlestring);

format

figure(7) % Select step response graphs

% Calculate the Discrete Closed Loop Systems frequency response
i=i+1;
figure(i)
bode(CLsysz)
grid on
titlestring = ['TF - Bode Plot of Discrete Closed Loop Position System, Rd = ',num2str(rd),' m, Kp = ',num2str(Kp)];
title(titlestring);

```

Appendix B - Proportional Controller C Source Code

From ctrltest.c (Contains Proportional Controller Code

/******

This program was produced by the
CodeWizardAVR V1.0.2.0 Standard
Automatic Program Generator
© Copyright 1998-2001
Pavel Haiduc, HP InfoTech S.R.L.
http://infotech.ir.ro
e-mail:dhptech@ir.ro , hpinfotech@xnet.ro

Project : Linear Motor Model and Controller
Version : 1.20
Date : 11/11/2002
Author : Erick L. Oberstar
Company : Oberstar Consulting
Comments:

Chip type : AT90S8535
Clock frequency : 8.000000 MHz
Memory model : Small
Internal SRAM size : 512
External SRAM size : 0

Data Stack size : 128

Hardware Configuration

Motor Feedback - Encoder Channel 0
Manipulated Output - DAC Channel 0

Address Bus - PORTA
Data Bus - PORTB
LCD Port - PORTC

Serial - PORTD to RS232-Spare Header
PD0 - RXD
PD1 - TXD

*****/

```
#include <90s8535.h>
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
```

```
#include "dcmh.h"
#include "encoder.h"
#include "dac.h"
#include "pencilctrl.h"
```

```
#define samplingrate 300 // Sampling rate (Hz) (Integers only)
#define positiongoal 2000 // Position Goal Value (counts) (Integers only)
#define gainval .01 // Proportional Gain value
#define t2countval (7813/(samplingrate))// Have the compiler calculate timer count // to load to achieve sampling rate

#define ENCODER_CPR 500
#define ENCODER_CTR_RANGE 4096 // 12-bit encoder counter range 2^12 = 4096
#define DC_OFFSET 2 // DC offset in Counts to Make Zero Volts be Zero Volts
#define POSITIVE_CLIP 127.0
#define NEGATIVE_CLIP -128.0
// #define POSITIVE_CLIP 25.0
// #define NEGATIVE_CLIP -25.0
```

```
#define TRUE 0xFF
#define FALSE 0x00
```

```
// Alphanumeric LCD Module functions
// .equ __lcd_port=0x12 ; = PORTD
#asm
.equ __lcd_port=0x15 ; 0x15 = PORTC
#endasm
#include <lcd4x40.h>
```

```
#pragma used+
// UART Receiver buffer
char rx_buffer[32];
volatile unsigned char rx_wr_index,rx_rd_index,rx_counter;
// This flag is set on UART Receiver buffer overflow
bit rx_buffer_overflow;
volatile unsigned char datahere;
#pragma used-
```

```
// Declare your global variables here
// Declare a Union for ActualPosition long int with bytes for serial output
union CharLongUnion
{
    unsigned char bytes[4]; //These two bytes overlap the integer variable
    unsigned long int ULinteger;
};
```

```

//long int currentposition;      // Contains current axis position 32-bit
union CharLongUnion currentposition;// Current position accounting for rollover 32 Bit
long int actualposition;        // Actual position accounting for rollover
long int commandposition;      // Commanded Position value
long int positionerror;        // Error between commandposition and actualposition

float manipulation;            // Manipulated output voltage
float gain;                    // Gain for control equation
float positionerror_fp;        // floating point value for position error

char commandvalue;            // Integer format command value
int deltaposition;            // Change in position (used for handling overflow)
int newposition;              // Current position read in from encoder counter
int oldposition;              // Contains past axis position @ last sample

unsigned char i;              // Loop counter for training sequence
volatile char bytein;         // holds serial byte in - holds up serial output stream
char flag;                    // Flag used for allowing control / serial stream
char PencilStatus;           // Holds pencil sharpener status byte
char* asciigain[10];          // ASCII string conversion of floating pt. gain value
char* ascisamplingrate[6];    // ASCII string conversion of sampling rate integer value
char* asciicommandposition[6]; // ASCII string conversion of long integer command position value

