Lab Assignment 4: Inverted Pendulum

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 System Modeling

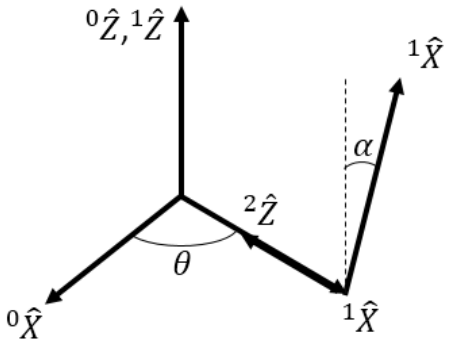

Figure 1: The co-ordinate frames of the inverted pendulum system

a. Physical Model

We model the physical system as a pair of rigid links, the horizontal arm and the pendulum mounted to the end of the arm, neglecting the presence of the encoder cable, using a lumped parameter model.

b. Assumptions

We make the following assumptions while modeling the system:

- All of the links are rigid - this allows us to neglect work done by the internal constraining forces of the system as well as losses due to collisions and deformations. This assumption very nearly holds because system is built using stiff metals and bearings that are well fitted.
- Air resistance is neglected - this is due to the small cross sectional areas and low velocities experienced by the system. Additionally this is neglected due to the difficulty in constructing an appropriate model for these losses.
- The axis of rotation of the horizontal arm is perfectly level with the ground - this assumption ensures that there will be no potential energy term associated with the arm.
- The presence of the encoder cable connected to the arm is neglected - We found that the cable has a large impact on our system however its influence would be very difficult to model because it is not fixed to the arm and because the forces it introduces are highly non linear.

Apart from the above list, the standard set of control systems assumptions are also made:

- The current driver acts instantaneously - the transients involved occur so quickly with respect to the system response time that we virtually have immediate response. Also, the motor, acting as a low pass filter, will reject the high frequency components of the driver.
- The DAQ has instantaneous and correct response - we ignore the quantization and discretization errors of the DAQ and further assume that offset errors have been accounted for. This is valid because our control loop runs at 100 Hz while the DAQ can be sampled at 1000 Hz. Additionally we believe that the DAQ has been calibrated and static errors have been removed.

c. Mathematical Modeling

Forwards Kinematics

Let ${}^0F$ represent the fixed world coordinate frame, with $\mathbf{{}^0\hat{z}}$ aligned with the motor shaft and with $\mathbf{{}^0\hat{x}}$ pointing along the initial direction of the horizontal arm as seen in figure 1 on page 2. Let the origin of ${}^0F$ be located to coincide with the motor axis and the axis of rotation of the pendulum. Let ${}^1F$, representing the local frame of the horizontal arm, be attached to the horizontal arm, with $\mathbf{{}^1\hat{z}}$ aligned with the motor shaft and $\mathbf{{}^1\hat{x}}$ aligned with the motor arm pointed towards the pendulum. Let ${}^2F$ be located such that its origin is along $\mathbf{{}^1\hat{x}}$ at a distance l_1, with $\mathbf{{}^2\hat{x}}$ pointing along the pendulum and $\mathbf{{}^2\hat{z}}$ along the $-\mathbf{{}^1\hat{x}}$ direction. Where l_1 is the distance from the axis of rotation of the motor to the center of mass of the pendulum along the $\mathbf{{}^1\hat{x}}$ direction, shown in figure 7. Let θ denote the angle between $\mathbf{{}^0\hat{x}}$ and $\mathbf{{}^1\hat{x}}$ and let α denote the angle between $\mathbf{{}^1\hat{z}}$ and $\mathbf{{}^2\hat{x}}$ measured clockwise when viewed from $\mathbf{{}^1\hat{x}}$. We assume that the motor and the horizontal arm are rigidly connected and can be treated as a single inertia for purposes of our energy calculations below.

Then:

$$ {}^0F = R_z(\theta)T_x(l_1)R_y(-\pi/2)R_z(\alpha)\left({}^2F\right) $$

Let l_{2c} be the distance along $\mathbf{{}^2x}$ from the origin of ${}^2F$ to the center of mass of the pendulum, shown in figure 5. Then the location of the center of mass of the pendulum in the world coordinate system $\mathbf{{}^0P_{2c}}$ is given as:

$$ \mathbf{{}^0P_{2c}} = R_z(\theta)T_x(l_1)R_y(-\pi/2)R_z(\alpha)\left(\mathbf{{}^2P_{2c}}\right) = R_z(\theta)T_x(l_1)R_y(-\pi/2)R_z(\alpha)\left[l_{2c},0,0\right]^T $$

$$ \mathbf{{}^0P_{2c}} = \begin{bmatrix} -l_{2c}\sin(\theta)\sin(\alpha) + l_1\cos(\theta) \\ l_{2c}\cos(\theta)\sin(\alpha) + l_1\sin(\theta) \\ l_{2c}\cos(\alpha) \end{bmatrix} $$

From this position we can then determine the linear velocity of the center of mass of the pendulum in the world frame denoted $\mathbf{{}^0\dot{P}_{2c}}$. Let J denote the matrix of partial derivatives of $\mathbf{{}^0P_{2c}}$ with respect to θ and α. Then the $\mathbf{{}^0\dot{P}_{2c}}$ is given by:

$$ \mathbf{{}^0\dot{P}_{2c}} = J\begin{bmatrix} \dot{\theta} \\ \dot{\alpha} \end{bmatrix} = \begin{bmatrix} -l_{2c}sin(\alpha)\cos(\theta) - l_1\sin(\theta) & -l_{2c}\sin(\theta)\cos(\alpha) \\ -l_{2c}sin(\alpha)\sin(\theta) + l_1\cos(\theta) & l_{2c}\cos(\theta)\cos(\alpha) \\ 0 & -l_{2c}\sin(\alpha) \end{bmatrix}\begin{bmatrix} \dot{\theta} \\ \dot{\alpha} \end{bmatrix} $$

We can also determine the angular velocity of the pendulum about its center of mass in the pendulum fixed frame ${}^2F$ by the following:

$$ \omega_{\mathbf{2}} = \dot{\theta}(\mathbf{{}^1\hat{z}}) + \dot{\alpha}(\mathbf{{}^2\hat{z}}) = (R_y(-\pi/2)R_z(\alpha))^{-1}\left[0,0,\dot{\theta}\right]^T + \dot{\alpha}(\mathbf{{}^2\hat{z}}) $$

$$ \mathbf{{}^2}\omega_{\mathbf{2}} = \begin{bmatrix} \dot{\theta}\cos(\alpha) \\ -\dot{\theta}\sin(\alpha) \\ \dot{\alpha} \end{bmatrix} $$

Rather than concern ourselves with the forward kinematics of the horizontal arm, we assume that its only motion is a rotation about $\mathbf{{}^0\hat{z}}$ which allows us to simply apply the parallel axis theorem to obtain its moment of inertia about $\mathbf{{}^0\hat{z}}$.

System Energy

Our system is composed of two rigid bodies, the horizontal arm and the pendulum. Each has an energy denoted E_1 and E_2 respectively. Using the forward kinematics derived above we can compute the kinetic and potential energies associated with each body as a function of θ, α, $\dot{\theta}$, and $\dot{\alpha}$.

$$E_1 = T_1 + V_1 \qquad E_2 = T_2 + V_2$$

By our assumption that ${}^{\mathbf{1}}\hat{\mathbf{z}}$ is parallel to the direction of gravity, the term V_1 can be taken to be 0. Furthermore, rather than expand T_1 into linear and rotational components, we can simply use a single rotation term with a modified moment of inertia, I_{1z1}. Let l_{1c} be the distance along ${}^{\mathbf{1}}\hat{\mathbf{x}}$ from the origin to the center of mass of the rotor ${}^{1}P_{1c}$, and let m_1 be the mass of the horizontal arm, and let I_{rotor} be the moment of inertia of the rotor. Then the total energy of the horizontal arm is given by:

$$E_1 = I_{1z1}\dot{\theta}^2 = \left(I_{1zzc} + I_{rotor} + m_1 l_{1c}^2\right)\dot{\theta}^2$$

Because the pendulum is perpendicular to the horizontal arm which is perpendicular to the ground, the potential energy of the pendulum, V_2 only depends on α, the distance from the axis of rotation to the center of mass along ${}^{\mathbf{2}}\hat{\mathbf{x}}$, l_{2c}, the mass of the pendulum, m_2, and the acceleration due to gravity, g. Without loss of generality, we assume that zero potential energy occurs for $\alpha = 0$.

$$V_2 = \left(\cos(\alpha) - 1\right) l_{2c} m_2 g$$

The expression for kinetic energy of the pendulum, T_2 is broken into a translational and a rotational component. The translation kinetic energy of the pendulum T_{2T} is given by the following expression:

$$T_{2T} = \frac{1}{2}m_2\left({}^{\mathbf{0}}\dot{\mathbf{P}}_{\mathbf{2c}}\right)^2 = \frac{1}{2}m_2\left({}^{\mathbf{0}}\dot{\mathbf{P}}_{\mathbf{2c}}\right)^T\left({}^{\mathbf{0}}\dot{\mathbf{P}}_{\mathbf{2c}}\right) = \frac{1}{2}m_2\left[\dot{\theta},\dot{\alpha}\right]J^T J\begin{bmatrix}\dot{\theta}\\ \dot{\alpha}\end{bmatrix}$$

$$T_{2T} = \dot{\theta}^2\left(l_{2c}^2\sin^2(\alpha) + l_1^2\right) + 2l_1 l_{2c}\dot{\theta}\dot{\alpha}\cos(\alpha) + \left(\dot{\alpha}l_{2c}\right)^2$$

The rotational kinetic energy of the pendulum is given by the following expression where I_2 is the 3 by 3 moment of inertia tensor of the pendulum.

$$T_{2R} = \frac{1}{2}{}^{\mathbf{2}}\omega_{\mathbf{2}}{}^{T} I_2 {}^{\mathbf{2}}\omega_{\mathbf{2}} = \frac{1}{2}\left[\dot{\theta}^2\left(I_{2xx}\cos^2(\alpha) - 2I_{2xy}\sin(\alpha)\cos(\alpha) + I_{2yy}\sin^2(\alpha)\right) + 2I_{2xz}\dot{\theta}\dot{\alpha}\sin(\alpha) + I_{2zz}\dot{\alpha}^2\right]$$

As discussed later I_{2xy} and I_{2yz} are very nearly 0 and have been neglected for the rest of the derivation.

$$T_{2R} = \frac{1}{2}\left[\dot{\theta}^2\left(I_{2xx}\cos^2(\alpha) + I_{2yy}\sin^2(\alpha)\right) + I_{2zz}\dot{\alpha}^2\right] - I_{2xz}\dot{\theta}\dot{\alpha}\cos(\alpha)$$

Lagrange's Equation and non-Linear Equations of Motion

The Lagrange method can be used to compute the equations of motion of complex dynamical systems using the total system energy derived above along with the generalized coordinates of the system q_i and generalized non-conservative forces on the system Q_{NCi}. These non-conservative forces are the damping forces exerted on both the pendulum and the horizontal arm determined by b_α and b_θ respectively, the motor torque exerted on the horizontal arm, τ_m, and the force of coulomb friction exerted on both the pendulum and the horizontal arm denoted $\tau_{C\alpha}$ and $\tau_{C\theta}$ respectively.

$$\frac{d}{dt}\frac{\partial\mathcal{L}}{\partial\dot{q}_i} - \frac{\partial\mathcal{L}}{\partial q_i} = Q_{NCi}$$

$$\frac{d}{dt}\frac{\partial\mathcal{L}}{\partial\dot{\theta}} - \frac{\partial\mathcal{L}}{\partial\theta} = -b_\theta\dot{\theta} + \tau_m + \tau_{C\theta} \qquad \frac{d}{dt}\frac{\partial\mathcal{L}}{\partial\dot{\alpha}} - \frac{\partial\mathcal{L}}{\partial\alpha} = -b_\alpha\dot{\alpha} + \tau_{C\alpha}$$

For our friction model here we must make a simplifying assumption to avoid having to estimate internal constraint forces. In a correct model to handle static friction correctly if the joint angle, $\dot{\theta}$ or $\dot{\alpha}$ is 0, then we must estimate the torque exerted on the joint attempting to overcome static friction. This torque can really be thought of as the constraint torque required to keep the angle fixed, but our Lagrange method sacrifices knowledge of these constraint forces to simplify the equation of motion. To avoid having to estimate this

constraint torque we make a simplifying assumption. We only consider the external torques acting on each joint, rather than the internal forces at the joint. These external torques are known, simply the motor torque, τ_m for the horizontal arm and the torque exerted by gravity on the pendulum τ_g.

$$\tau_g = g m_2 l_{2c} \sin(\alpha)$$

With this assumption we can use our previous coulomb friction model with no change. Note that we again use two different frictional torques for the horizontal arm because our previous experiments with the motor discovered significant differences in friction in the forwards and reverse directions. The pendulum has been modeled with a single friction torque because we have no reason to believe that the direction of rotation changes friction.

$$\tau_{C\alpha} = \begin{cases} -\tau_g & \text{if } \dot{\alpha} = 0 \text{ and } |\tau_g| < \tau_{F\alpha} \\ -sgn(\tau_g) \cdot \tau_{FA} & \text{if } \dot{\alpha} = 0 \text{ and } |\tau_{motor}| > \tau_{F\alpha} \\ -sgn(\dot{\alpha}) \cdot \tau_{F\alpha} & \text{if } \dot{\alpha} \neq 0 \end{cases}$$

$$\tau_{C\theta} = \begin{cases} -\tau_m & \text{if } \dot{\theta} = 0 \text{ and } 0 \leq \tau_{motor} < \tau_{F\theta F} \\ -\tau_m & \text{if } \dot{\theta} = 0 \text{ and } -\tau_{F\theta R} \leq \tau_m < 0 \\ -sgn(\tau_m) \cdot \tau_{F\theta F} & \text{if } \dot{\theta} = 0 \text{ and } \tau_m > \tau_{F\theta F} \\ -sgn(\tau_m) \cdot \tau_{F\theta R} & \text{if } \dot{\theta} = 0 \text{ and } \tau_m \leq -\tau_{F\theta R} \geq \\ -sgn(\dot{\theta}) \cdot \tau_{F\theta F} & \text{if } \dot{\theta} > 0 \\ -sgn(\dot{\theta}) \cdot \tau_{F\theta R} & \text{if } \dot{\theta} < 0 \end{cases}$$

The right hand side of the equation contains partial derivatives of the Lagrangian, $\mathcal{L}$, which is given below as the difference of the total kinetic energy of the system, T, and the total potential energy of the system V. Using the expressions derived above for the energies of each component of the system we can construct an expression for $\mathcal{L}$ and take partial derivatives

$$\mathcal{L} = T - V = T_1 + T_2 - (V_1 + V_2) = T_1 + T_{2R} + T_{2T} - V_2$$

$$\mathcal{L} = \frac{1}{2}\dot{\theta}^2 \left(I_{1z1} + C_2 \sin^2(\alpha) + I_{2xx} \cos^2(\alpha) + m_2 l_1^2\right) + \frac{1}{2} C_1 \dot{\alpha}^2 + C_3 \dot{\theta}\dot{\alpha} \cos(\alpha) - C_4 \cos(\alpha) + C_4$$

$$C_1 = I_{2zz} + m_2 l_{2c}^2 \qquad C2 = I_{2yy} + m_2 l_{2c}^2 \qquad C_3 = m_2 l_1 l_{2c} - I_{2xz} \qquad C_4 = l_{2c} m_2 g$$

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\theta}} = \ddot{\theta}\left(I_{1z1} + C_2 \sin^2(\alpha) I_{2xx} \cos^2(\alpha) + m_2 l_1^2\right) + 2\left(C_2 - I_{2xx}\right)\dot{\theta}\dot{\alpha} \sin(\alpha)\cos(\alpha) + C_3 \ddot{\alpha}\cos(\alpha) - C_3 \dot{\alpha}^2 \sin(\alpha)$$

$$\frac{\partial \mathcal{L}}{\partial \theta} = 0$$

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\alpha}} = C_1 \ddot{\alpha} + C_3 \ddot{\theta}\cos(\alpha) - C_3 \dot{\theta}\dot{\alpha}\sin(\alpha)$$

$$\frac{\partial \mathcal{L}}{\partial \alpha} = \dot{\theta}^2\left(C_2 \sin(\alpha)\cos(\alpha) - I_{2xx}\cos(\alpha)\sin(\alpha)\right) - C_3 \dot{\theta}\dot{\alpha} sin(\alpha) + C_4 \sin(\alpha)$$

$$\begin{aligned} \ddot{\theta}\left(I_{1z1} + C_2 \sin^2(\alpha) + I_{2xx}\cos^2(\alpha) + m_2 l_1^2\right) + 2(C_2 - I_{2xx})\dot{\theta}\dot{\alpha}\sin(\alpha)\cos(\alpha) \\ + C_3 \ddot{\alpha}\cos(\alpha) - C_3 \dot{\alpha}^2 \sin(\alpha) = -b_\theta \dot{\theta} + \tau_m + \tau_{C\theta} \end{aligned} \tag{1}$$

$$C_1 \ddot{\alpha} + C_3 \ddot{\theta}\cos(\alpha) - (C_2 - I_{2xx})\dot{\theta}^2 \sin(\alpha)\cos(\alpha) - C_4 \sin(\alpha) = -b_\alpha \dot{\alpha} + \tau_{C\alpha} \tag{2}$$

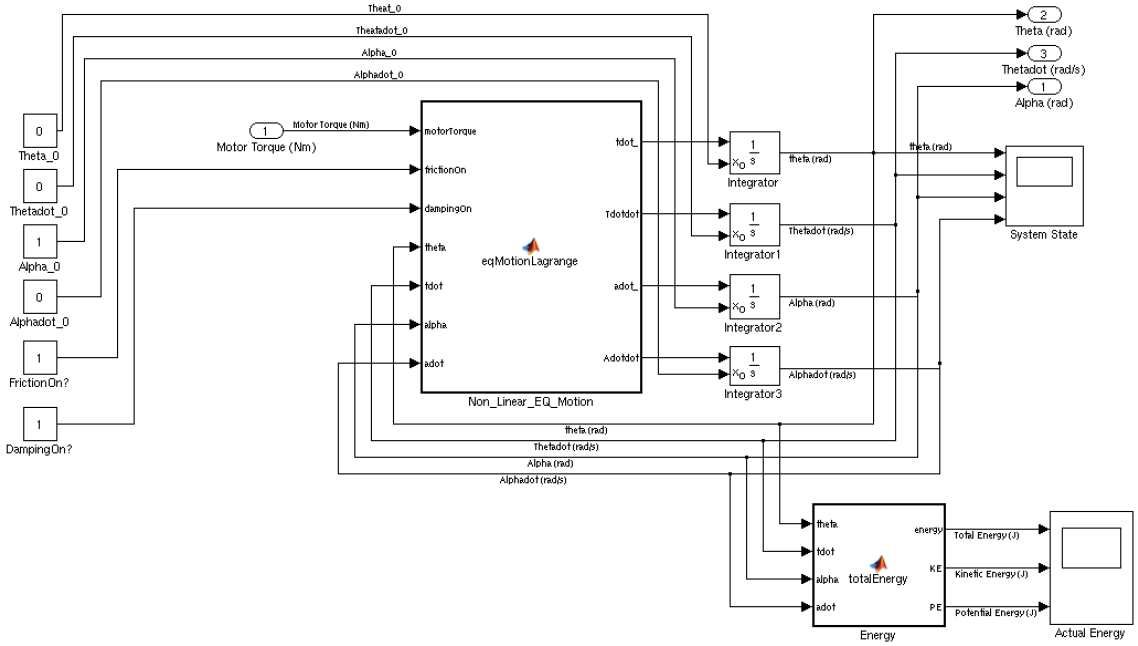

Figure 2: Block Diagram of Non-Linear System Model

The pair of non-linear differential equations can be used to numerically model the behavior of the system, but they are not particularly useful for actual controller design. Both state space controls and our traditional techniques require the system to be linearized around a particular operating point. This allows us to generate a transfer function to approximate the behavior of the system and lets us construct root locii and bode plots to quantify system behavior.

Electrical Driver and Motor Model

We also modeled the effects of motor back emf which serves to limit the maximum angular velocities that the arm can achieve. For the given command current, we compute the back emf across the motor according to its present angular velocity, $\dot{\theta}$ and the back-emf constant K_b, and compare the required supply voltage with the actual supply voltage, V_s. If the supply voltage is insufficient, we use the supply voltage and the back-emf to compute the voltage across the motor and from that compute the actual motor current. Otherwise motor current is directly proportional to command voltage.

d.

See figure 2 for the block diagram of the non-linear model within Simulink. See the included code in the Appendix for the implemented functions.

e.

Linearized Equations of Motion

We will linearize the equations of motion about $\theta = 0$, $\alpha = 0$. By taylor expansion about this point and dropping higher order terms we obtain:

$$\sin(\alpha) \approx 0 \qquad \cos(\alpha) \approx \alpha \qquad \sin(\theta) \approx 0 \qquad \cos\theta \approx \theta$$

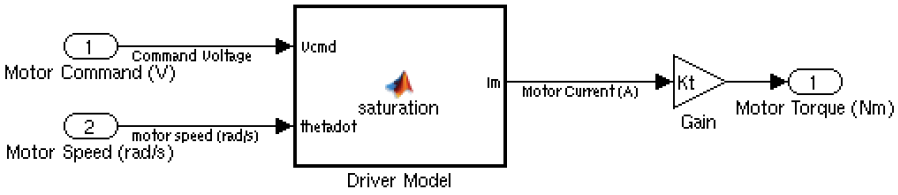

Figure 3: Block Diagram of Driver and Electrical Motor Model

We note that the coulomb friction terms from the previous non-linear equations must be dropped completely as there is no reasonable linearization of the effects of friction. We also note that we expect $\dot{\theta}$ and $\dot{\alpha}$ to be relatively small, which implies:

$$\dot{\theta}\dot{\alpha} \approx 0$$

Applying these to equations (1) and (2) yields (3) and (4) respectively.

$$\ddot{\theta}\left(I_{1z1} + I_{2xx} + m_2 l_1^2\right) + \ddot{\alpha}C_3 = -b_\theta\dot{\theta} + \tau_m \tag{3}$$

$$\ddot{\alpha}C_1 + \ddot{\theta}C_3 - \alpha C_4 = -b_\alpha\dot{\alpha} \tag{4}$$

Using equations (3) and (4) together we can generate two independent equations for $\ddot{\alpha}$ and $\ddot{\theta}$.

$$C_5 = (I_{1z1} + I_{2xx} + m_2 l_1^2)$$

$$\ddot{\theta} = \frac{C_3 b_\alpha\dot{\alpha} - C_3 C_4\alpha - C_1 b_\theta\dot{\theta} + C_1\tau_m}{C_5 C_1 - C_3^2} \tag{5}$$

$$\ddot{\alpha} = \frac{-C_5 b_\alpha\dot{\alpha} + C_3 b_\theta\dot{\theta} + \alpha C_4 C_5 - C_3\tau_m}{C_5 C_1 - C_3^2} \tag{6}$$

State Space Model

State Space Representation models a dynamical system as a set of linear, time-varying differential equations. What is known as the state of the system is a set of variables, the generalized coordinates, q_i that completely describe the internal state of the system. For the inverted pendulum we choose the coordinates:

$$\mathbf{X}(t) = [q_1(t), q_2(t), q_3(t), q_4(t)]^T = [q_1(t), \dot{q}_1(t), q_3(t), \dot{q}_3(t)]^T = [\theta(t), \dot{\theta}(t), \alpha, \dot{\alpha}(t)]^T$$

In this generalized coordinate system, we compute the derivative of the state, $\dot{X}$ as a function of the current state $\mathbf{X}$ and the inputs $\mathbf{u}$.

$$\dot{\mathbf{X}}(\mathbf{t}) = [\dot{q}_1(t), \dot{q}_2(t), \dot{q}_3(t), \dot{q}_4(t)]^T = [\dot{q}_1(t), \ddot{q}_1(t), \dot{q}_3(t), \ddot{q}_3(t)]^T = [\dot{\theta}(t), \ddot{\theta}(t), \dot{\alpha}(t), \ddot{\alpha}(t)]^T$$

Because our system is linear this relationship can be related through the State Matrix A and the Input Matrix B which map the current state and external inputs into the system into the change in state through the following equation.

$$\dot{\mathbf{X}}(t) = A(t)\mathbf{X}(\mathbf{t}) + B(t)\mathbf{u}(t)$$

$$A_{ij} = \frac{\partial\dot{q}_j}{\partial q_i}(t) \qquad B_i = \frac{\partial\dot{q}_i}{\partial u_i}(t)$$

The actual output that we wish to control is determined from $\dot{\mathbf{X}}(t)$ and from the inputs $\mathbf{u}(t)$ through the Output Matrix, $C(t)$ and the Direct Transmission Matrix $D(t)$. Then our output $\mathbf{Y}(t)$ can be represented as follows:

$$\mathbf{Y}(t) = C(t)\dot{\mathbf{X}}(t) + D(t)\mathbf{u}(t) = C(t)\left\{A(t)\mathbf{X}(\mathbf{t}) + B(t)\mathbf{u}(t)\right\} + D(t)\mathbf{u}(t)$$

As noted above the matrices A, B, C, and D may in general be functions of time, however because our system is time invariant, these matrices become independent of time. The notation below has been simplified accordingly. The matrices A and B may be obtained directly from our linearized equations of motion above.

$$A = \begin{bmatrix} \frac{\partial\dot{\theta}}{\partial\theta} & \frac{\partial\dot{\theta}}{\partial\dot{\theta}} & \frac{\partial\dot{\theta}}{\partial\alpha} & \frac{\partial\dot{\theta}}{\partial\dot{\alpha}} \\ \frac{\partial\ddot{\theta}}{\partial\theta} & \frac{\partial\ddot{\theta}}{\partial\dot{\theta}} & \frac{\partial\ddot{\theta}}{\partial\alpha} & \frac{\partial\ddot{\theta}}{\partial\dot{\alpha}} \\ \frac{\partial\dot{\alpha}}{\partial\theta} & \frac{\partial\dot{\alpha}}{\partial\dot{\theta}} & \frac{\partial\dot{\alpha}}{\partial\alpha} & \frac{\partial\dot{\alpha}}{\partial\dot{\alpha}} \\ \frac{\partial\ddot{\alpha}}{\partial\theta} & \frac{\partial\ddot{\alpha}}{\partial\dot{\theta}} & \frac{\partial\ddot{\alpha}}{\partial\alpha} & \frac{\partial\ddot{\alpha}}{\partial\dot{\alpha}} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{\partial\ddot{\theta}}{\partial\dot{\theta}} & \frac{\partial\ddot{\theta}}{\partial\alpha} & \frac{\partial\ddot{\theta}}{\partial\dot{\alpha}} \\ 0 & 0 & 0 & 1 \\ 0 & \frac{\partial\ddot{\alpha}}{\partial\dot{\theta}} & \frac{\partial\ddot{\alpha}}{\partial\alpha} & \frac{\partial\ddot{\alpha}}{\partial\dot{\alpha}} \end{bmatrix}$$

$$\frac{\partial\ddot{\theta}}{\partial\ddot{\theta}} = -\frac{C_1 b_\theta}{C_5C_1 - C_3^2} \qquad \frac{\partial\ddot{\theta}}{\partial\alpha} = -\frac{C_3C_4}{C_5C_1 - C_3^2} \qquad \frac{\partial\ddot{\theta}}{\partial\dot{\alpha}} = \frac{C_3 b_\alpha}{C_5C_1 - C_3^2}$$

$$\frac{\partial\ddot{\alpha}}{\partial\dot{\theta}} = \frac{C_3 b_\theta}{C_5C_1 - C_3^2} \qquad \frac{\partial\ddot{\alpha}}{\partial\alpha} = \frac{C_4C_5}{C_5C_1 - C_3^2} \qquad \frac{\partial\ddot{\alpha}}{\partial\dot{\alpha}} = -\frac{C_5 b_\alpha}{C_5C_1 - C_3^2}$$

$$B = \begin{bmatrix} \frac{\partial\dot{\theta}}{\partial\tau_m} \\ \frac{\partial\ddot{\theta}}{\partial\tau_m} \\ \frac{\partial\dot{\alpha}}{\partial\tau_m} \\ \frac{\partial\ddot{\alpha}}{\partial\tau_m} \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{C_1}{C_5C_1 - C_3^2} \\ 0 \\ -\frac{C_3}{C_5C_1 - C_3^2} \end{bmatrix}$$

We wish to stabilize the pendulum vertically, so the output be are interested in is precisely $\alpha(t)$. Thus C becomes simply:

$$C = [0, 0, 1, 0]^T$$

For this system, the only external input, the motor torque $\tau_m(t)$, does not directly influence the output, $\mathbf{Y}(t)$ except for its influence on $\dot{\mathbf{X}}(t)$ making D simply the zero matrix.

$$D = [0, 0, 0, 0]^T$$

System Transfer Functions

To generate the transfer functions it is more straight forwards to proceed from equations (3) and (4) than from (5) and (6). Applying the Laplace transform (3) and (4) yields and performing arithmetic operation to isolate $\theta(s)$ and $\alpha(s)$ yields the following transfer functions from motor torque, $\tau(s)$ to $\theta(s)$ and $\alpha(s)$ respectively.

$$\frac{\theta(s)}{\tau_m(s)} = \frac{s^2C_1 + sb_\alpha - C_4}{\left(s^2C_1 + sb_\alpha - C_4\right)\left(s^2C_5 + sb_\theta\right) - s^4C_3^2} \tag{7}$$

$$\frac{\alpha(s)}{\tau_m(s)} = \frac{-s^2C_3}{\left(s^2C_1 + sb_\alpha - C_4\right)\left(s^2C_5 + sb_\theta\right) - s^4C_3^2} \tag{8}$$

If we were to neglect the effects of damping, then (7) and (8) would reduce to the following:

$$\frac{\theta(s)}{\tau_m(s)} \approx \frac{s^2C_1 - C_4}{s^2\left[s^2\left(C_5C_1 - C_3^3\right) - C_4C_5\right]}$$

$$\frac{\alpha(s)}{\tau_m(s)} \approx \frac{-s^2 C_3}{s^2\left[s^2\left(C_5 C_1 - C_3^3\right) - C_4 C_5\right]}$$

In the simplified transfer function for $\alpha(s)$ above it appears as if we can cancel the s^2's, but this not the case as we see when we examine equation (8).

2 Parameter Identification

The inverted pendulum system includes many parameters related to the mechanical and electrical properties of the system. To determine these, we used a combination of manufacturer data sheets, CAD modeling, physical measurements, and experimental regression.

Most of the relevant motor parameters were previously determined in the DC motor lab. In that case, most of the parameters were provided by the manufacturer, but we performed regression on experimental data to fit our model with the test data and found that the true values were different from those on the data sheet. The relevant values are shown in table 1. An abbreviated excerpt from our DC motor lab is given below to show our procedure for determining these values.

	Data-Sheet Values	Fit Values
J_{motor} $\left(\frac{N \cdot s^2}{m}\right)$	$8.5 \cdot 10^{-6}$	$3.50514 \cdot 10^{-5}$
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 1: DC Motor Final Parameter Values vs Original Data Sheet Values

To determine the parameters of the motor we 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.

With this model and some intelligent guessing for new parameter values, we arrived at an excellent fit shown in figure 4. The most notable feature of these values is the large difference between the torque of friction in the forwards and reverse directions.

With the motor parameters largely known already, we focused on determining the parameters of the new components comprising the pendulum and horizontal arm. To ensure accuracy, we used multiple methods and compared results. For example, the mass of every component was determined on the scale provided in the mechatronics lab. We then built CAD models of every component in SolidWorks using a combination of engineering drawings, physical measurements, and already built models from McMaster-Carr. By setting the material properties correctly, SolidWorks can determine the mass of every part. We compared this calculated value to the measured values and they all matched within the limits of resolution of the scale. This gave us confidence in the ability of SolidWorks to calculate the other necessary parameters.Therefore, we built assemblies of the pendulum, the horizontal arm and complete system, and we used SolidWorks to calculate the masses, centers of mass, and moments of inertia that we needed. Figures 5, 6, and 7 show the CAD assemblies. The inertia matrix for the pendulum and other relevant values are given below. Note that the axes labels in the SolidWorks models are not the same as those in our derived equations.

$$\begin{vmatrix} \bar{I}_{xx} & \bar{I}_{xy} & \bar{I}_{xz} \\ \bar{I}_{yx} & \bar{I}_{yy} & \bar{I}_{yz} \\ \bar{I}_{zx} & \bar{I}_{zy} & \bar{I}_{zz} \end{vmatrix}_{Pendulum} = \begin{vmatrix} 8.1 \cdot 10^{-6} & 2.03 \cdot 10^{-5} & 0 \\ 2.03 \cdot 10^{-5} & 3.711 \cdot 10^{-4} & 0 \\ 0 & 0 & 3.646 \cdot 10^{-4} \end{vmatrix}$$

Translating from the coordinate system established in Solid Works to our coordinate system yields the following moment of inertia tensor:

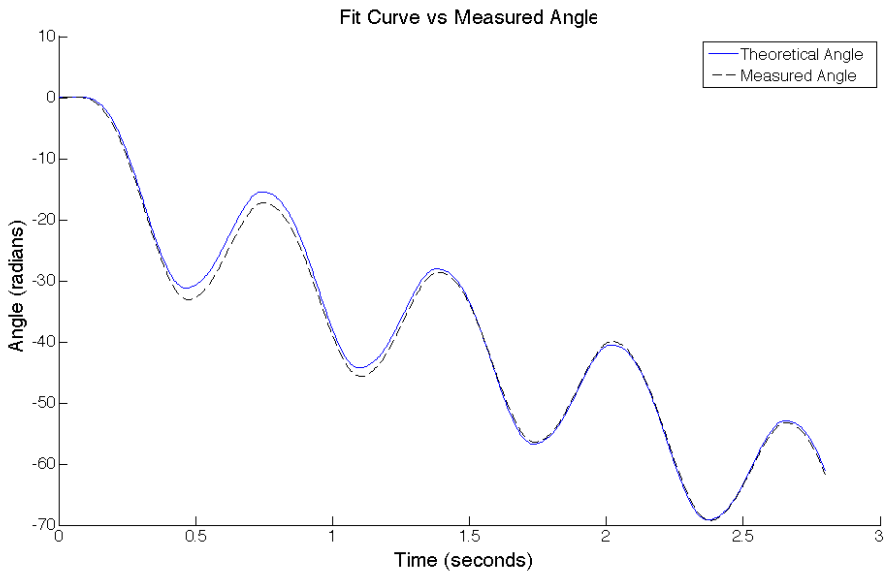

Figure 4: Predicted behavior vs Actual behavior using fit parameter values and enhanced modeling

$$I_2 = \begin{bmatrix} 8.1 \times 10^{-6} & 0 & 2.03 \times 10^{-5} \\ 0 & 3.711 \times 10^{-4} & 0 \\ 2.03 \times 10^{-5} & 0 & 3.646e{-}4 \end{bmatrix} kg \cdot m^2$$

$$m_1 = 0.1481kg \quad m_2 = 0.0820kg$$

$$l_1 = 0.10826m \quad l_{2C} = 0.08568m$$

$$I_{1zt} = \left(6.78 \times 10^{-4} + 3.50514 \times 10^{-5}\right) \, kg \cdot m^2 = 7.1305 \cdot 10^{-4} kg \cdot m^2$$

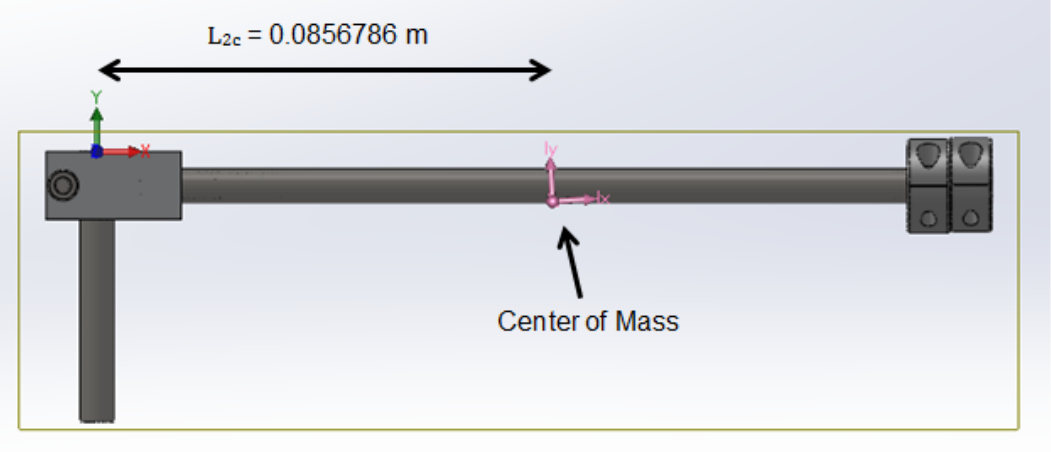

Figure 5: SolidWorks Model of Pendulum

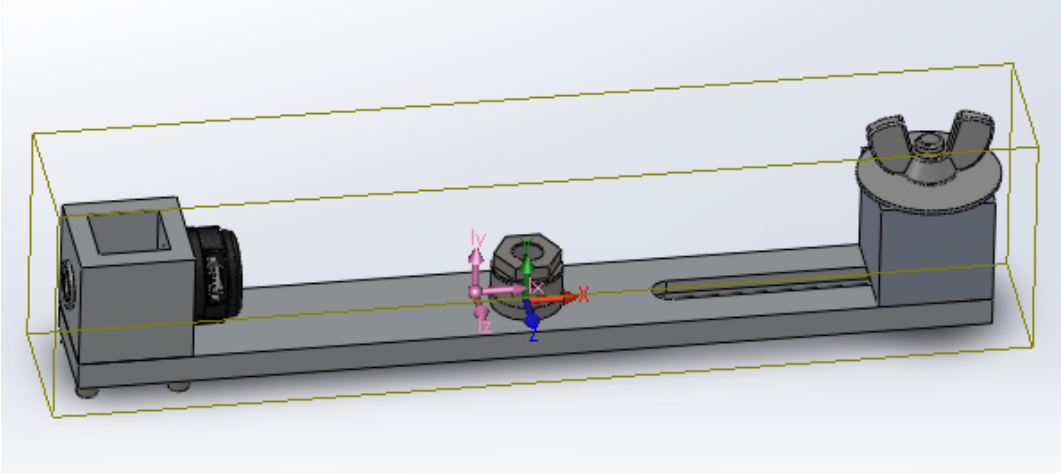

Figure 6: SolidWorks Model of Horizontal Link

For the pendulum we assume that frictional torque, $\tau_{F\alpha}$, and the viscous damping coefficient, b_α are the same in each direction. These terms for the pendulum were estimated in an experiment where the horizontal arm was fixed in place and the pendulum was allowed to swing freely from an initial angle. By comparing the decaying envelope, extracted as the maximum and minimum of each oscillation, of the actual system motion with a similar plot generated from a simulated model, we formed a rough estimate of these terms. MATLAB initialization files containing these parameters can be found in the appendix.

$$b_\alpha = 6.5 \cdot 10^{-6}\, N \cdot s \qquad \tau_{F\alpha} = 7.0 \cdot 10^{-6}\, N \cdot m$$

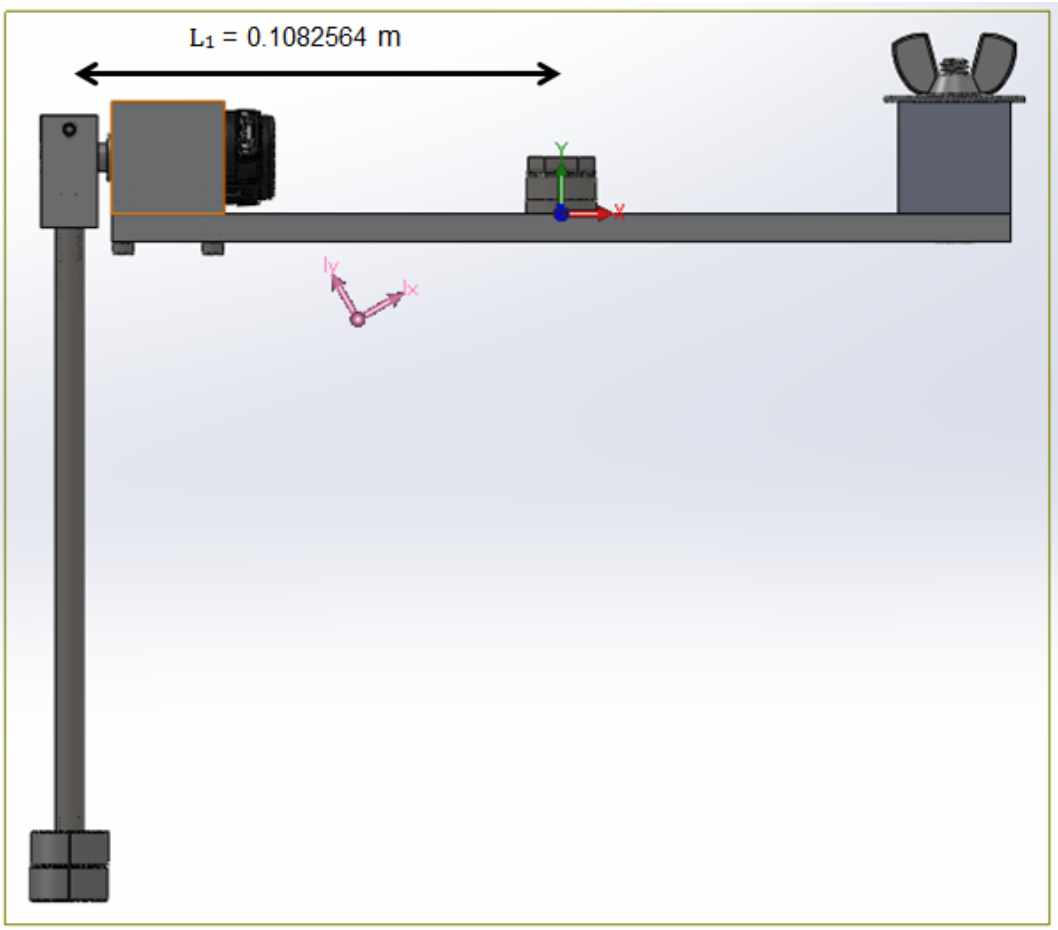

Figure 7: SolidWorks Model of Assembled Pendulum and Horizontal Link

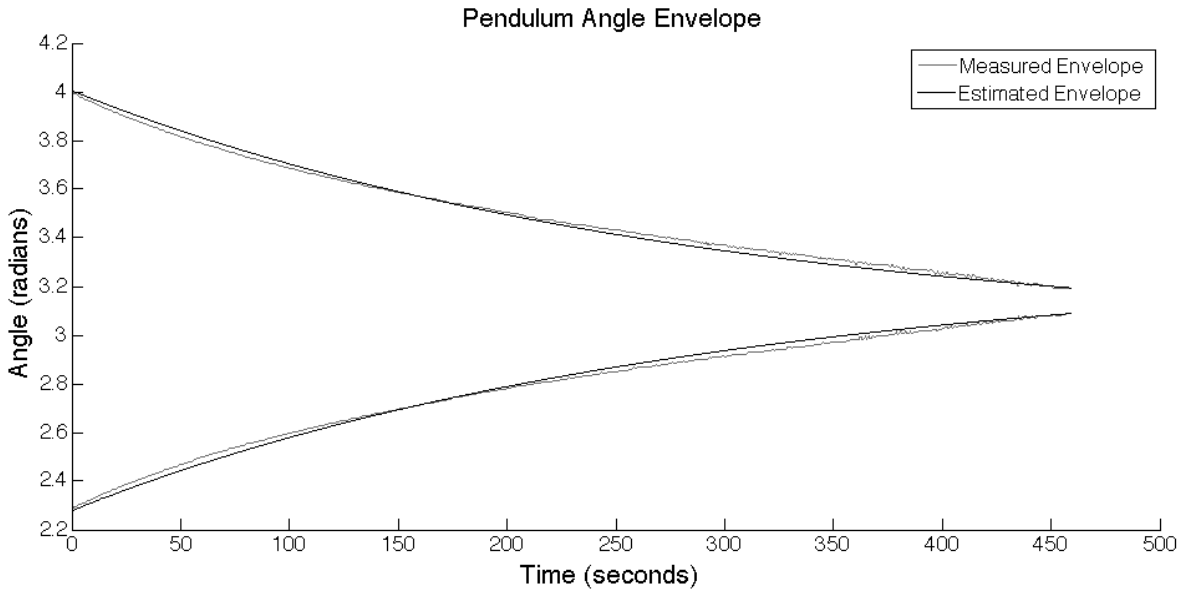

Figure 8: Actual Decaying Envelope of motion vs Simulated Envelope

3 Comparing Linear and Non-Linear Models

Figures 9 and 10 show the pendulum and horizontal arm angle response from the nonlinear model compared to experimental data. The data was obtained by allowing the pendulum to oscillate freely from a starting angle and then running the simulink model with the same initial conditions. The predicted pendulum angle agrees with the experimental data quite well for the first few periods, but quickly deviates. Interestingly, after several periods the experimental angle doesn't decay as quickly as the predicted angle, but the decay accelerates after several more periods and the pendulum comes to rest much sooner in the real system than in the model. A possible cause of this is the force applied by the encoder cable on the horizontal arm, which is unmodelled and very nonlinear and unpredictable. This is especially apparent in figure 10, where the horizontal arm prediction and real behavior were very far off.

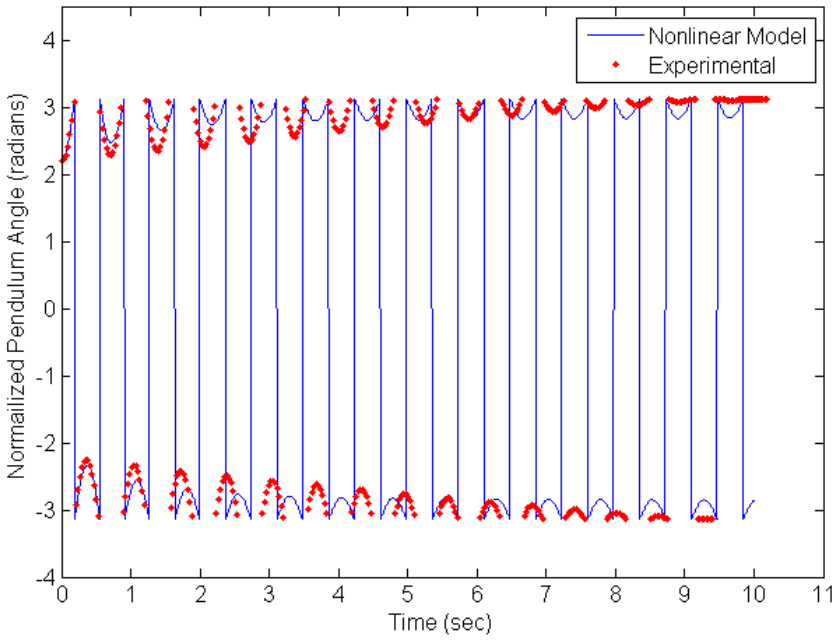

Figure 9: Comparison of Normalized Pendulum Angle From the Nonlinear Model and Experimental Data

Figures 11 and 12 show the pendulum and horizontal arm angle response from the linearized model compared to experimental data. The predicted pendulum angle agrees with the experimental quite well for several periods. Somewhat surprisingly, the linear model matches the experimental data longer than the nonlinear model did. However, the linear model decays extremely slowly. This is largely because the friction terms are no longer present, and friction on the horizontal arm played a large role in reducing the energy of the system. While the linear pendulum angle seems to perform reasonably well initially, the linear horizontal arm angle does not agree with the test data at all. If the angle data was not normalized, the predicted arm angle in figure 12 would grow without bound. A possible cause of this is that our linear model does not incorporate all of the friction and damping forces that act on the horizontal arm, and the encoder cable applies significant resistance to rotation on the arm.

To investigate the effect that the encoder cord has on the system, we ran tests with the cable removed. This meant that we could only get data on the horizontal arm angle. To ensure consistency, we raised the pendulum to a horizontal position before releasing it, then ran the simulink model from a horizontal starting

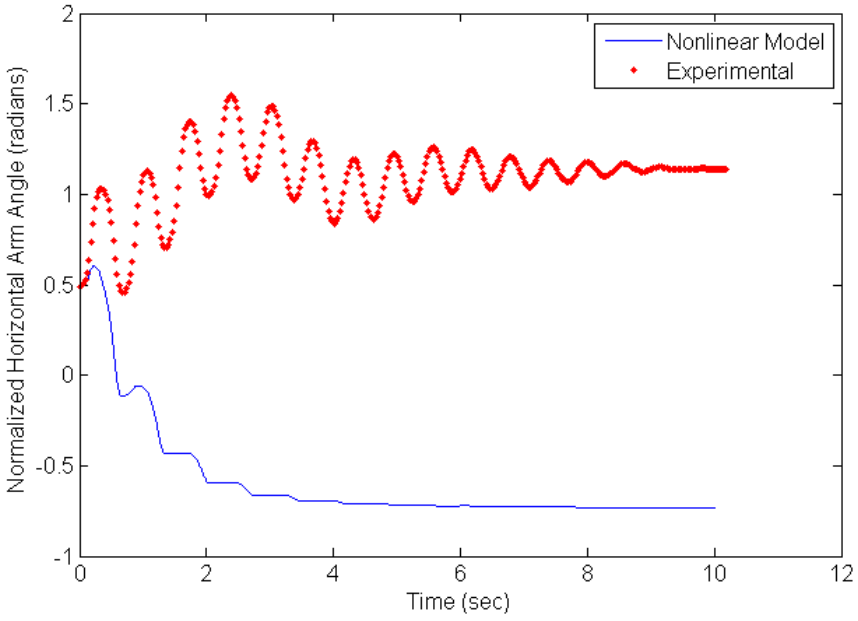

Figure 10: Comparison of Normalized Horizontal Arm Angle From the Nonlinear Model and Experimental Data

position. Figure 13 shows the results of such a test. The initial transient phase of the two datasets still doesn't agree, and the real system decays much slower, but it's interesting that both move towards +-π (though the normalization of the angles makes this somewhat hard to see). This suggests to us that the cord is having a large effect on the system, but there are other factors which haven't been considered (such as tilt of the table).

Figure 14 shows the predicted pendulum angle from the nonlinear model and from the linearized model. Just as when comparing both models to the experimental data, there is some agreement in the first few periods, but then the two models diverge. This is largely because the lack of friction in the linearized model causes it to decay much more slowly. There is also a noticeable phase difference as time increases because the nonlinear model predicts a slightly longer period of oscillation.