//*****
// Timer 2 output compare interrupt service routine
interrupt [TIM2_COMP] void timer2_comp_isr(void)
{
// Place your code here
//PORTD=0XFF; // ACTIVE LOW for external triggering of Labview DAQ

PORTD.6 = ~PORTD.6;          // debugging to figure out sampling rate

newposition = RDEncoder(0);  // Reads encoder channel 0

// Handle Rollover cases
deltaposition = newposition - oldposition; // Calculate delta to check rollover

if (deltaposition > (ENCODER_CTR_RANGE/2))
{
deltaposition = deltaposition - (ENCODER_CTR_RANGE); // Rolled over negative (underflow)
}

if (deltaposition < -(ENCODER_CTR_RANGE/2))
{
deltaposition = deltaposition + (ENCODER_CTR_RANGE); // Rolled over positive (overflow)
}

actualposition = actualposition + deltaposition; // Calculate actual position accounting for rollover
oldposition = newposition; // update old position for next iteration

// currentposition is used for control purposes
//currentposition = actualposition;
currentposition.UInteger = actualposition;

// Implement Proportional Control algorithm
//positionerror = commandposition - currentposition; // Calculate position error (current - command)
positionerror = commandposition - currentposition.UInteger; // Calculate position error (current - command)

positionerror_fp = (float) positionerror; // Convert position error to float type
manipulation = gain*positionerror_fp;

if (manipulation > POSITIVE_CLIP)
{
manipulation = POSITIVE_CLIP;
}

```

```

if (manipulation<NEGATIVE_CLIP)
{
    manipulation = NEGATIVE_CLIP;
}

commandvalue = (unsigned char)(manipulation + 128.0 + DC_OFFSET); // Take care of DC offset & type cast to right size
//commandvalue = 6 + 128.0 + DC_OFFSET; // Put .25 volts out for velocity step test
//commandvalue = 13 + 128.0 + DC_OFFSET; // Put .5 volts out for velocity step test
//commandvalue = 2*25.5 + 128.0 + DC_OFFSET; // Put 2.0 volts out for velocity step test
//commandvalue = 25.5 + 128.0 + DC_OFFSET; // Put 1.0 volts out for velocity step test

PencilStatus = RDPencilStatus(); // Read in limit switch states from pencil sharpener
if (((PencilStatus & Soft_Lim_Left)==0)((PencilStatus & Soft_Lim_Right)==0))
{
    commandvalue = 128; // If either soft limit is hit stop
}

//output = (signed char)(gain*(positiongoal - currposition));
WRSingleDAC(commandvalue,0); // Write encoder count to DAC0

//WRSingleDAC(i,1); // Write encoder count to DAC1
//i+=i;

//commandvalue = 255;
//currentposition.UInteger = 4294967295;

putchar('m'); //Put a "m" out serial to delimit manipulation value (Start of frame)
putchar(commandvalue); // Send DAC value out Serial
putchar('p'); // Put a "p" out serial to delimit Position value
putchar(currentposition.bytes[0]); // Send LSB of current position count value out Serial
//putchar('p'); // Put a "p" out serial to delimit Position value
putchar(currentposition.bytes[1]); // Send 2nd byte of current position count value out Serial
//putchar('p'); // Put a "p" out serial to delimit Position value
putchar(currentposition.bytes[2]); // Send 3rd byte of current position count value out Serial
//putchar('p'); // Put a "p" out serial to delimit Position value
putchar(currentposition.bytes[3]); // Send MSB of current position count value out Serial
// Send end of frame character
//putchar(0x0D); // Send last character for framing (Carriage Return)

//PORTD.6 = 1; // debugging to figure out sampling rate
//PORTD=0X00; // sET LOW AGAIN

}

/*****
/*****
void main(void)
{
// Declare your local variables here

// initialize values
flag = 0; // don't initially do control or send data
commandposition = 0; // set Command position to zero initially
gain = gainval;

// Input/Output Ports initialization
// Port A
PORTA=0x3F;
DDRA=0x3F;

// Port B
PORTB=0xFF;
DDRB=0xFF;