It is clear that there are weakness in the linear model based on the weak agreement with the nonlinear model and real system. However, this is largely because the linearization assumptions are based on the pendulum being very close to the vertical equilibrium point for use with a balancing controller. While we linearized the system about the bottom position for the purposes of comparing to the test data, the assumptions that the small angle theorem would hold and that $\dot{\theta}$ and $\dot{\alpha}$ would be small were not necessarily true when in a free swinging situation instead of being balanced. Therefore, we believe that our linearization assumptions are valid as long as we only apply the linearization within a small angle range and have a balance controller active to minimize $\dot{\theta}$ and $\dot{\alpha}$. The greatest effect of the linearization was the dropping of the friction terms (which was more of a practical necessity rather than an assumption). This had a large effect on the decay of the systems oscillations, but with our balance controller active, this assumption should not have a significantly negative effect.

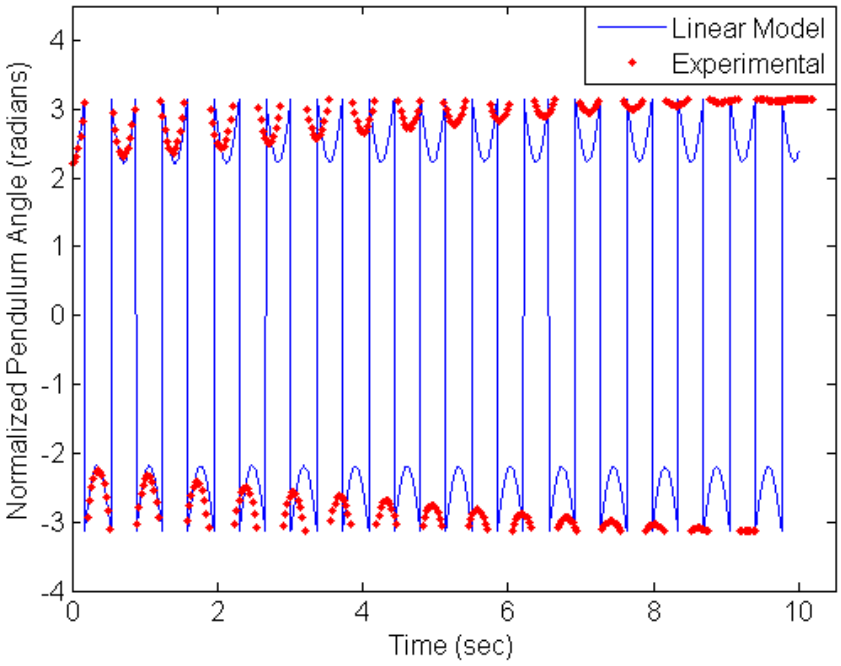

Figure 11: Comparison of Normalized Pendulum Angle From the Linearized Model and Experimental Data

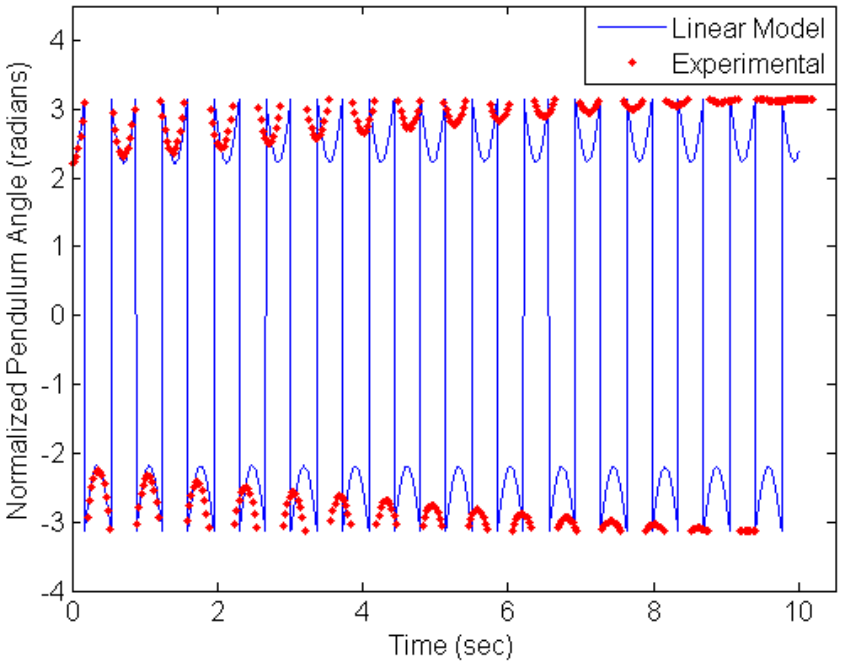

Figure 12: Comparison of Normalized Horizontal Arm Angle From the Linearized Model and Experimental Data

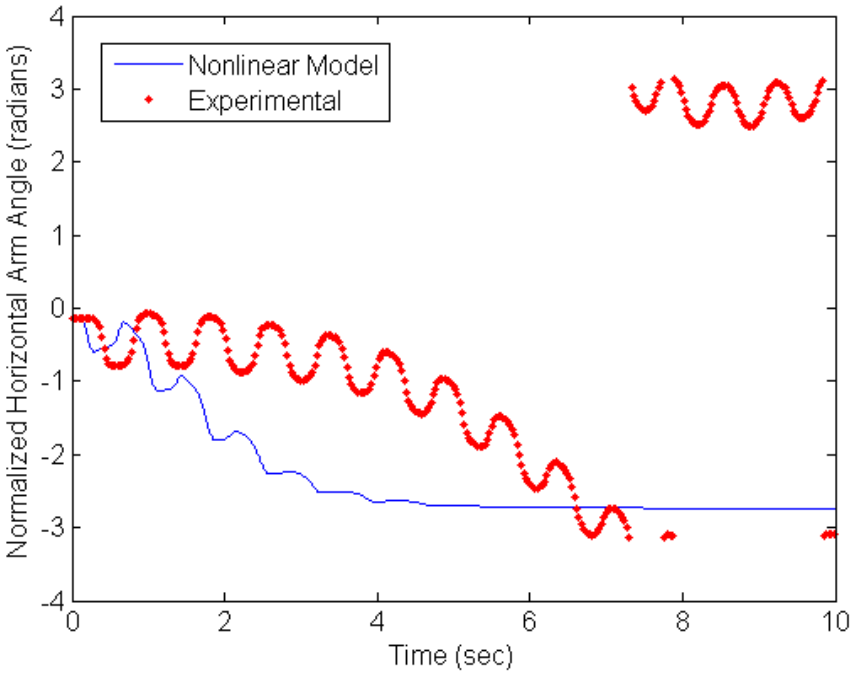

Figure 13: Comparison of Normalized Horizontal Arm Angle From the Nonlinear Model and Experimental Data with the Encoder Cable Removed

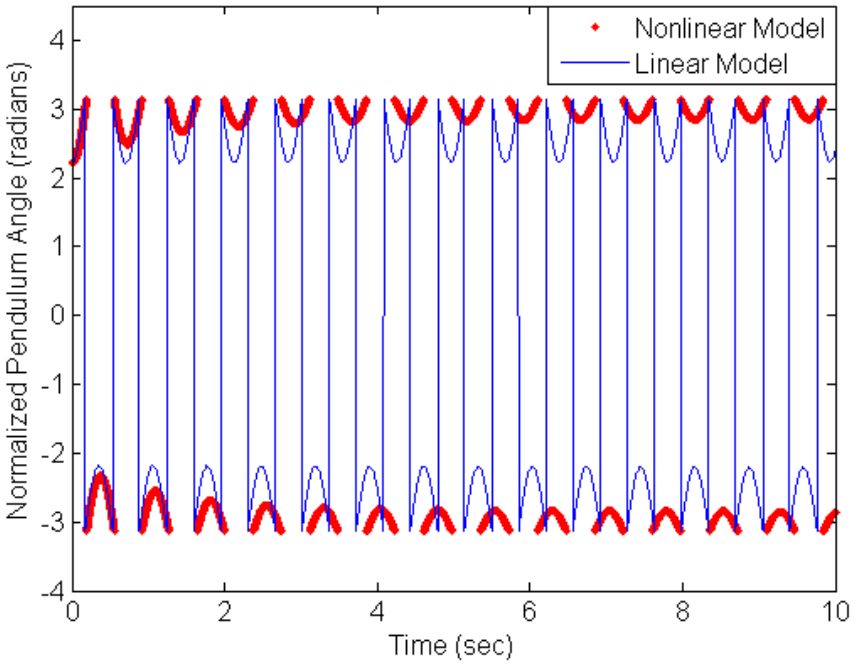

Figure 14: Comparison of Normalized Pendulum Angle From the Nonlinear Model and Linearized Model

4 State Space Controls in Simulink

Using the A, B, C, and D matrices derived earlier together with the tuned Q and R matrices below, we constructed a full-state feedback controller using the Matlab *lqr* function.

$$Q = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 40 & 0 & 0 \\ 0 & 0 & 120 & 0 \\ 0 & 0 & 0 & 60 \end{bmatrix} \qquad R = [6]$$

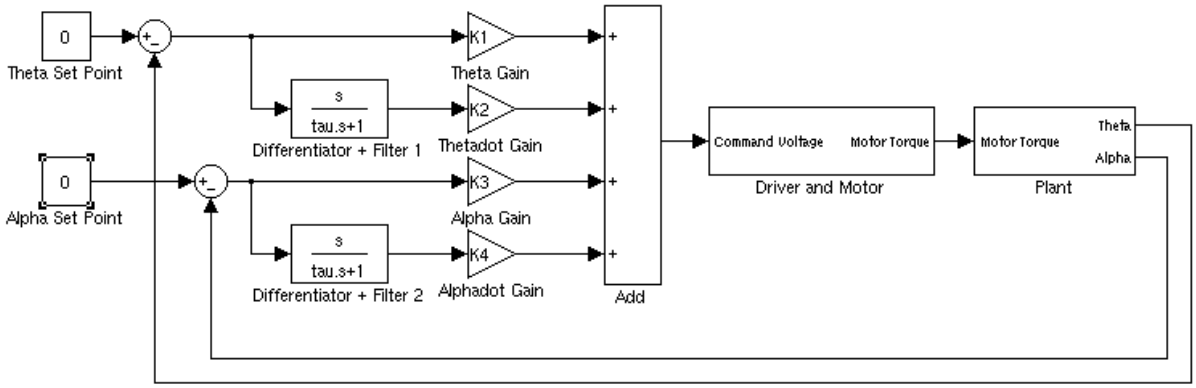

Figure 15: Block Diagram of State Space Balancing Controller

$$K_1 = -1.2910 \qquad K_2 = -2.9161 \qquad K_3 = -66.3961 \qquad K_4 = -8.4496$$

The full-state feedback controller using the gains above successful stabilized both of our linear and non-linear models. The effects of disturbing the system with an angle of 0.1 radians is shown in figures 16 and 17. Figure 16 compares the response of the linear system to the response of the non-linear model. The most obvious difference between both responses is the significantly larger oscillations present in the linear system although each system shows similar behavior. We note in the α response that the linear model overshoots the set point angle before returning to the set point angle over a long period of about 0.6 seconds. The non-linear model on the other hand shows very little overshoot at all and settles in under 0.2 seconds. However the controller has difficulty returning θ to the set point in the non-linear model taking approximately 10 seconds to return to 0.

Figure 17 introduces frictional forces to the non-linear model. Here we note that the system remains stable, but θ is unable to return to its set-point settling at 0.545 radians after about 10 seconds.

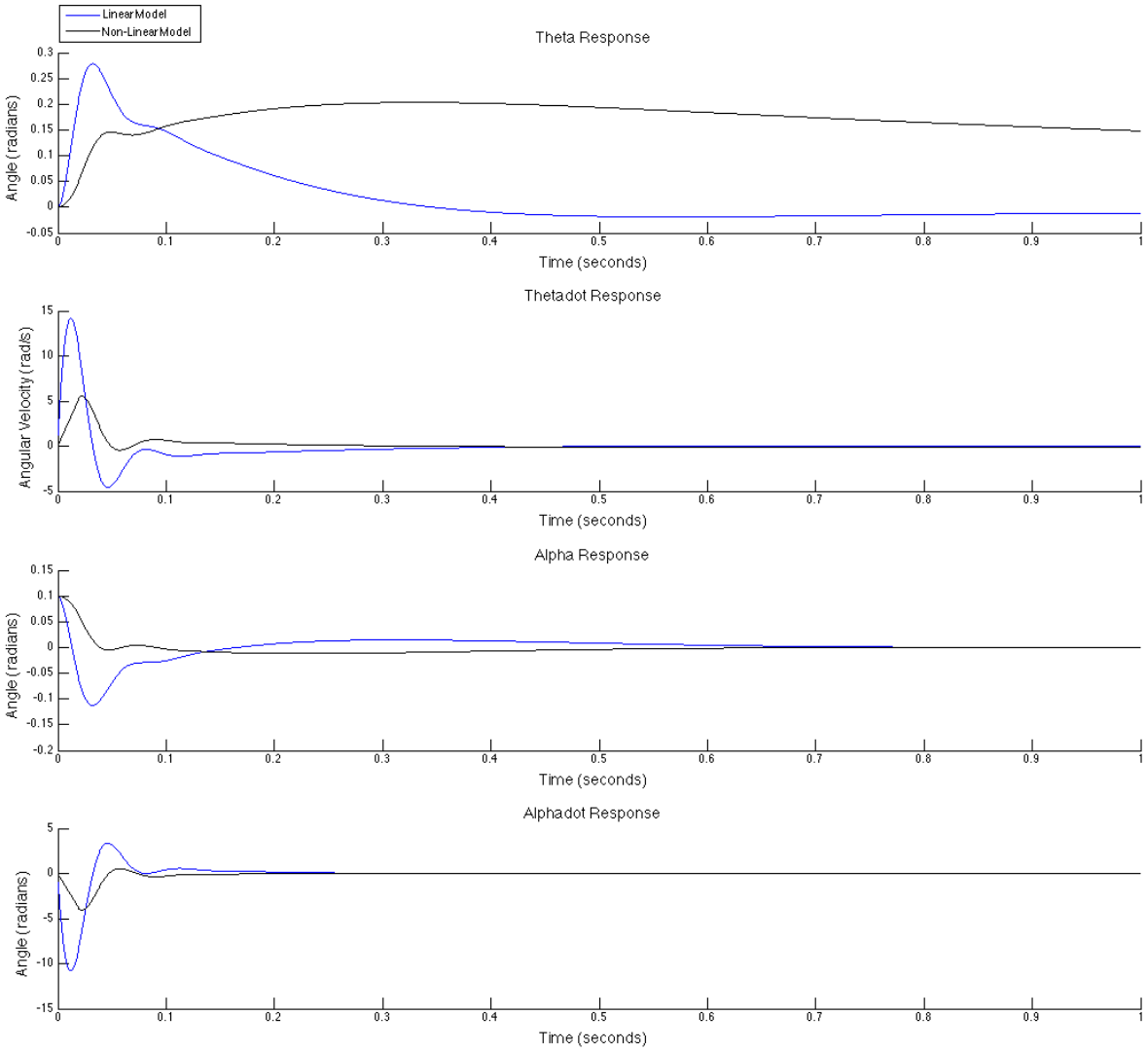

Figure 16: Comparison of Balancing controller response to perturbation, $\alpha_0 = 0.1$, for Linear and non-Linear models

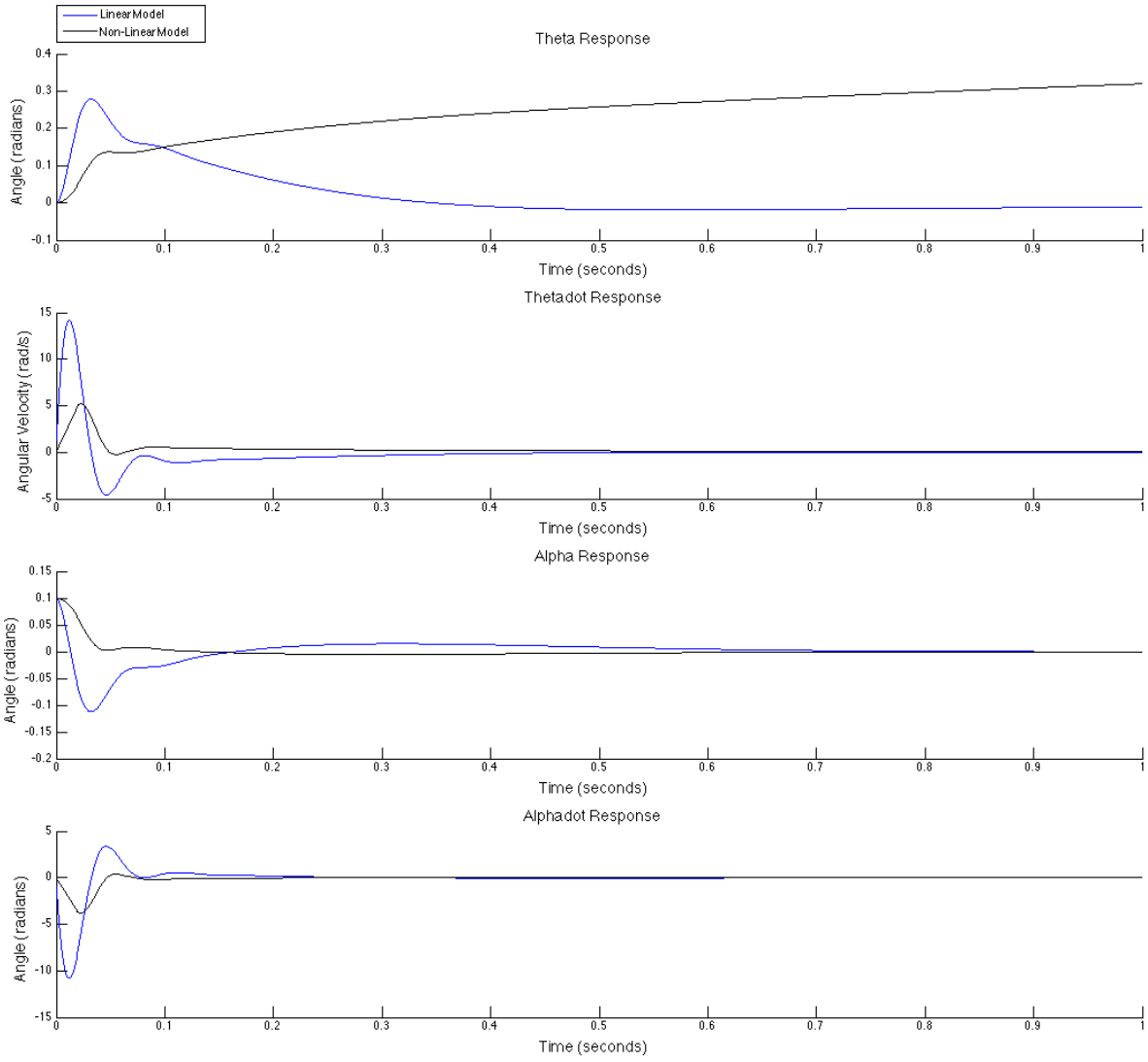

Figure 17: Comparison of Balancing controller response to perturbation, $\alpha_0 = 0.1$, for Linear and non-Linear models including effects of friction in the non-Linear Model

5 Swing Up Controller in Simulink

The objective of the swing-up controller is to provide the pendulum enough energy to allow it to reach the top position where its potential energy is maximized. Fig. 18 shows the overall scheme of the swing-up controller. This design follows the equation provided in the lecture slides: $V = K_a(E - E_{set})sgn(d\alpha/dt\cos(\alpha))$.

E_{set} represents the desired total energy and is set to 0, while E is the total energy of the system (consisting of the kinetic and potential energy of both motor and pendulum arms). Figure 19 shows the switching mechanism between the balancing controller and the swing-up controller. We use the angle α to determine when to switch from the swing up controller to the balancing controller described earlier. The swing-up controller shuts off at $0.45\,radians$ and the balancing controller activate at $0.35\,radians$. The dead-band of 0.1 rad, provides a buffer for the pendulum to decrease its speed so that the balancing controller could catch it more easily.

Both controllers have four main inputs: α, $\dot{\alpha}$, θ, $\dot{\theta}$, and the balancing controller has two additional inputs including the equilibrium points of α and θ. The lead filters used to estimate the derivatives of θ and α introduce some lag into $\dot{\theta}$ and $\dot{\alpha}$. Without compensation this causes the estimates of kinetic energy and potential energy to be out of phase with one another resulting in an incorrect energy total. To compensate, we applied the same lag to the θ and α channels.

Fig. 20 and 21 show the simulation results for two different initial conditions (3.14 rad, 0.5 rad) of α with the aggressive gain equal to 6. As we can see in both conditions, it takes less than 3 seconds to reach the steady state. The one with IC=3.14 required 4 swings to reach the balancing range, while the other one with IC=0.5 even achieved the goal in 1 second since it had more potential energy at the beginning.

We have added two additional features to our swing up controller beyond the original specification. The first is a mechanism to reasonably convert from the energy error produced by the difference of the total energy to a command. We can estimate the amount of energy that will be imparted to the system by applying a particular torque.

$$Work = \tau \cdot \delta\theta \approx \tau \cdot \dot{\theta} \cdot \Delta time$$

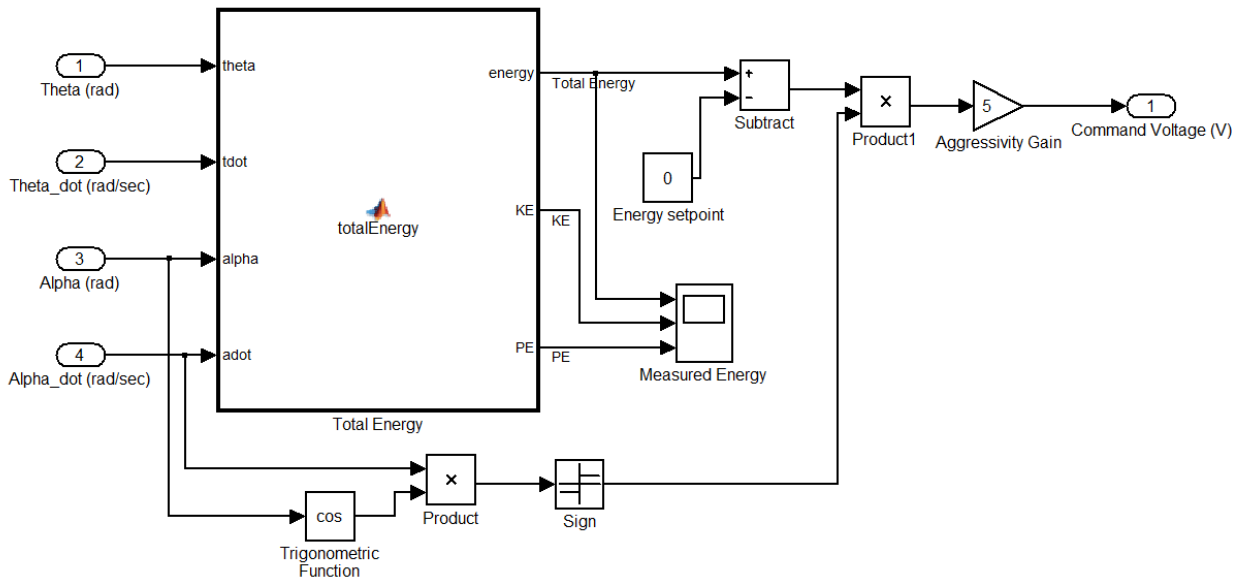

Figure 18: Swing-up and Balancing Controllers with a Switching Mechanism

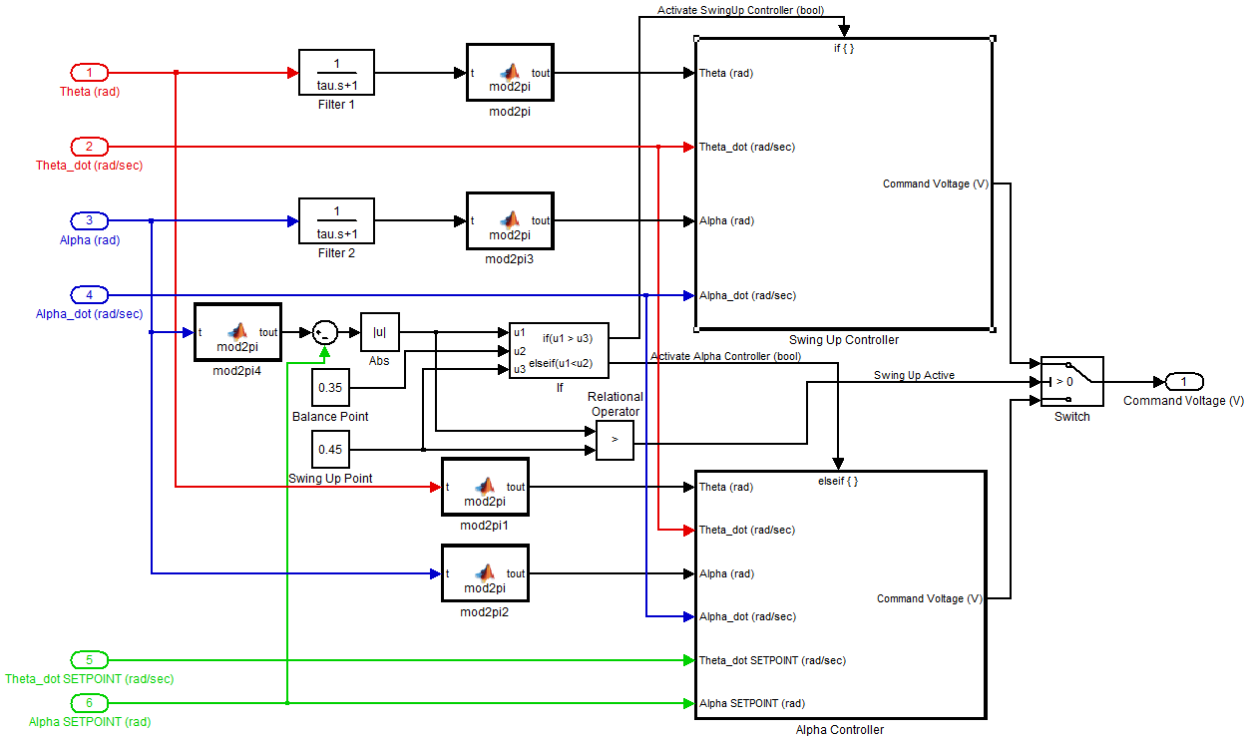

Figure 19: Swing-Up Controller

The other additional feature is an alternate system to reduce system energy when it exceeds the set point. The default swing up controller can get trapped in a cycle where it continues to add more energy because it has too much energy. For $|\alpha| > \frac{\pi}{2}$ and $\dot{\alpha} > 0$ we would normally swing increase $\dot{\alpha}$ however, if we have too much energy we will attempt to decrease $\dot{\alpha}$ by increasing $\dot{\theta}$. When $\dot{\alpha}$ is negative we will do the opposite and the system can become trapped in a loop where the horizontal arm is rotating rapidly. The net result of these additional features is that the system performed much better than the previous one. As shown in Fig. 23, the pendulum reaches the balancing steady-state with only two swings.

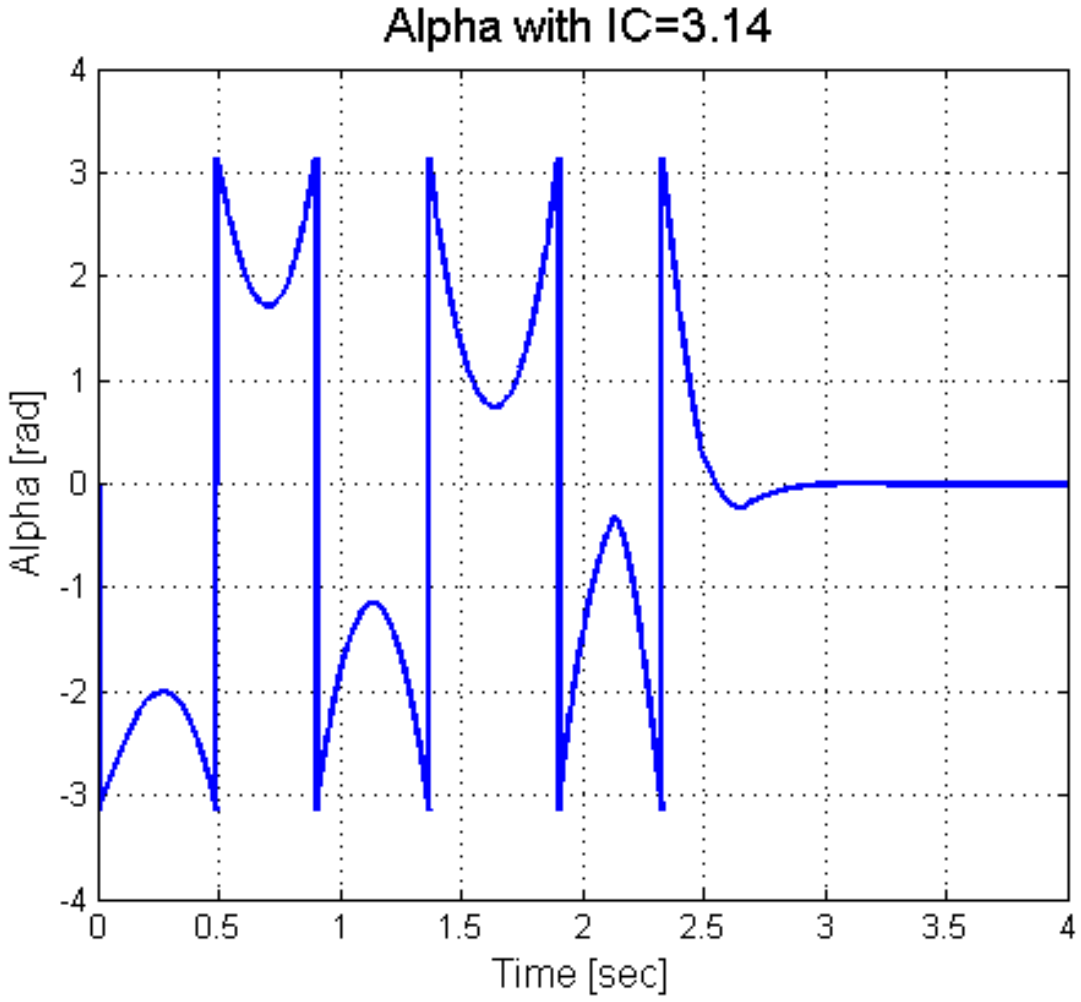

Figure 20: Behavior of α with IC=3.14 rad

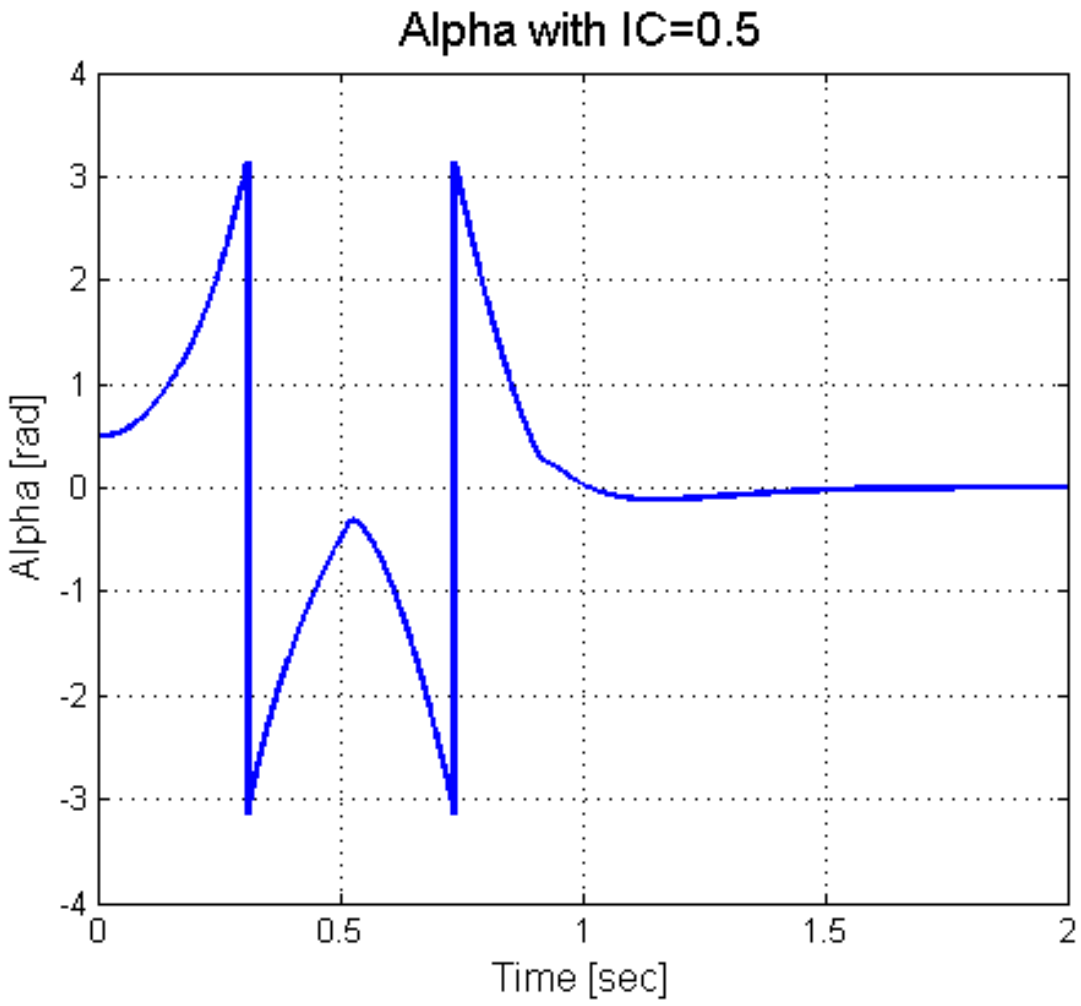

Figure 21: Behavior of α with IC=0.5 rad

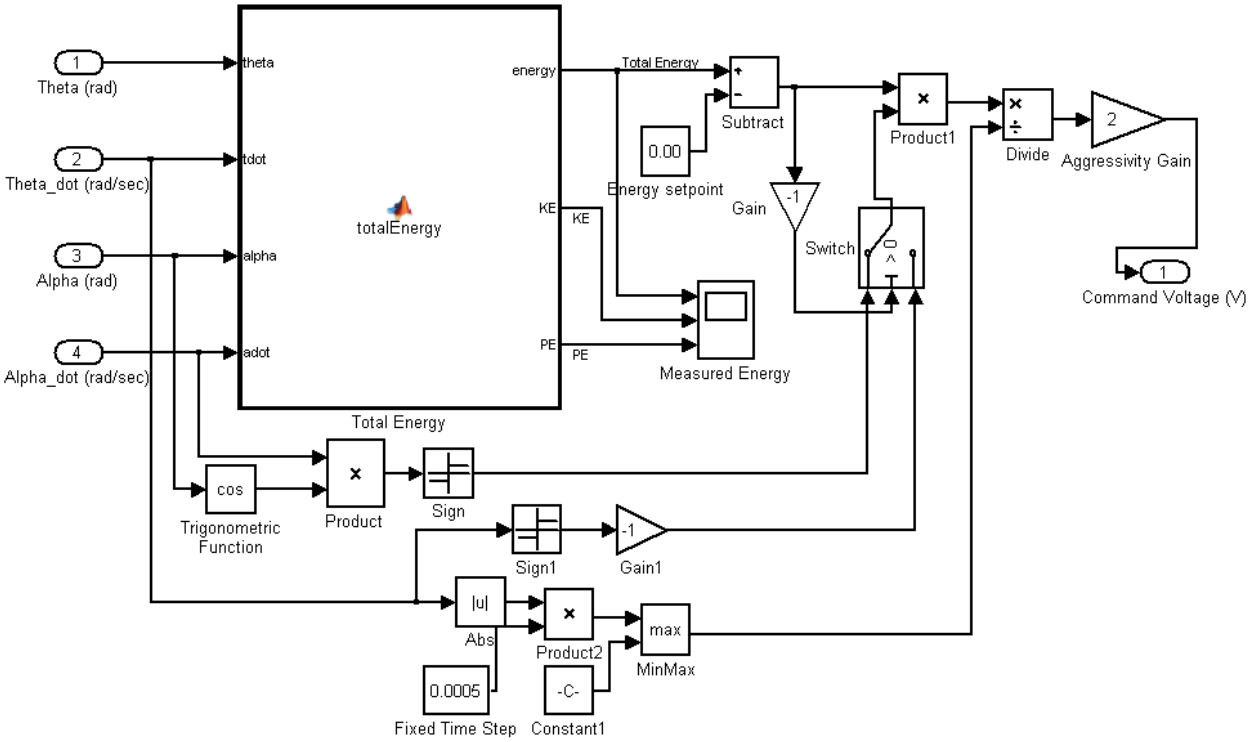

Figure 22: Swing-Up Controller with Energy-Limit Mechanism

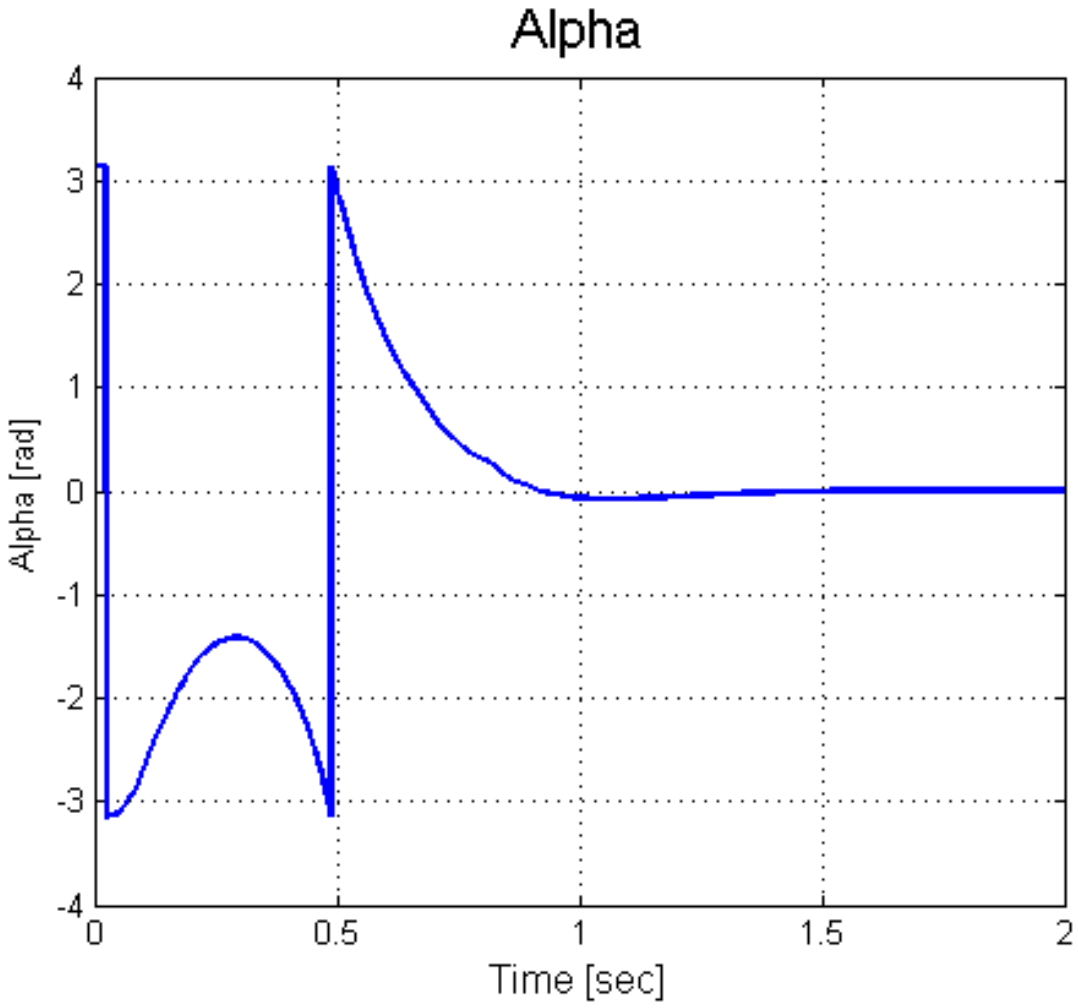

Figure 23: Behavior of α with IC=3.14 rad

6 LabView Implementation

a.

Angle normalization was implemented using the $mod2\pi()$ function. The function works by limiting an input angle, θ to $\pm\pi$ radians which effectively represents the entire angular spectrum of 2π radians. The advantage of using $mod2\pi()$ over a function that normalizes θ from 0 to 2π is that the angular reset point is at $\pm\pi$ as opposed to 0 radians. For our inverted pendulum we would like to ensure that α and θ are close to zero. Using $mod2\pi()$ ensures that spikes do not occur about the transition point.

The code for the $mod2\pi()$ operation is as follows:

```
if t<0
    t = -t;
    sign = -1;
else
    sign = 1;
tout = sign*(t - floor(t*0.5/pi + 0.5) * 2 *pi);
```

In the code above t represents θ. For our normalization, we require t to be positive and we keep track of the original sign of t through the variable *sign*. *tout* represents the output value of $mod2\pi(\theta)$. The $floor$ operation essentially finds the quotient of dividing t by 2π and the the $+0.5$ operation allows the angle to be normalized between $+\pi$ on one side and $-\pi$ on the other. This way, we can constrain θ to stay between $\pm\pi$.

$mod2\pi$ is very useful for controlling α. α will reach its unstable equilibrium every 2π radians and we are happy with it as long as the pendulum balances in the unstable equilibrium. Additionally the change from $+\pi$ to $-\pi$ occurs when the the pendulum is facing downwards, but at this point the swing-up controller would be operational and for the swing-up controller, the sign of $\cos(\alpha)$ in the $+\pi$ and $-\pi$ quadrants are the same, so this does not affect the system.

For θ control we have to be a little careful - the horizontal arm moves a lot during swing up and generally during balancing as well, if we apply a $mod2\pi()$ operation before calculating $\dot{\theta}$, the derivative will spike up violently during the $\pm\pi$ crossover point - this in turn would destabilize the inverted pendulum system. Hence it is a good idea to calculate $\dot{\theta}$ from θ_{Absolute}.

In general it is good practice to use a $mod2\pi()$ operation on repetitive angular functions, it helps in numerical precision in trigonometric functions and also captures the nature of the system - after a point, we get back to where we were.

While implementing the system in Labview we used the Matlab script block which allowed us to write Matlab code for execution as opposed to creating a messy wire-block diagram. The front panel and the block diagram can be seen in figure 24.

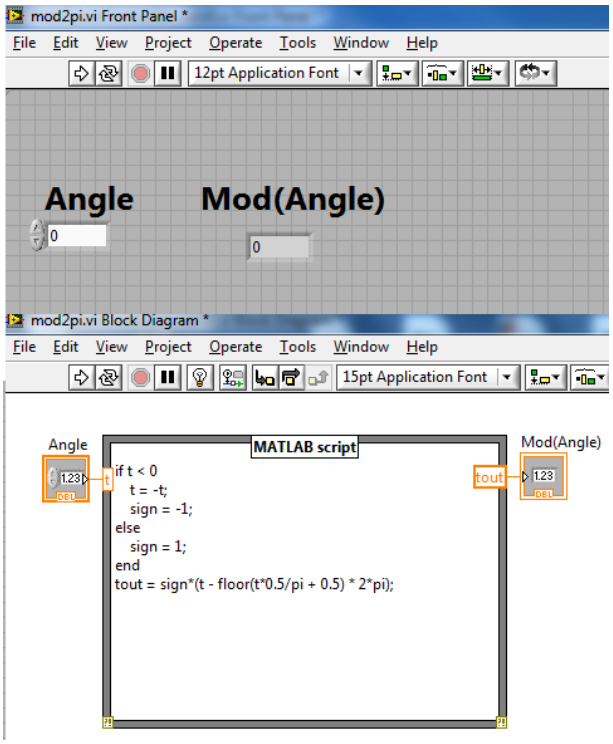

Figure 24: Front panel and block diagram of $mod2\pi$.

b.

Figures 25 through 32 show the various front panels and block diagrams of the various VIs and subVIs that make up the LabView controllers.

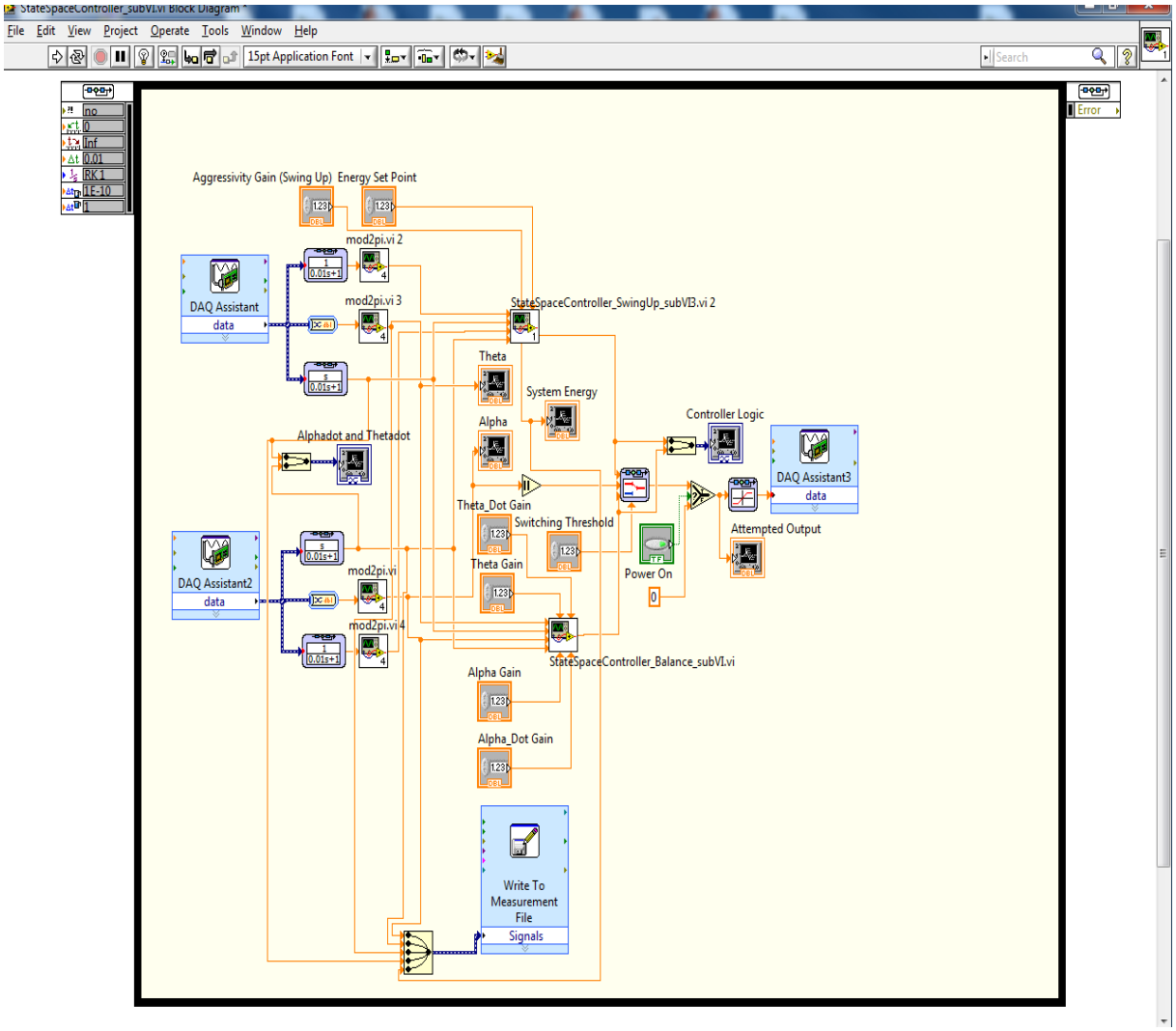

Figure 25: Block diagram of State-Space Controller VI.

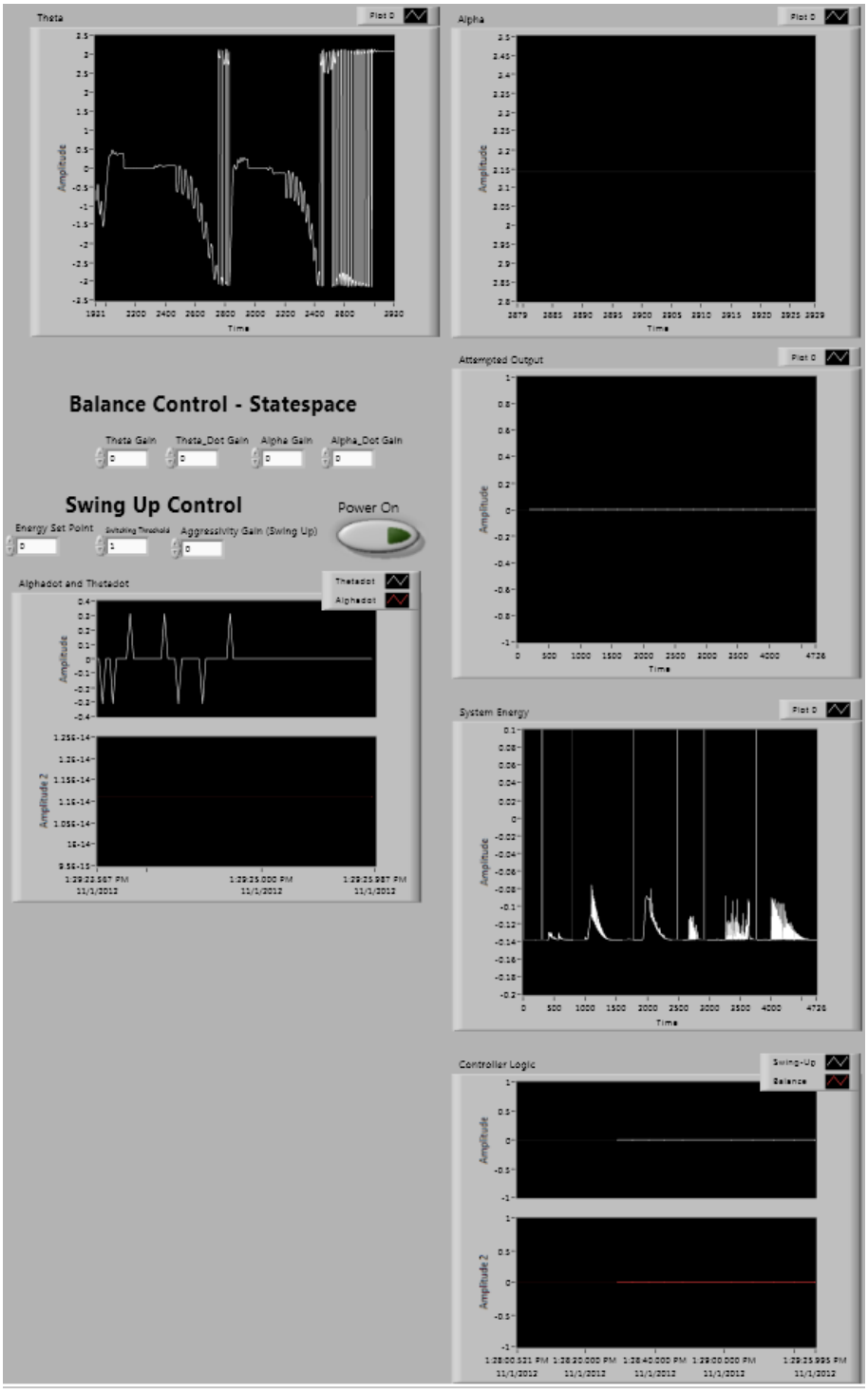

Figure 26: Front panel of State-Space Controller VI.