// Port C
PORTC=0xFF;
DDRC=0xFF;

```

```

// Port D
/*
PORTD=0x00;
DDRD=0x00;
*/
PORTD=0xFF;
DDRD=0xFF;

// Reset all 3 encoders
RESEncoders(); // Redudant to get leading zeros

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Output Compare
// OC0 output: Disconnected
//TCCR0=0x00;
//TCNT0=0x00;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 125.000 kHz
TCCR0=0x03;
//TCNT0=0x7D;
TCNT0=0x19;// 5 kHz Rate

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer 1 Stopped
// Mode: Output Compare
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: 7.813 kHz
// Mode: Output Compare
// OC2 output: Disconnected
// Timer/Counter 2 is cleared on compare match
TCCR2=0x0F;
ASSR=0x00;
TCNT2=0x00;
OCR2= (char)(t2countval);

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
GIMSK=0x00;
MCUCR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
// TIMSK=0x80; // enable timer 2 interrupt
TIMSK = 0; // Disable timer 2 interrupt

// UART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// UART Receiver: On

```

```

// UART Transmitter: On
UCR=0xD8;

// UART Baud rate: 9600
//UBRR=0x33; // 8MHz Crystal
//UBRR=0x2F; // 7.3728Mhz Crystal

// UART Baud rate: 19200
//UBRR=0x19; // 8MHz Crystal

// UART Baud rate: 38400
UBRR=0x0C; // 8MHz Crystal

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;

// LCD module initialization
lcd_init();

// Reset all 3 encoders
RESEncoders();

// Zero out all bipolar DAC outputs (0x7F is 1/2 value of -5 to +5V)
WRSingleDAC(0x7F + DC_OFFSET,0); // Zero out output voltage to DAC0
WRSingleDAC(0x7F + DC_OFFSET,1); // Zero out output voltage to DAC0
WRSingleDAC(0x7F + DC_OFFSET,2); // Zero out output voltage to DAC0
WRSingleDAC(0x7F + DC_OFFSET,3); // Zero out output voltage to DAC0

commandposition = positiongoal; // set Command position
gain = gainval;

// Global enable interrupts
#asm("sei")

bytein = 0; //

ftoe(gain, 3, asciigain); //Generate text string of gain
itoa(samplingrate, asciisamplingrate); //Generate text string of samplingrate
ltoa(commandposition, asciicommandposition); //Generate text string of commandposition

WRPencilCommand(0b11111011); // Deactivate all outputs except motor inhibit (logic inverted in hardware)

while (1)
{
// Update LCD inbetween interrupts

lcd_gotoxy(0,0);
lcd_putsf(" ECE717 - Linear Systems ");
lcd_gotoxy(0,1);
lcd_putsf(" Linear Motor Model & Control ");
lcd_gotoxy(0,2);
lcd_putsf("*****");
lcd_gotoxy(0,3);
lcd_putsf("SR= Hz, Gain= , PosGoal= ");

//display sampling rate
lcd_gotoxy(3,3);
lcd_puts(asciisamplingrate);

//display Gain value
lcd_gotoxy(16,3);
lcd_puts(asciigain);
}

```

```

//display position goal
lcd_gotoxy(34,3);
lcd_puts(asciicommandposition);

// while(1)
// {
//     PORTD.4 = ~PORTD.4; // debugging to figure out sampling rate
//     newposition = RDEncoder(0); // Reads encoder channel 0
// }
bytein = getchar();
if( bytein == 's')
{
    TIMSK=0x00; // disable timer 2 interrupt
    commandposition = positiongoal; // set Command position
    oldposition = 0; // Set initial position as 0;
    // Reset all 3 encoders
    RESEncoders(); // Set current position indirectly to 0;
    for (i = 0;i<2;i++)
    {
        putchar('m');
        putchar(0);
        putchar('p');
        putchar(0);
        putchar(0);
        putchar(0);
        putchar(0);
        putchar(0);
    }
    TIMSK=0x80; // enable timer 2 interrupt
    //TIMSK=0x81; // enable timer 0 & 2 interrupt
    WRPencilCommand(0b11111111); // Deactivate all outputs & Enable motor (logic inverted in hardware)
    //flag = 1; // start sending
}

}; // end of while(1)
// End of void main();
}

```