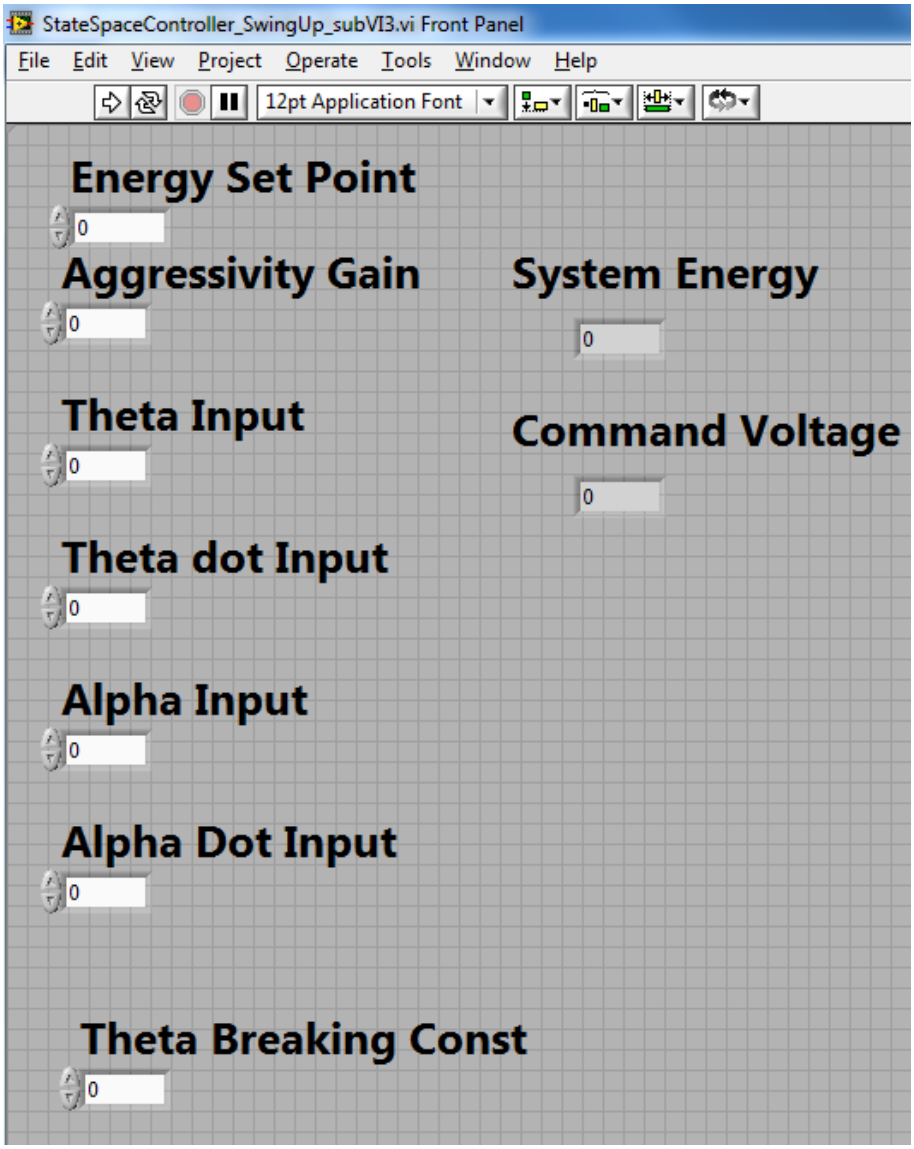

Figure 27: Front panel of the swing up controller. The *Theta Breaking Constant* is used to limit the excessive overshoot of the pendulum - this could happen during swing-up or even if someone adds excessive energy to the system by tapping the pendulum very aggressively. The theta breaking algorithm attempts to move the motor in such a way that it opposes movement in θ, this would dissipate excessive energy from the system.

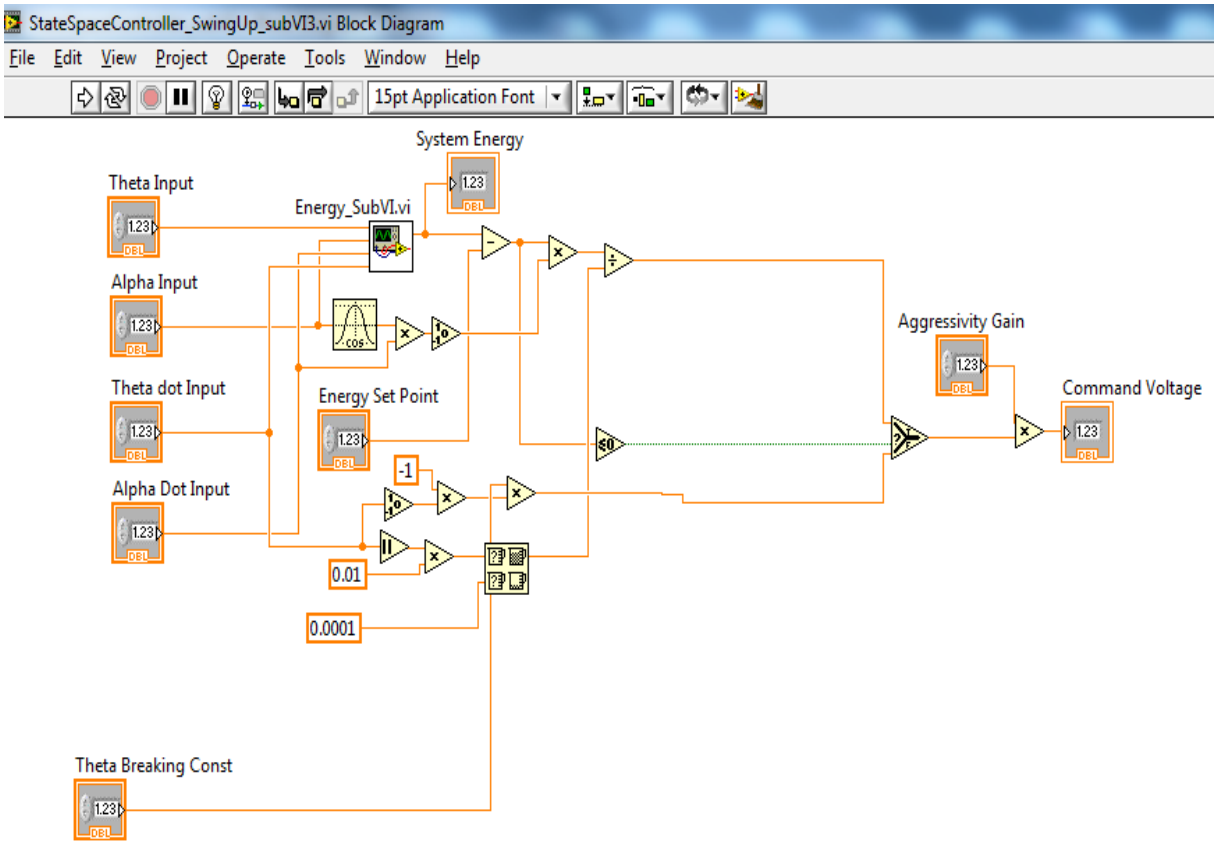

Figure 28: Block diagram of the swing up controller. Theta breaking is achieved by pulsing in the opposite direction of θ. The magnitude is manually tuned. If it is too high, it would inevitably add energy to the system, too small and it does not make a difference. The best hand tuned value was found to be 1.

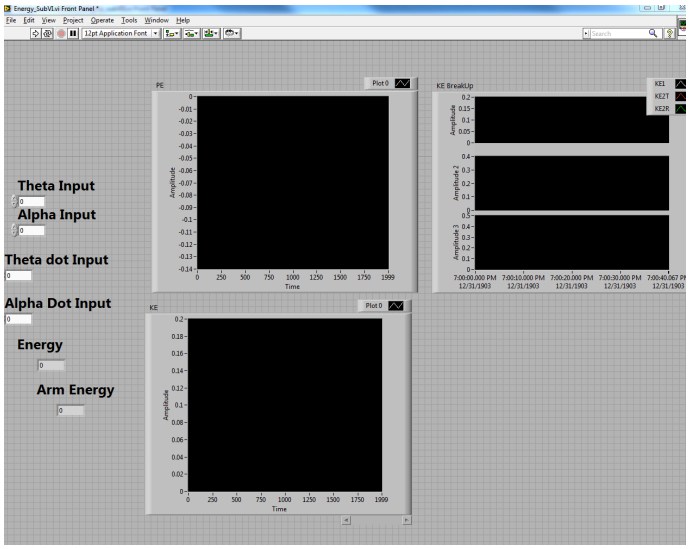

Figure 29: Front panel energy calculation subVI.

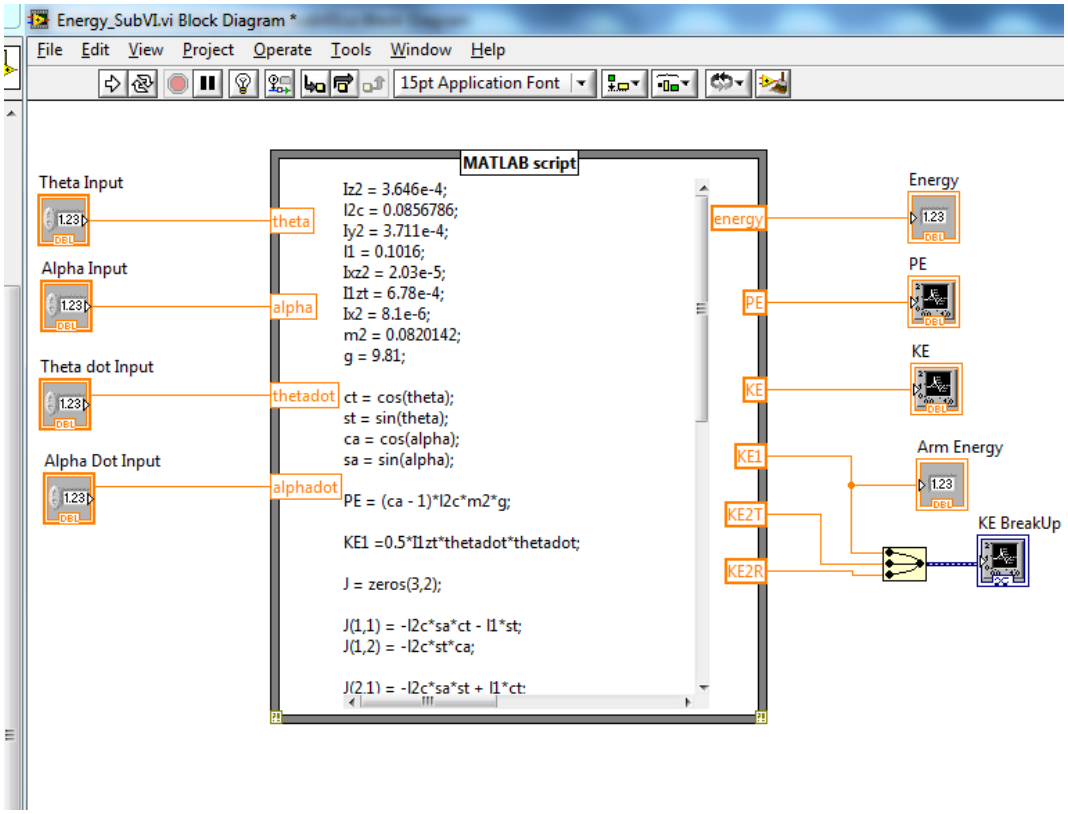

Figure 30: Block diagram of the energy calculation subVI. We used the Matlab script block again to aide in the writing the complicated equations instead of using block-based logic. The code is available in the Appendix.

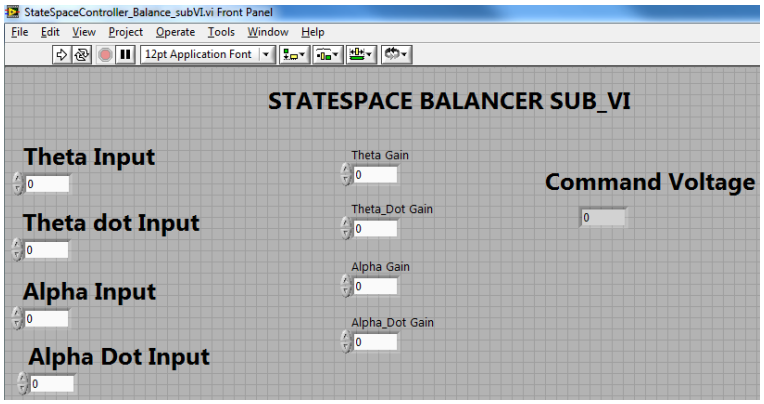

Figure 31: Front panel of the State Space Controller.

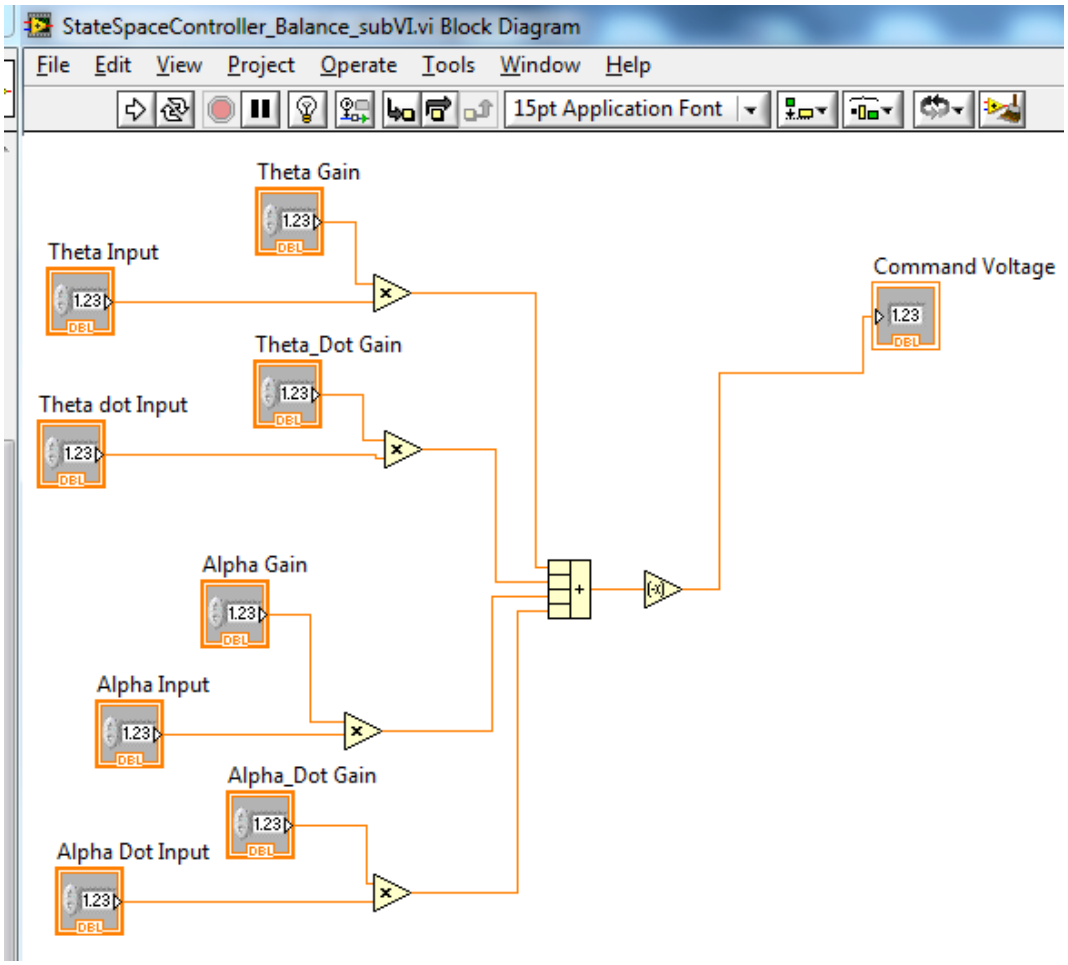

Figure 32: Block diagram of the State Space Controller.

c.

For the comparison of the actual system with the simulink model, we positioned the pendulum at $\alpha = 0.2$ radians so that the balance controller could grab the pendulum and balance it. A comparison between the predicted response and actual response can be seen in figure 33 and 34. 0.1 seconds of lag can be accounted for the delay we purposefully introduced in α and θ so that they are in phase with $\dot{\alpha}$ and $\dot{\theta}$. This minor introduction of lag helps stabilize the system better as the angle will not lead the angular velocity. Other delays and discrepancies can be attributed to unmodeled system properties, unaccounted motor inertia and external disturbances.

The trends for α (theoretical and observed) seem similar and the predicted model closely resembles the real system. However, for θ, we see a considerable gap in the prediction v.s. the actual behavior. We attribute these differences to the fact that we manually lifted the pendulum up to $\alpha = 0.2$ radians - which was an imperfect operation, which in itself changes the initial starting condition for θ. When combined with the fact that we are more interested in holding α at zero, the system will stabilize α much faster than θ.

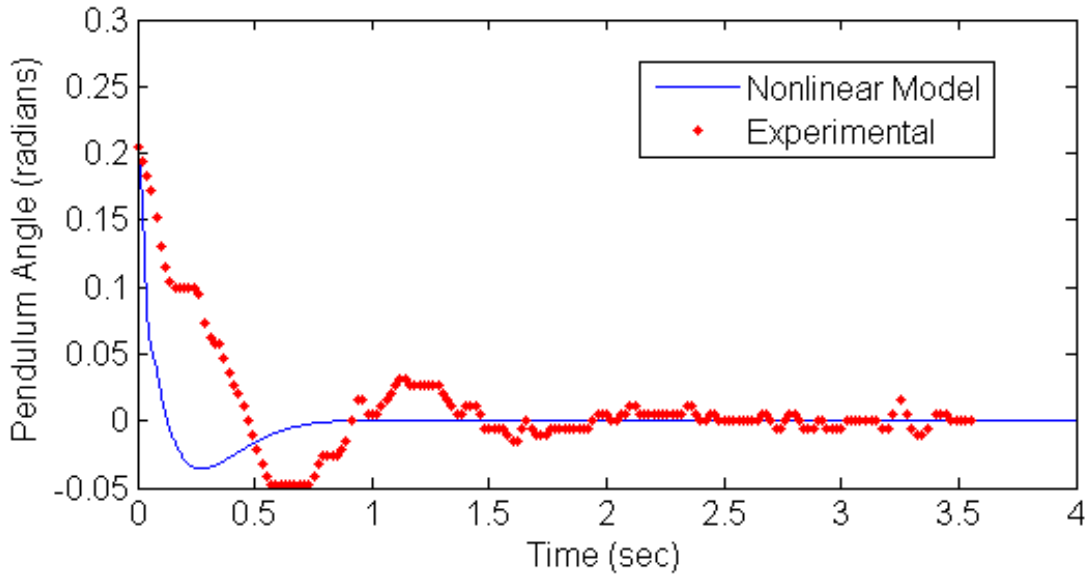

Figure 33: Alpha response during balancing - theoretical v.s. actual

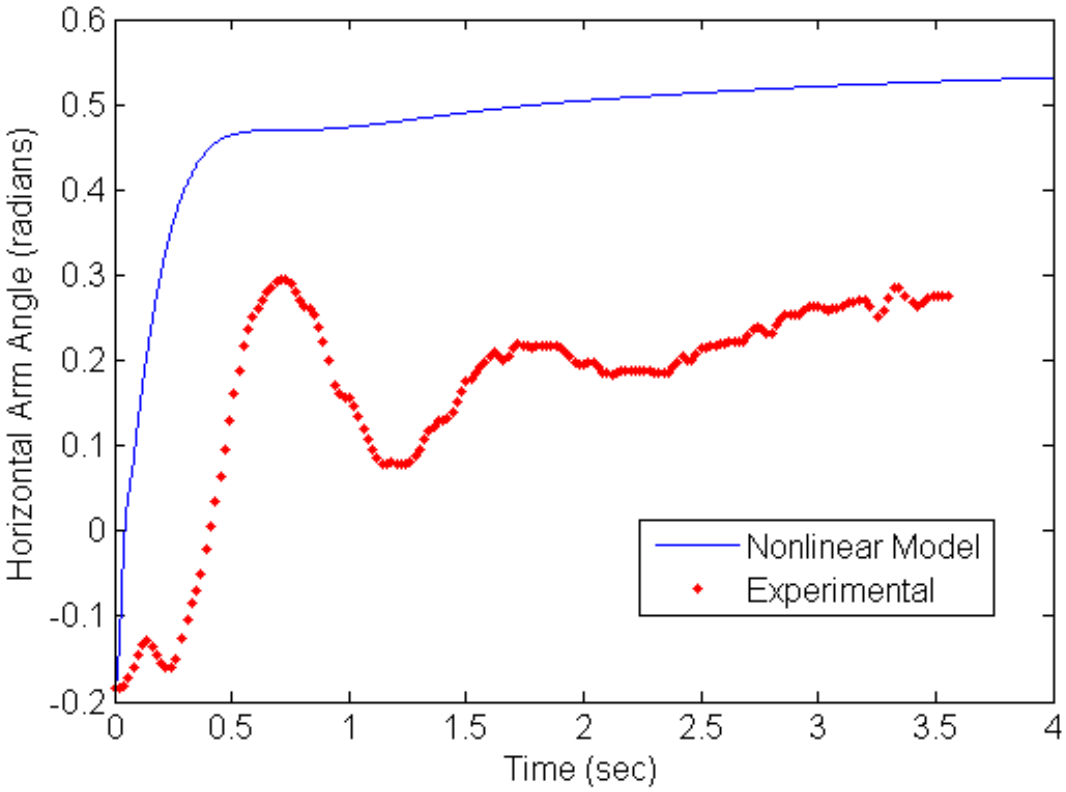

Figure 34: Theta response during balancing - theoretical v.s. actual

d.

Table 2 shows the results for testing the system under various conditions. Distances were measured using a vernier caliper between the outer edges of the objects and the closest part of the pendulum's body. For the counter-weight, this was between the weight and the keyless bushing and for the pendulum masses, it was between the weight mounted lowest on the shaft to the aluminum connector.

We noticed that the system performed robustly for major changes in the system. The only point where it could not swing up was when the masses were mounted *much* lower than usual. We attribute this to the fact that the system is unable to accurately estimate the energy of the system and to the fact that our motor was not strong enough to swing up at that point.

System Configuration	**Property**	**Value / Position**	**Observations**	**Comments**
Standard	Counter Weight position	67.45 mm	Normal	-
	Pendulum Mass	2 weights at 135.41mm	Normal	-
Modifying Weights	No Counter Weight	-	Swings up and balances	Shudders more, this could be due to less inertia.
	No pendulum mass	-	Swings up and balances	Had to increase aggressivity, balancing is slightly shaky
	No pendulum mass or Counter Weight	-	Swings up and balances	Had to increase aggressivity, balancing is VERY shaky
Modifying the Pendulum Masses	1 mass	End of pendulum	Swings up and balances	No noticeable change
	2 masses	32.52 mm	Able to balance, unable to swing up	Energy calculations could be off: system thinks it has more Energy than it does. Also our motor is weak
Modifying the Counter Weight		10.16 mm	Swings up and balances	Slow swing up due to higher inertia on motor

Table 2: Emperical results of various configurations to test system robustness

7 Effects of LabView Loop Rate on Stability

We ran the system at a loop rate of 0.01 seconds (100 Hz) and tuned our filter to reject noise at this rate. We had satisfactory system robustness and additionally, we were able to use stronger low pass filters to reject more noise.

In order to test the limits of stability, we changed the loop rate logarithmically. As we can see from table 3, there is a lower bound and an upper bound for the limits of stability. We can attribute the lower bound to the fact that our filters were not designed to operate at such high frequencies. The upper bound stability loss is attributed to the inability of the system to respond quickly enough to balance the system or capitalize on the swinging the pendulum in the opposite direction to lift it up.

Loop Rate(sec)	**Comment**
0.003	*Balancing doesn't work*
0.004	*Swing Up doesn't work*
0.005	Both work
0.006	Both work
0.007	Both work
0.008	Both work
0.009	Both work
0.01	Both work
0.02	*Both Fail*

Table 3: Emperical results of various configurations to test system robustness

8 Classical Controller

In addition to State Space Controls as detailed earlier in this report we can also attempt to apply our classical controls knowledge to design a controller in the Laplace domain. The transfer functions of the system, $G_1(s)$ and $G_2(s)$ between motor torque, $\tau_m(s)$, $\theta(s)$ and $\alpha(s)$ respectively computed earlier are repeated below. As mentioned earlier, it is necessary to consider the effects of damping as they serve to make the system less, rather than more stable.

$$G_1(s) = \frac{\theta(s)}{\tau_m(s)} = \frac{s^2C_1 + sb_\alpha - C_4}{(s^2C_1 + sb_\alpha - C_4)(s^2C_5 + sb_\theta) - s^4C_3^2} \tag{9}$$

$$G_2(s) = \frac{\alpha(s)}{\tau_m(s)} = \frac{-s^2C_3}{(s^2C_1 + sb_\alpha - C_4)(s^2C_5 + sb_\theta) - s^4C_3^2} \tag{10}$$

Because we have two coupled output variables that we wish to control, $\theta(s)$ and $\alpha(s)$ and only a single input to the system, we have constructed a non-traditional loop within a loop controller. Rather than tightly control $\theta(t)$ and then use that control to control $\alpha(t)$, we instead tightly control $\alpha(t)$ and then control $\theta(t)$ in the outer loop as shown in 35. Note the compensator block shown in figure 36. The following transfer functions are in terms of motor torque, however our controller does not output motor torque but rather a voltage which commands a current in the motor. If we assume a linear model of the current driver and the electrical properties of the motor, than the appropriate conversion factor is:

$$K_{compensator} = \frac{1}{K_a K_t} \qquad K_a = 1 \qquad K_t = 0.0314499$$

Substituting constants, we can get the open loop poles and zeros for the alpha transfer function, shown in table 4. Because of the zero at the origin and the pole at the right half plane, all normal controllers such

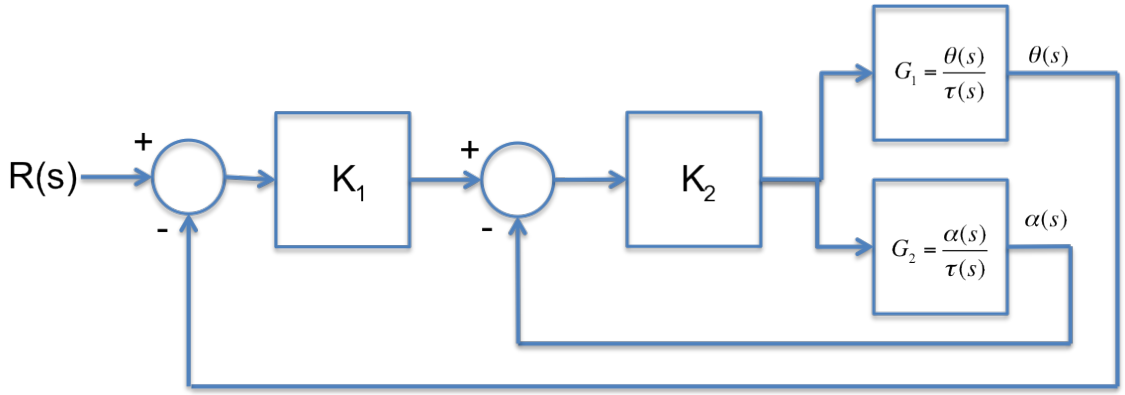

Figure 35: Block Diagram of Classical Balancing Controller

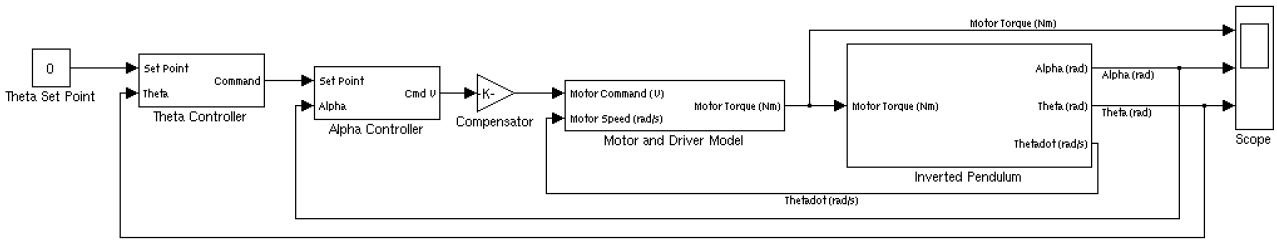

Figure 36: Loop within a loop to control both θ and α in Simulink Model

as lead, lead lag, PID, will all fail to bring that branch of the root locus out of the right half plane. Our controller shown below in (11) uses a pole at the origin to cancel the system zero and a pair of zeros to pull the root locus out of the right half plane. We added a filtering pole to try and make the system more robust to noise. The resulting system root locus is shown in figure 38 The proportional gain term was chosen to make the dominant pair of poles critically damped.

$$K_2 = -3.7203\frac{(s+3.75)(s+7.85)}{s(s+50)} \tag{11}$$

With the inner loop in place we now wish to control $\theta(s)$ which with just the inner loop is unstable and diverges. First we must compute a new effective transfer function between the set point of the inner loop, $R_\alpha(s)$ and $\theta(s)$

$$\frac{\theta(s)}{R_\alpha(s)} = \frac{K_2(s)}{1+K_2(s)G_2(s)}G_1(s) \tag{12}$$

Numerically this transfer function is given by (13) with its 0 degree root locus shown in figure 39.

Zeros	0	0		
Poles	0	-10.3814	10.3623	-0.0177

Table 4: Poles and Zeros of equation (10)

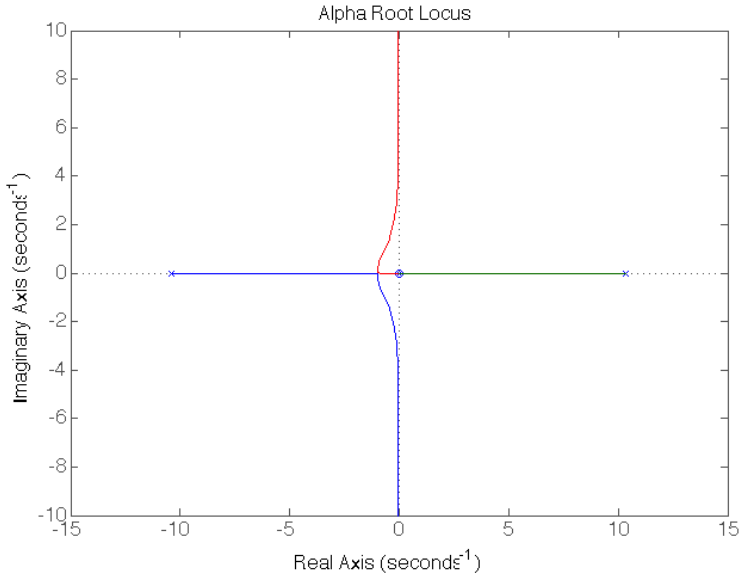

Figure 37: 180 Degree Root Locus of Transfer Function Relating α to input torque with a negative K

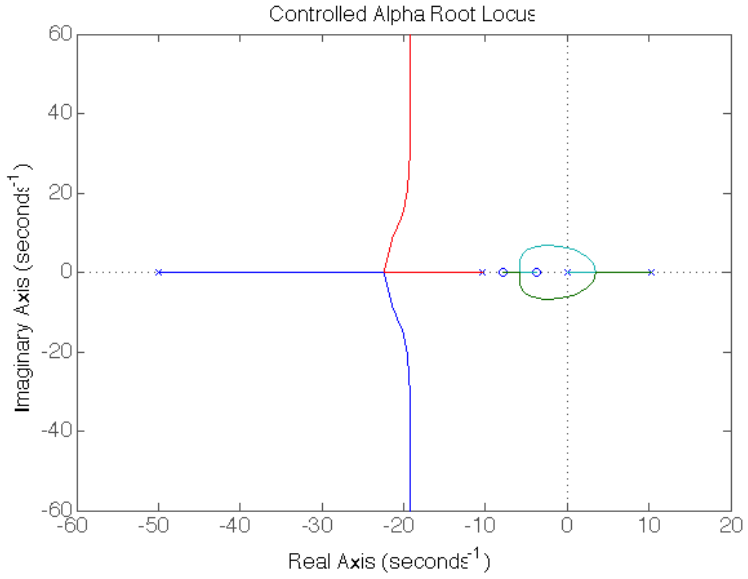

Figure 38: Root Locus for Alpha with Classical Controller

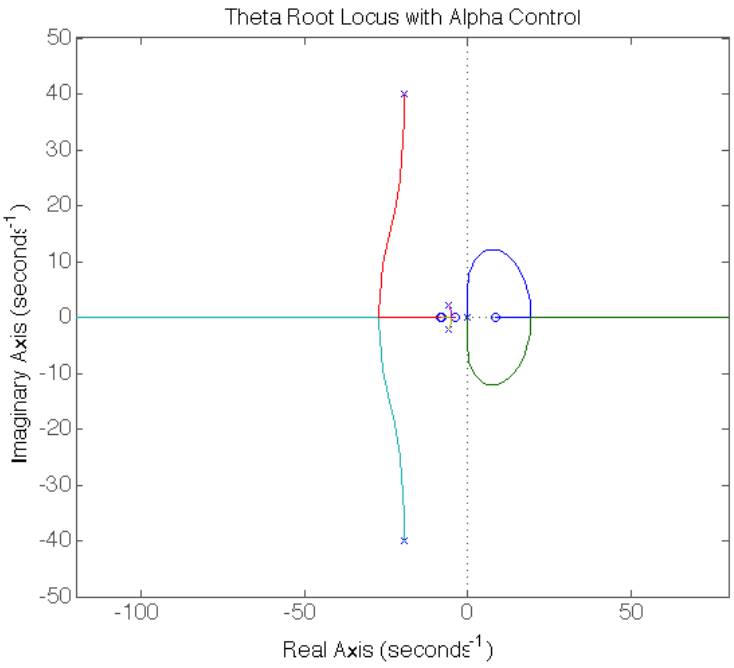

Figure 39: Transfer Function for Theta with Alpha Controlled

$$\frac{\theta(s)}{R_\alpha(s)} = \frac{-0.003596\,s^4 - 0.04174\,s^3 + 0.1503\,s^2 + 2.974\,s + 7.549}{(1.078\times10^{-6})s^6 + (5.394\times10^{-5})s^5 + 0.002641\,s^4 + 0.02615\,s^3 + 0.08098\,s^2} \tag{13}$$

Normally we would like to design a controller that implements negative feedback, however in this case a positive feedback is appropriate and allows us to use a simple Lead Controller shown in (14) to obtain a stable root locus.

$$K_1(s) = 0.01784\frac{s+1}{s+15} \tag{14}$$

The resulting root locus for the controlled system is shown in figures 40 and 41. It is worth noting that we are assured that alpha will remain stable regardless of what the outer loop does to the set point because all of its poles are in the left half plane, however this may not always be true due to the effects of saturation which could reduce our open loop gain and shift the poles into the right half plane.

With the classical controller constructed we first test it against the linear system model to confirm the design and it does prove to be stable, shown in 42. However we do notice that $\theta(t)$ has a very long settling time.

We next examine how using the non-linear model, which includes the effects of friction, impacts the performance of the controller. Figure 44 shows the limited effects of friction on the transient response. There is actually a slight improvement with decreased overshoot. However, when looking at the response over longer time scales, shown in figure 45, we see that the system exhibits a limit cycle likely caused by the friction.

Still in simulation, we also wish to compare the performance of the classical controller with the state space controller on the non-linear model. We note as mentioned earlier, the state space controller exhibits a large steady state error for θ while the classical controller exhibits a large amplitude limit cycle. Considering just the transient response we see that the state space controller responds slightly faster than the classical controller. Simulation also claims that the classical controller is able to stabilize the pendulum starting from and initial angle, α_0 up to $0.6125 radians$, while the state space controller has slightly worse performance

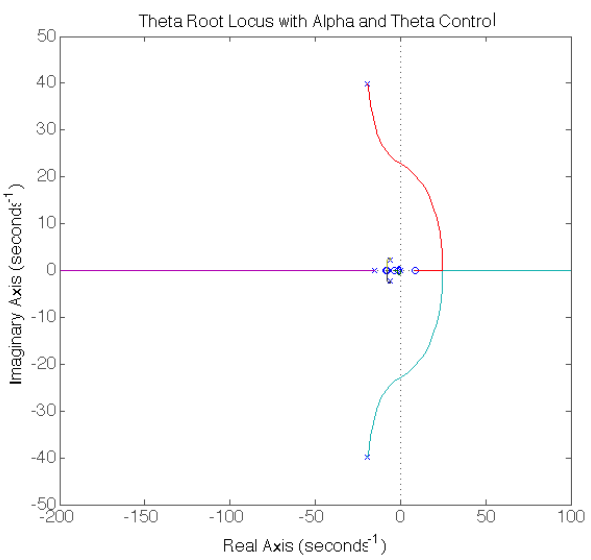

Figure 40: Root Locus of Controlled θ

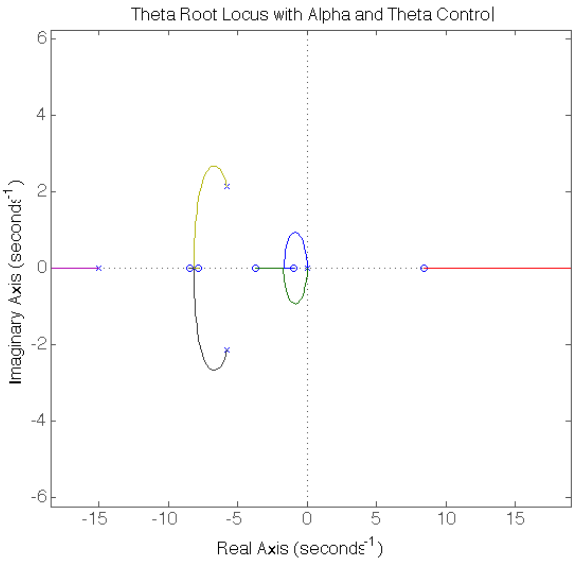

Figure 41: Detail of Root Locus to Left

being able stabilize initial α values up to $0.573 radians$. However we do not expect to see performance this good in reality because the real controller is running at much lower loop rates than these simulated controllers which likely helps the simulations' stability. Furthermore, there is no noise to corrupt our measurements and the models do not have to deal with un-modeled forces, specifically the cable which might limit our performance.

Finally we implemented our classical controller in reality. However the controller required some additional tuning of gains beyond the original design to successfully balance the pendulum, but this did result in a stable system.

$$K_{1\,adjusted} = 0.2676\frac{s+1}{s+15} \qquad K_{compensator\,adjusted} = 50$$

Plugging these modified values back into our classical controller in Simulink resulted in a stable system and its response is plotted along side the response of the physical system in figure 46. We see a similarity between the θ plots. Both experience limit cycles and our estimate of the amplitude and frequencies of this cycle do not seem to be very far off. On the α plot we see a large difference in the initial transient response, with the real system oscillating rapidly before settling, while the simulated system overshoots once and quickly settles into its limit cycle. All of our experiments so far have failed to produce smooth motion from the pendulum and this case is no different. However we again note a similarity between the limit cycles in the two graphs. We see that observed alpha shows very small peaks that correspond to $\dot{\theta}$ changing signs as our simulated system also shows.

Full state feedback control seems to be an easier system to design, simply construct a linear model of the system and run a function and we are given a set of values that work pretty well on the actual system. The main disadvantage of this approach is that we lose all of the Laplace domain intuition learned from classical controls. However, in designing this classical controller we had already lost a great deal of that intuition and were left with playing with the root locus until a stable system appeared. Despite these upsides, full state feedback has one major disadvantage compared to the classical controller. All of the states must be observable for the method to work. In this case we are forced to approximate the derivative using a lead controller which undoubtedly has negative consequences on performance and in other examples we may have no way of measuring the internal state of the system. In this case classical controls methods are still applicable and may be able to produce a stable system.

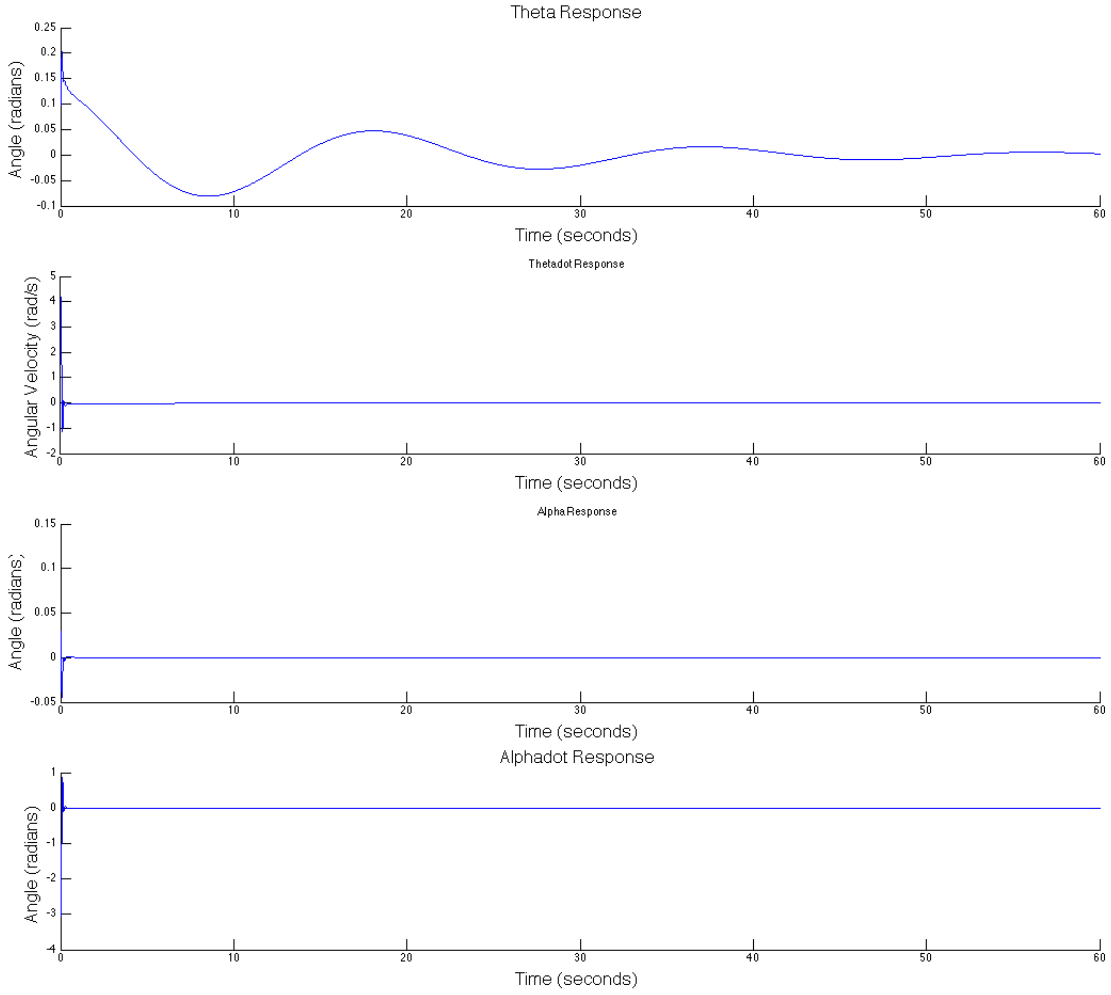

Figure 42: The Classical Controller successfully stabilizes the linearized system, however θ has a long settling time

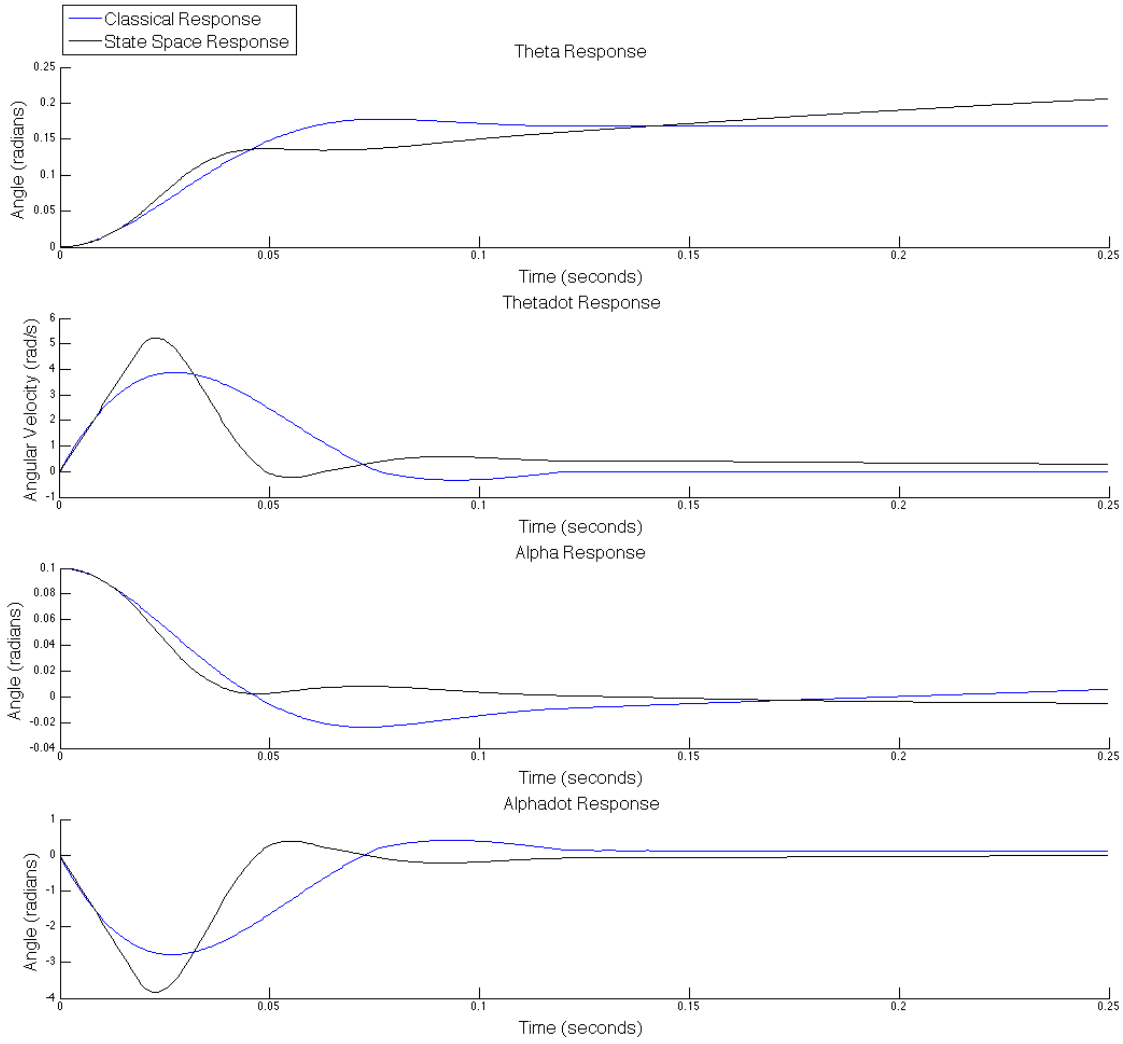

Figure 43: Comparison of the transient response of the non-linear model when controlled by the state-space controller and the classical controller.

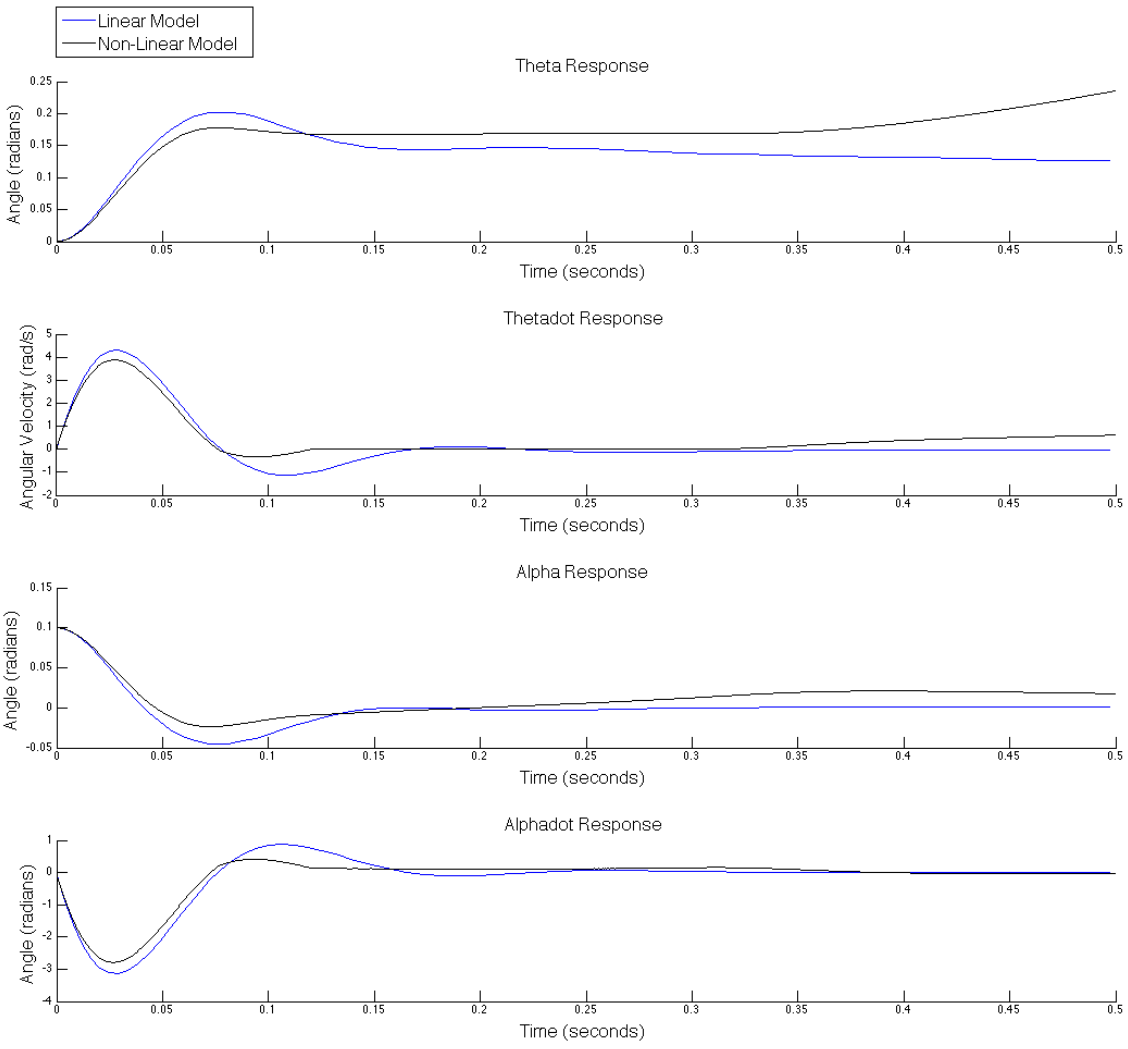

Figure 44: The difference in the transient response of the classical controller applied to the linear and non-linear models

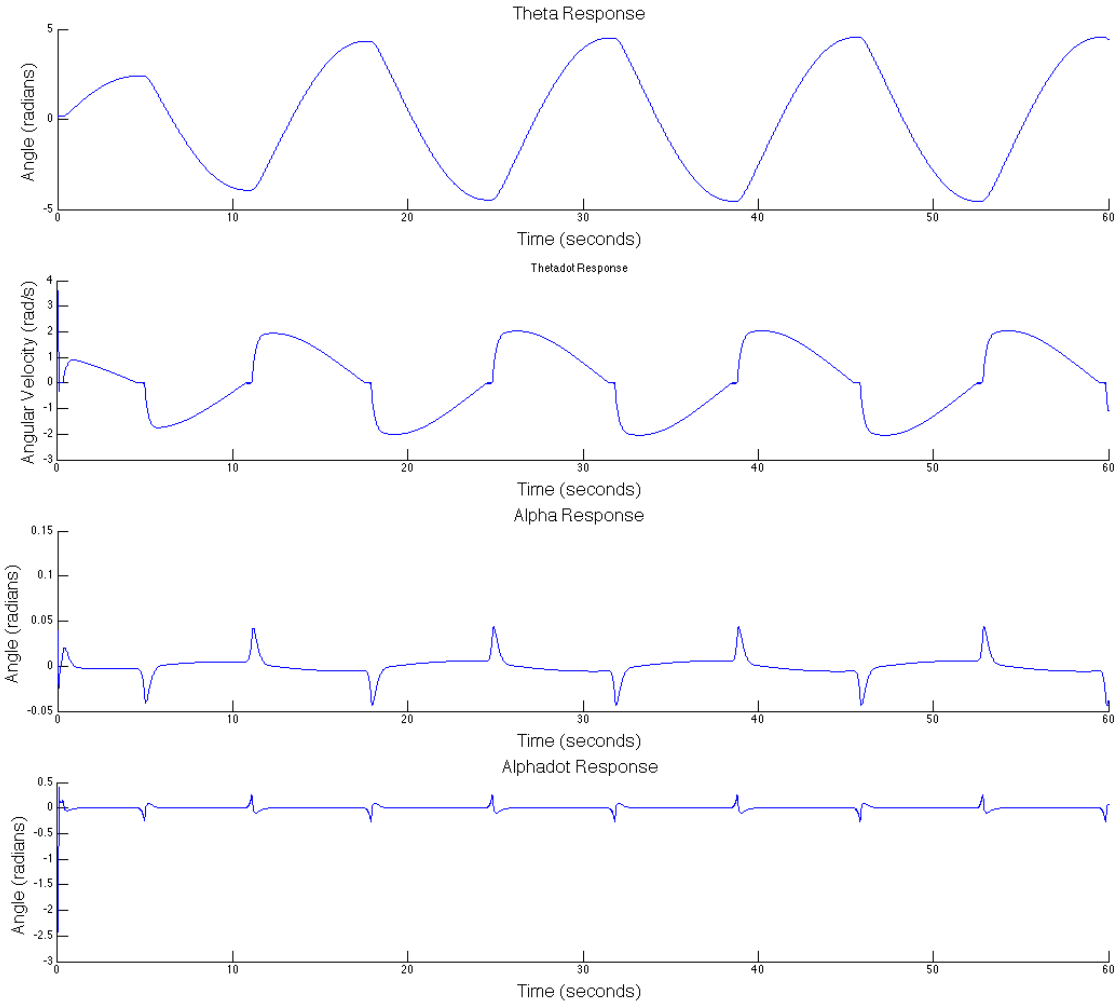

Figure 45: Limit cycle exerted by the classical controller when applied to the non-linear model for an initial α of 0.1 radians

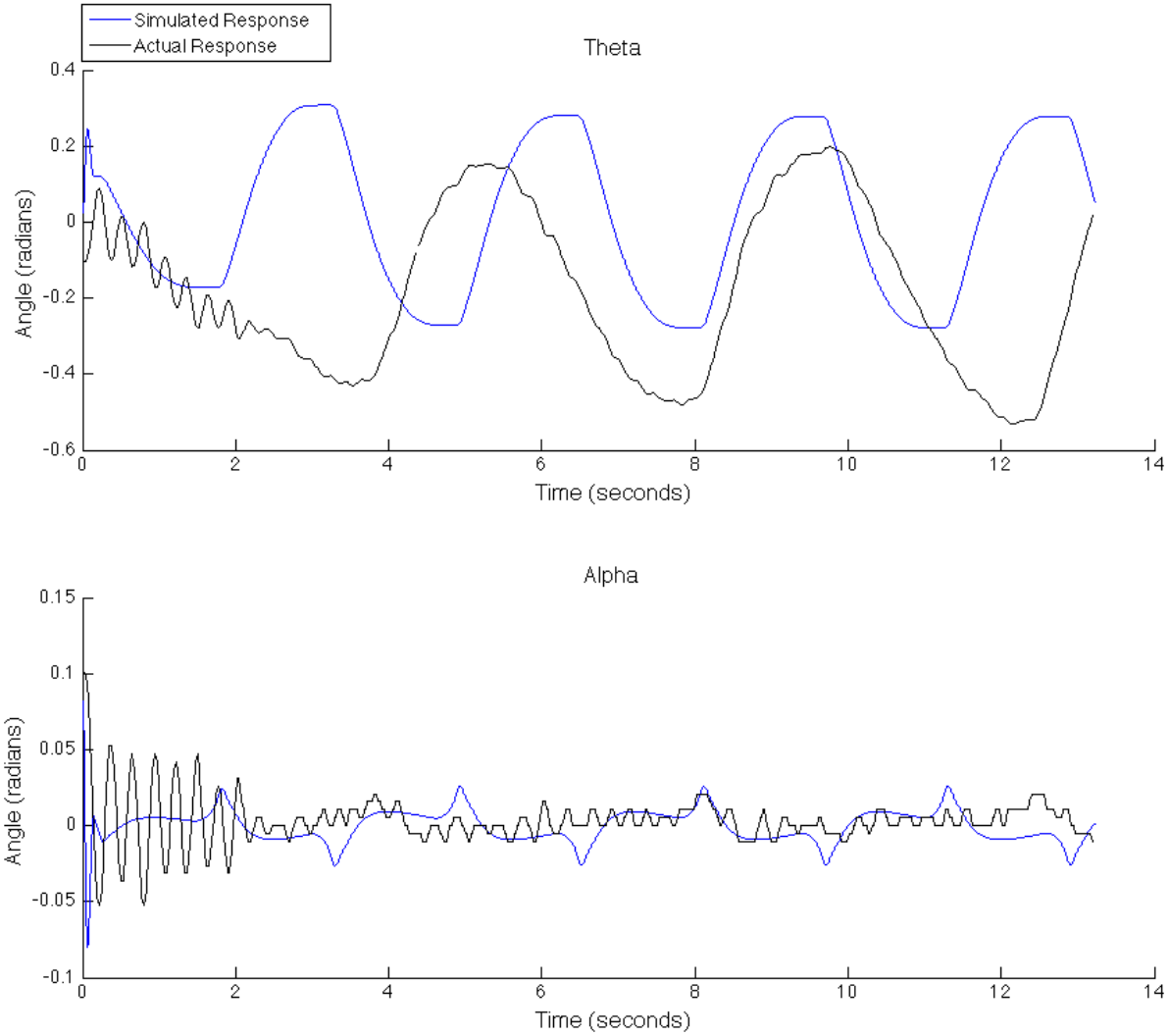

Figure 46: Comparison of simulated non-linear system with modified gains against the real response

9 Another Swing Up Controller

When designing a new swing-up controller our goal was to use a very simple method that was still reliable. The controller we implemented shared some similarities with the method discussed in lecture, but simplified significantly. Rather than calculating the energy of the system or measuring the pendulum's position, our controller only looks at the direction of the pendulum and applies a constant voltage such that it switches sign every time the pendulum switches direction. Our goal was for the motor to apply high torque in one direction until the pendulum had stopped moving upwards, and then apply high torque in the opposite direction. With proper tuning of the voltage used, we believed we could meet the goal of only two swings before balancing. However, in practice we were not able to achieve this as our motor couldn't accelerate quickly enough to raise the pendulum sufficiently before its cords began twisting up too much. Therefore, we had to settle for a relatively low voltage command which required more swings.

Figure 47 shows the simplified balance controller implemented on the real system. We used a controller switch-over point of 0.3 radians (switching to our state-space balancer) and a command voltage of +- 3.0V to get into balance mode in 19 swings (approximately 9.5 seconds). This was much worse than our prediction, which is shown in figure 48. With our nonlinear simulink model we were able to reach balance mode in 4 swings (approximately 2.3 seconds) with a switch-over setting of 0.6 radians and a command voltage of +- 2.2V. Our controller on the real system requires a switch-over point much closer to the vertical position than the simulation does, which accounts for some of the difference in performance. Similarly, when we increased the command voltage to attempt to reduce the swings needed, the pendulum often overshot the vertical position and the balance controller was not able to catch it. Therefore, we decided to use a lower command input which takes more swings, but is much more reliable. Figures 49 and 50 show the simplified controllers as they were implemented in simulink and LabView, respectively. In all cases, the state-space balancer was used along with the simplified swing-up controller.

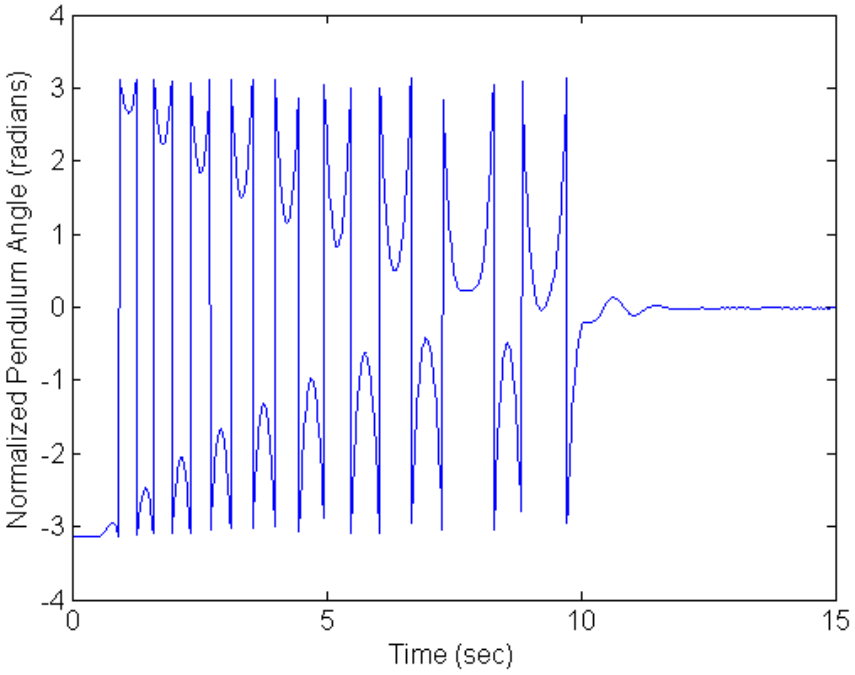

Figure 47: Simplified Swing-Up Controller on Real System

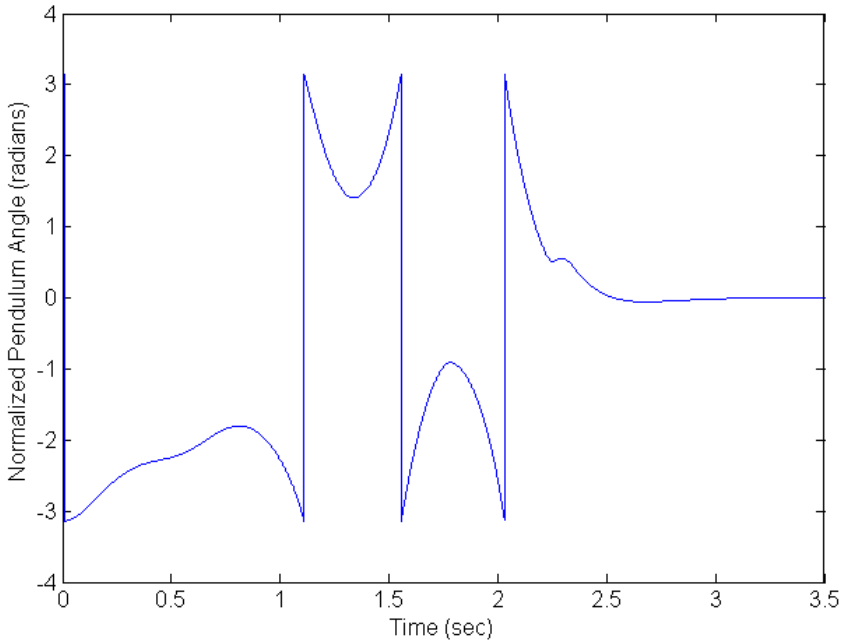

Figure 48: Simplified Swing-Up Controller with Nonlinear Simulink Model

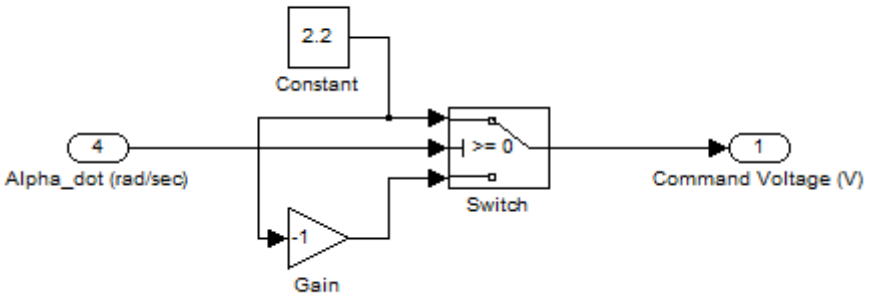

Figure 49: Simplified Swing-Up Controller in Simulink Model

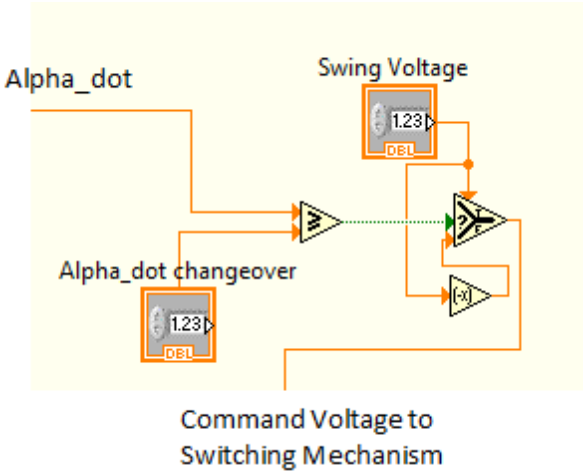

Figure 50: Simplified Swing-Up Controller in LabView

10 Parasitic Effects

- Overview of Modeling Parasitic Effects
  - Friction (see Fig. 51)
    Frictional torque was incorporated into the equations of motion using the coulombic friction model presented in problem 1. The frictional torques were identified experimentally.
  - Backlash/Deadband
    Elastic behavior of the motor shaft and the coupling between the shaft and the horizontal arm could introduce backlash/deadband behavior. The equations of motion were augmented to consider this effect. (Matlab function code).
  - Quantization (see Fig. 52)
    Two quantizer blocks were introduced to simulate the real behavior of optical encoders. According to the datasheets, the one used for measuring α has 1200 pulses per revolution (PPR), and the other one which measures θ has 2000 PPR with quadrature decoding.
  - Discretization
    Discretization was implemented by replacing all continuous Simulink blocks of the controller with discrete equivalents.
  - Saturation (see Fig. 53)
    The current limit of the servo amplifier was simulated through the addition of a $\pm$ 3V saturation block based on the datasheet to the command voltage. Motor back-emf was simulated to limit maximum angular velocities using $K_b = K_t$.
  - Noise (see Fig. 54)
    The input and output disturbances were modeled separately. Noise sources, with a frequency of $60Hz$ were added to the input to the motor driver and two the sensed θ and α values.
  - Time Delay (see Fig. 55)
    A time delay block was introduced between the "motor and driver model" and "Inverted Pendulum" blocks.

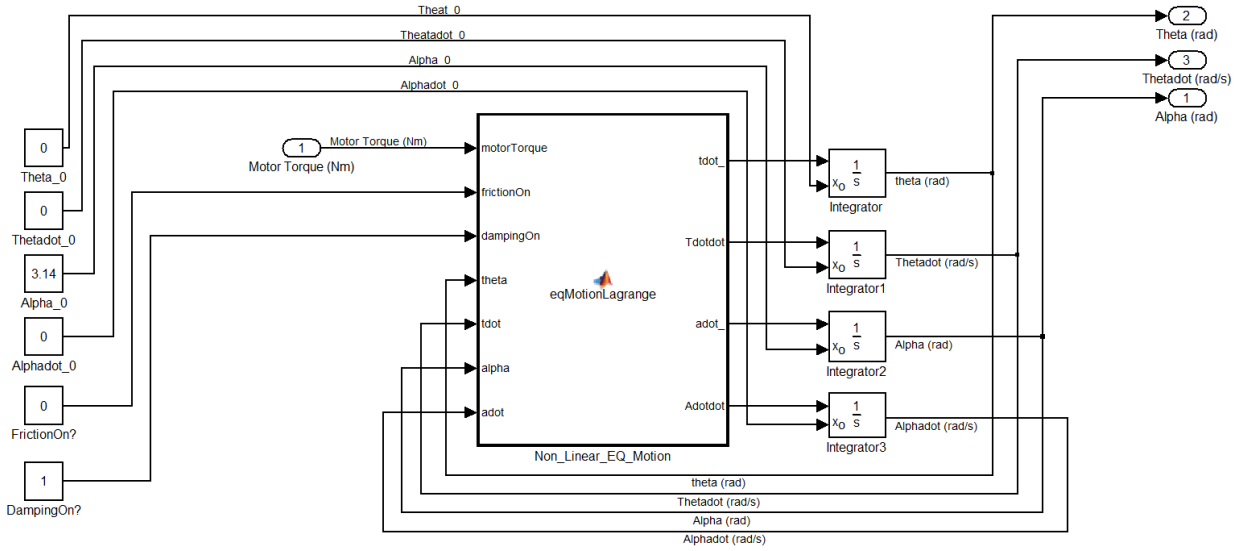

Figure 51: Modeling of Friction

- Friction (see Fig. 56)
  Friction dissipates more energy and as a result, more swings are needed for the pendulum to reach

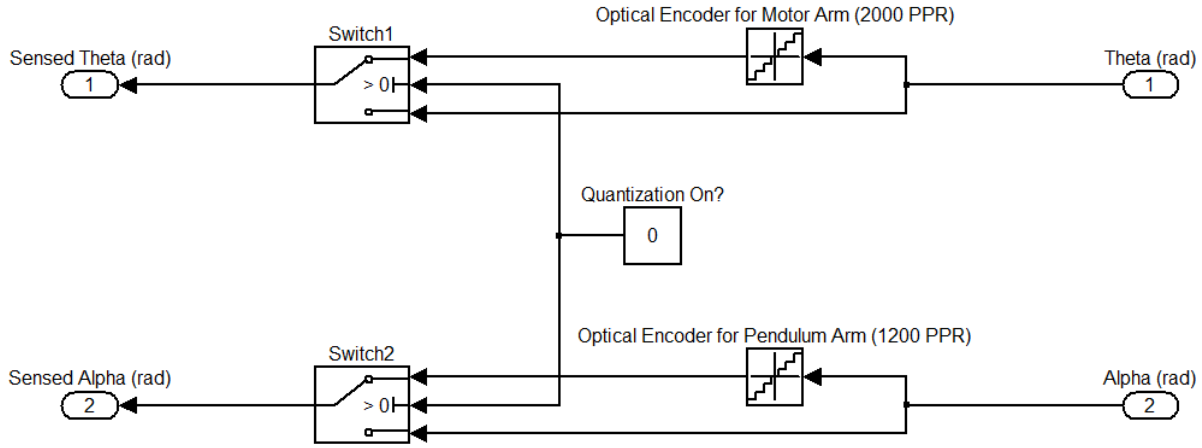

Figure 52: Modeling of Quantization

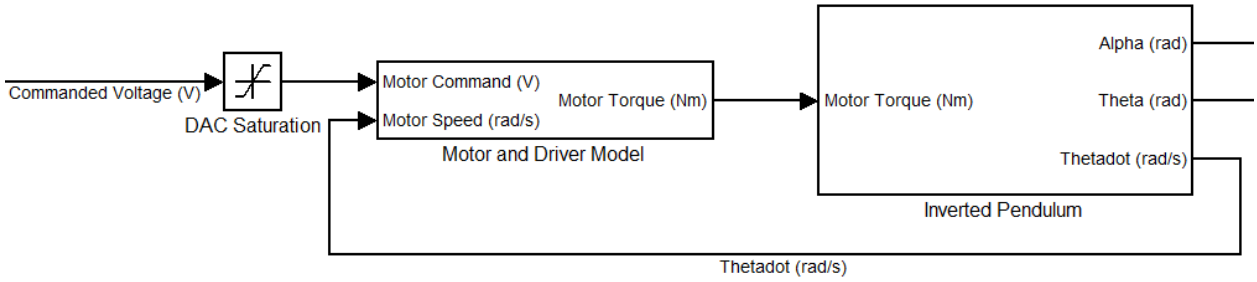

Figure 53: Modeling of Saturation

the range of the balancing controller. This increases the amount of time for the system to reach the balancing steady-state.

- Backlash / Deadband (see Fig. 57 and 58)
  Introducing backlash causes the system to take more time to reach the steady state. Furthermore, the magnitude of $\dot{\theta}$ seems to be smaller when swinging up but it oscillates within a small range during the balancing stage.

- Quantization (see Fig. 59, 60 and 61)
  In the real system, both optical encoders introduce quantization to the angle measurements. The simulation results show that during steady state, oscillations occur in α and θ. Unlike the original model which outputs very nearly zero torque, these oscillations in the angle measurements cause oscillations in motor torque, within the range of $0.2Nm$ to compensate for the perceived error.

- Discretization (see Fig. 62)
  The simulation results suggest that the system with discrete-time control performs better during the swing-up stage, reaching steady state sooner. However, if we increase the time step to a certain degree (Ts=0.005), the system becomes unstable, which means the discretization affects the system stability.

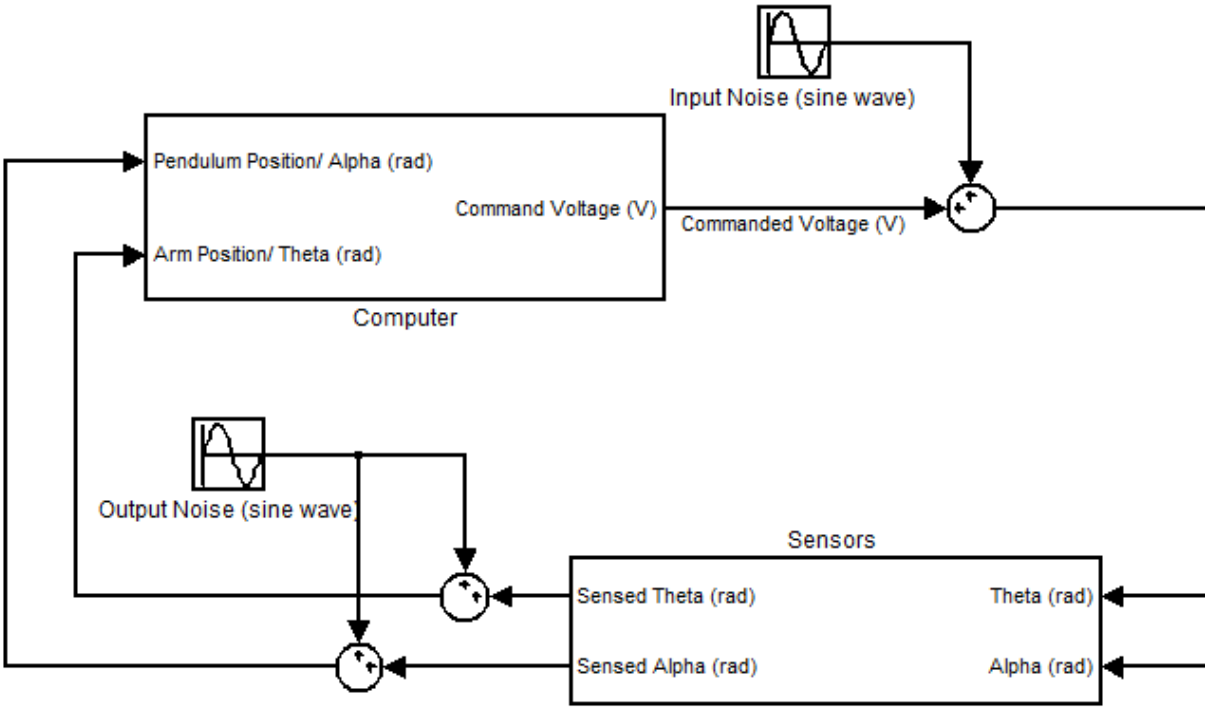

Figure 54: Modeling of Noise

- Saturation (see Fig. 63 and 64)
  As can be seen in Fig. 63 and 64, more time is required for the first swing when the saturation is considered. It suggests that previously large torque was available to provide sufficient energy for the pendulum system to swing at the beginning, but now more time is needed to add the same amount of energy with the limited torque.

- Noise (see Fig. 65 66 and 67)
  Both input and output disturbances have similar undesired influence on the system performance. Despite the tiny difference when the swing-up controller is working, during the balancing stage, oscillations occur in the α and θ. The range of oscillations depends on the magnitudes of noise. Noise added to sensor measurements has the greatest impact resulting in motor torques applied to the system during steady state.

- Time Delay (see Fig. 68, 69 and 70)
  In the physical model, the dynamic response of the servo-amplifier, encoders, and NI-DAQ are assumed instantaneous. In order to simulate the real behavior, a small time delay was introduced to our simulink model. For the swing-up stage, time delay has a small effect which simply causes the same amount of time shift. However, during the balancing stage, time delay has strong negative impact on the balancing control. α vibrates significantly as the time delay is increased; thus, the motor behaves aggressively and large torque is required during the transient response. This behavior in steady state resembles the oscillations seen in the real system and we may conclude that time delay is contributing to the poor system behavior.

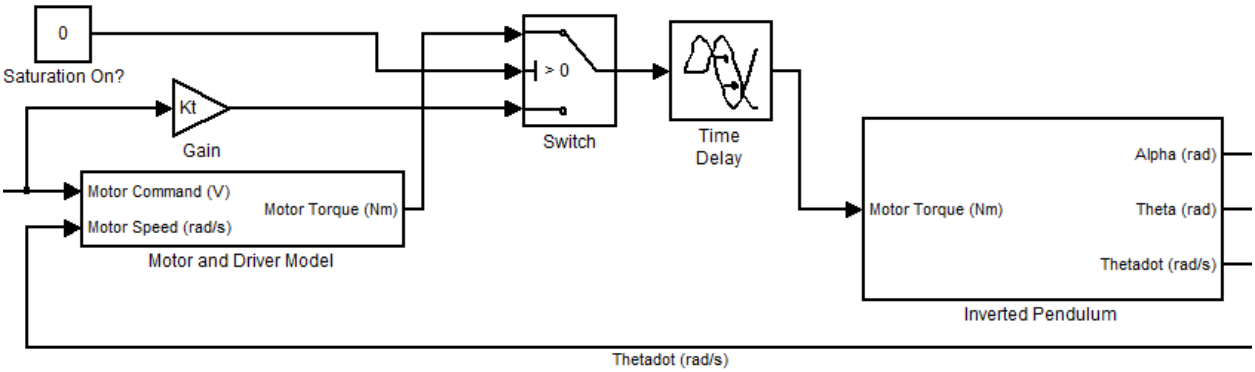

Figure 55: Modeling of Time Delay

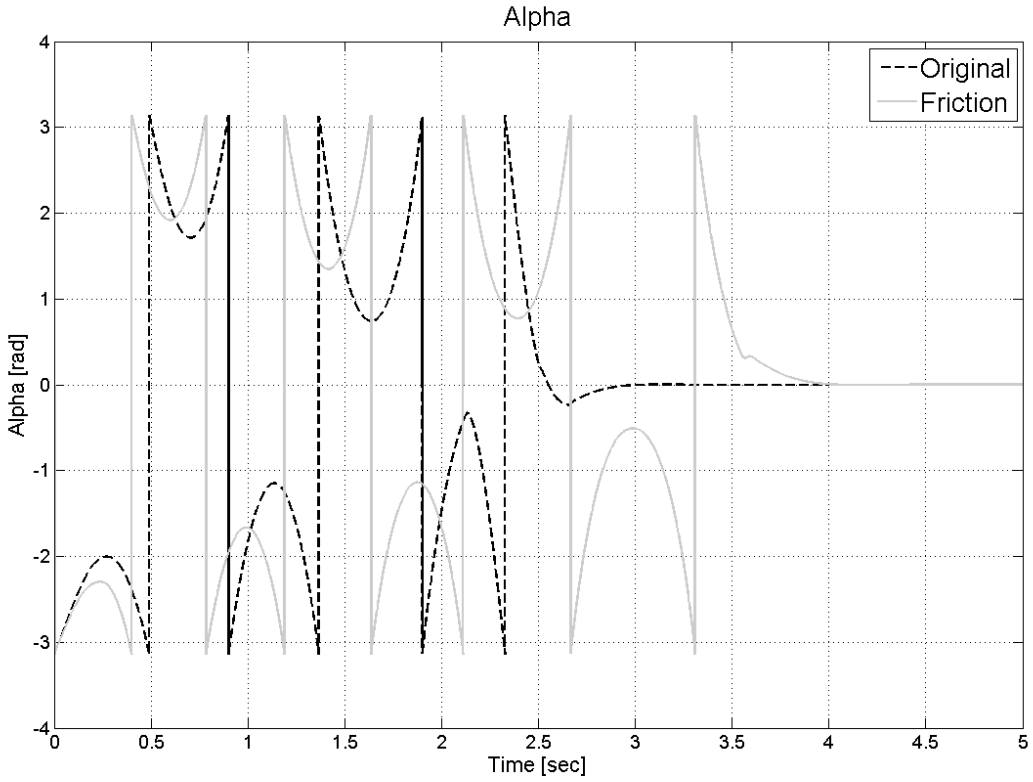

Figure 56: Behavior of α with Friction On

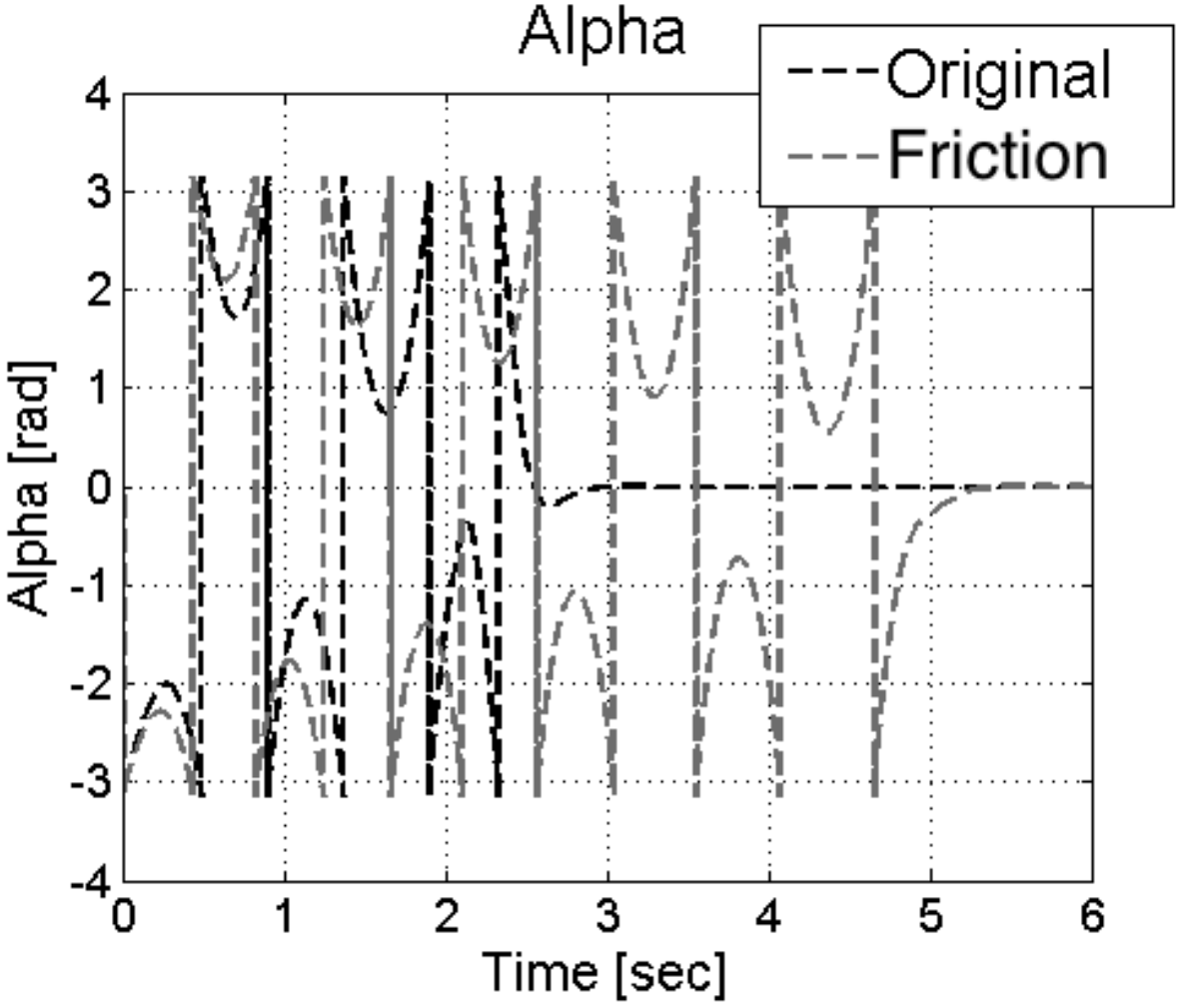

Figure 57: Behavior of α with Friction On

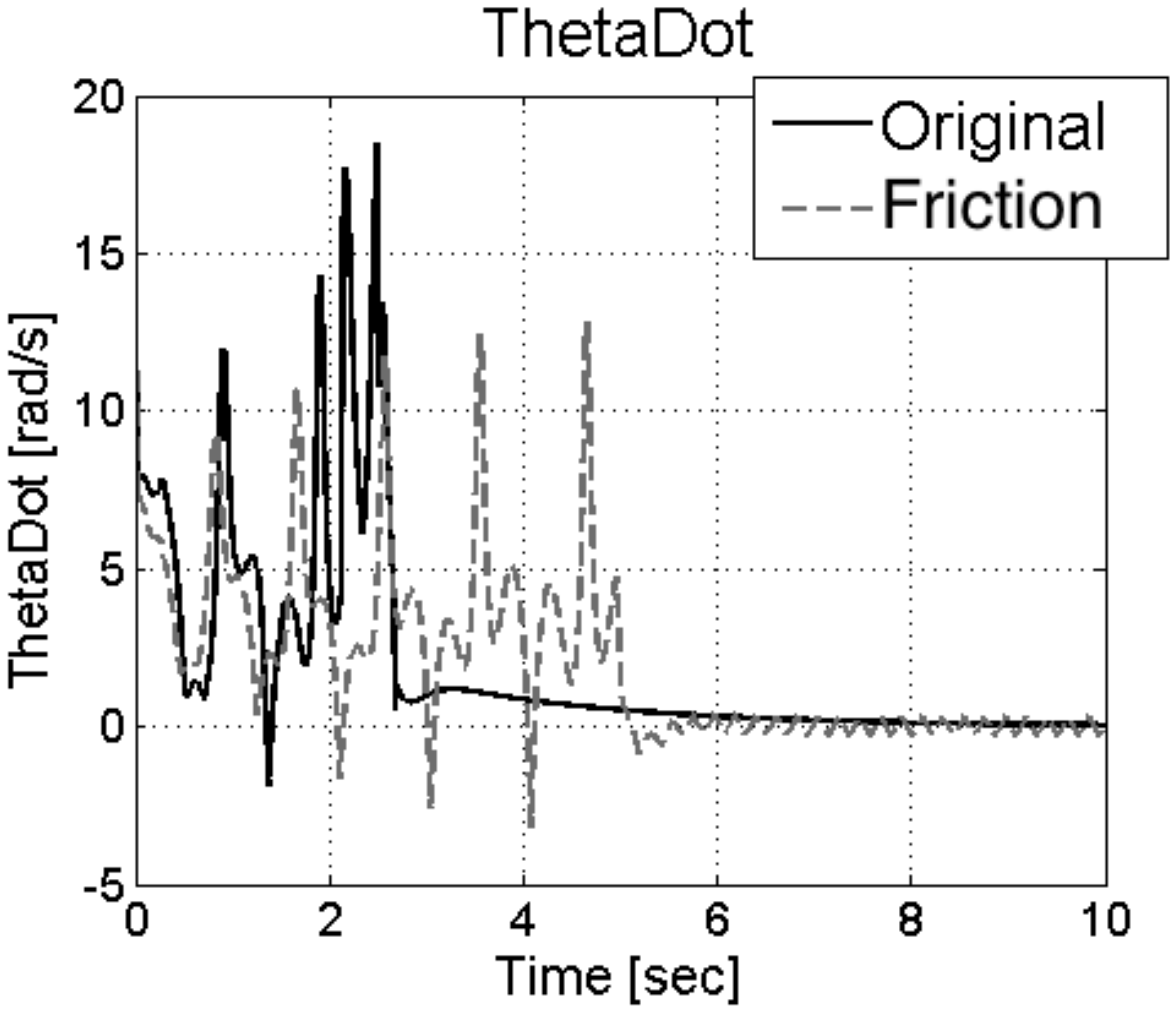

Figure 58: Behavior of $\dot{\theta}$ with Friction On

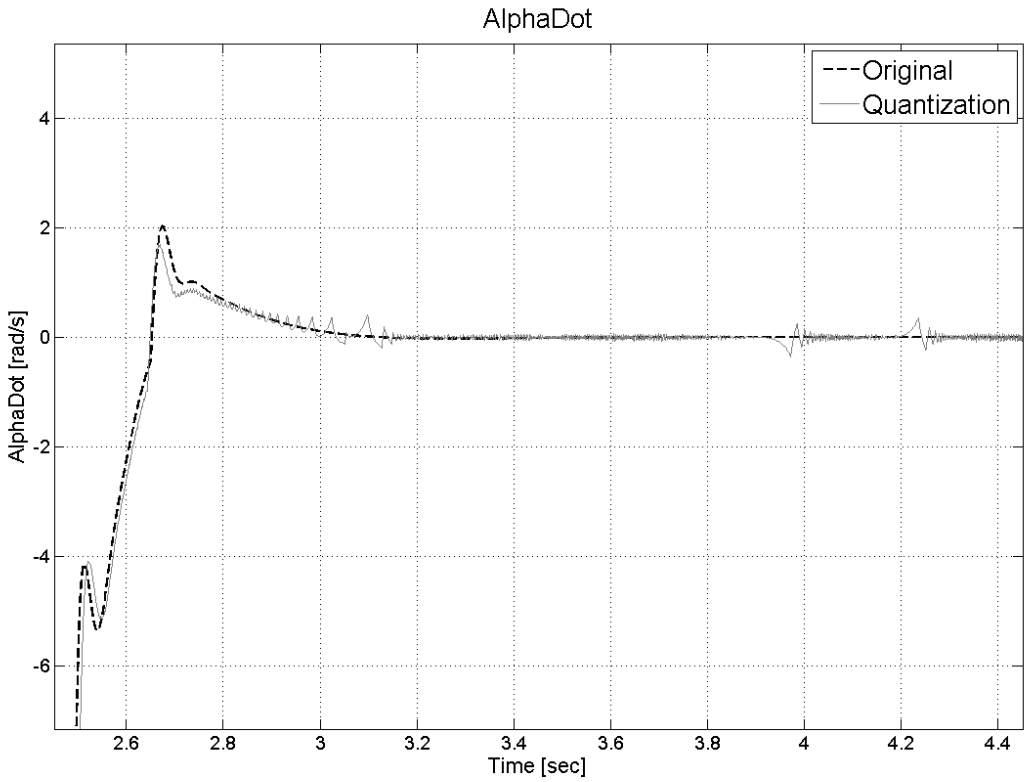

Figure 59: Behavior of $\dot{\alpha}$ with Quantization

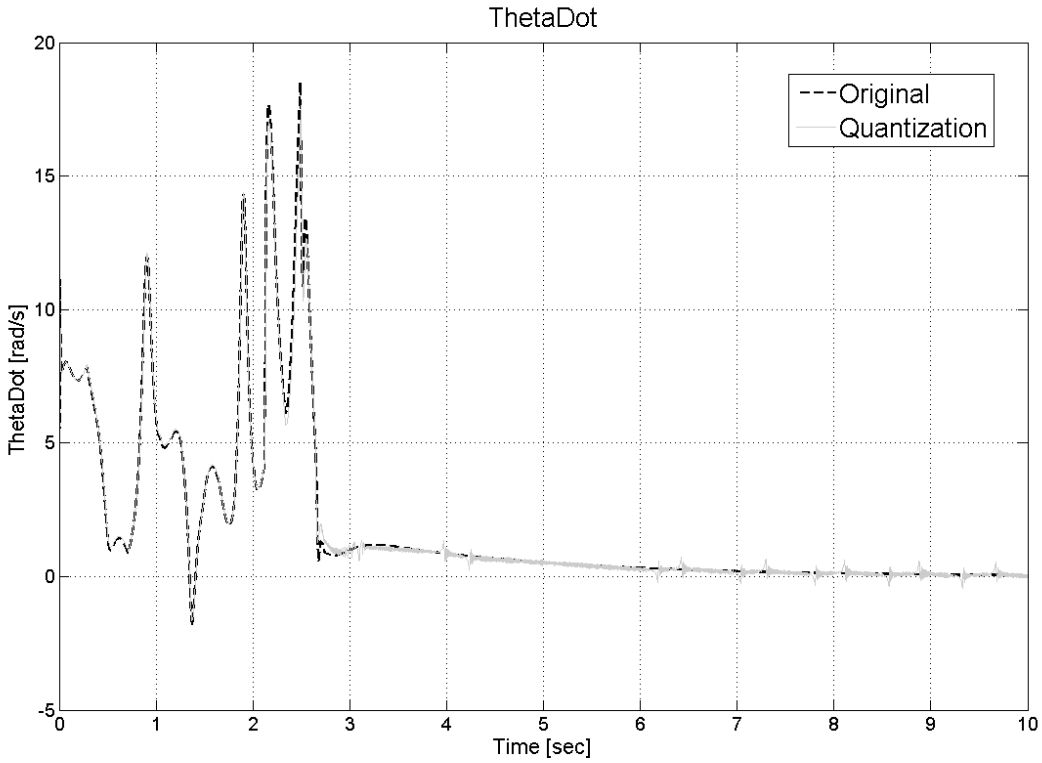

Figure 60: Behavior of $\dot{\theta}$ with Quantization

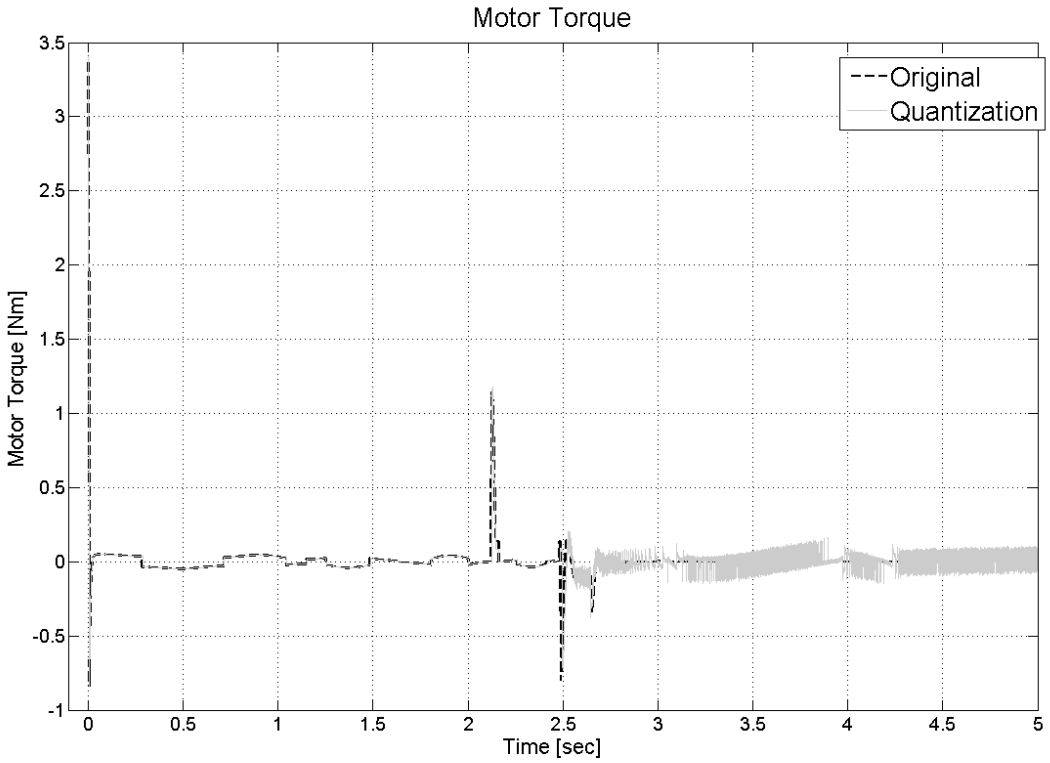

Figure 61: Behavior of Motor Toque with Quantization

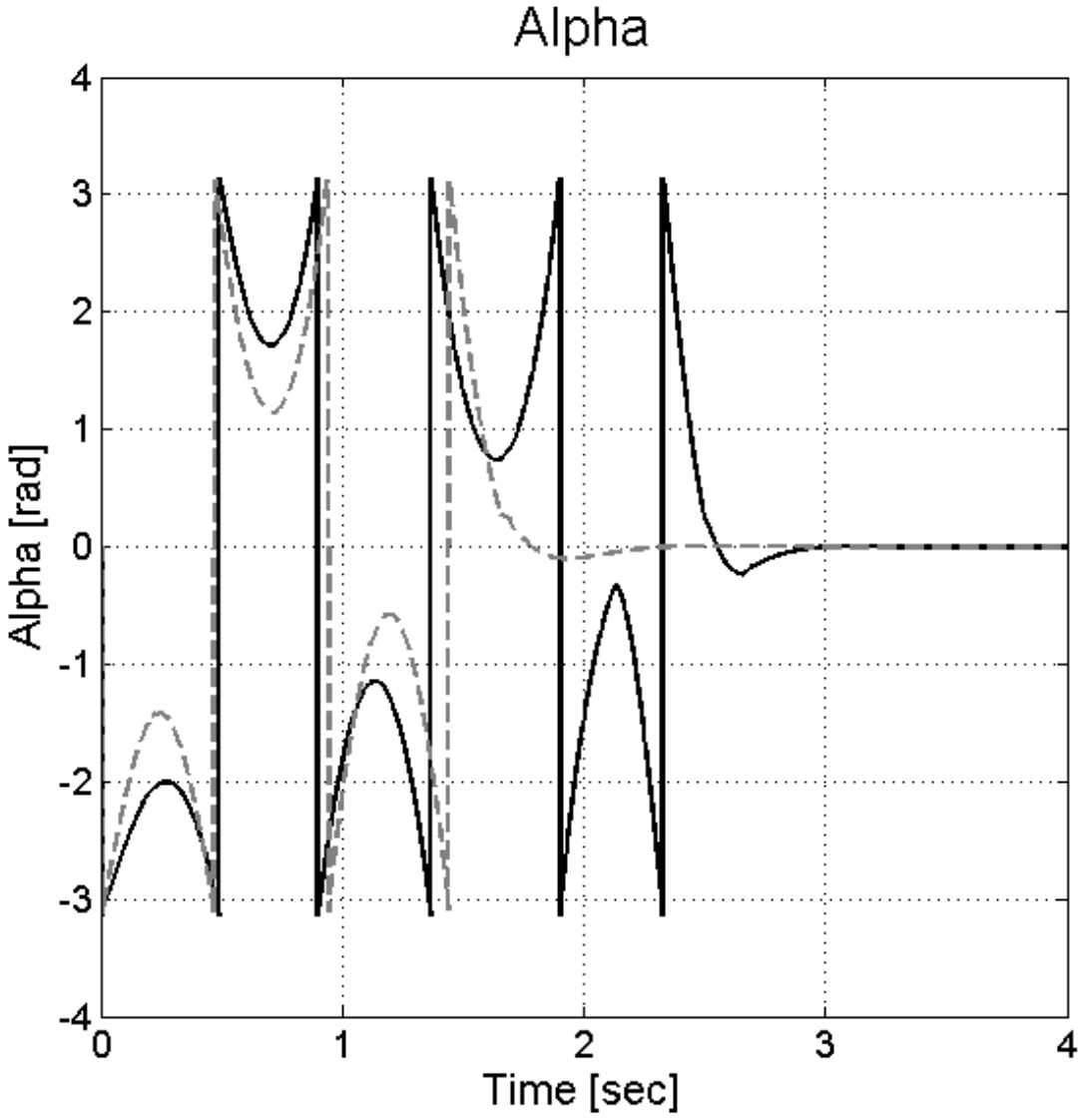

Figure 62: Behavior of α with Discretization

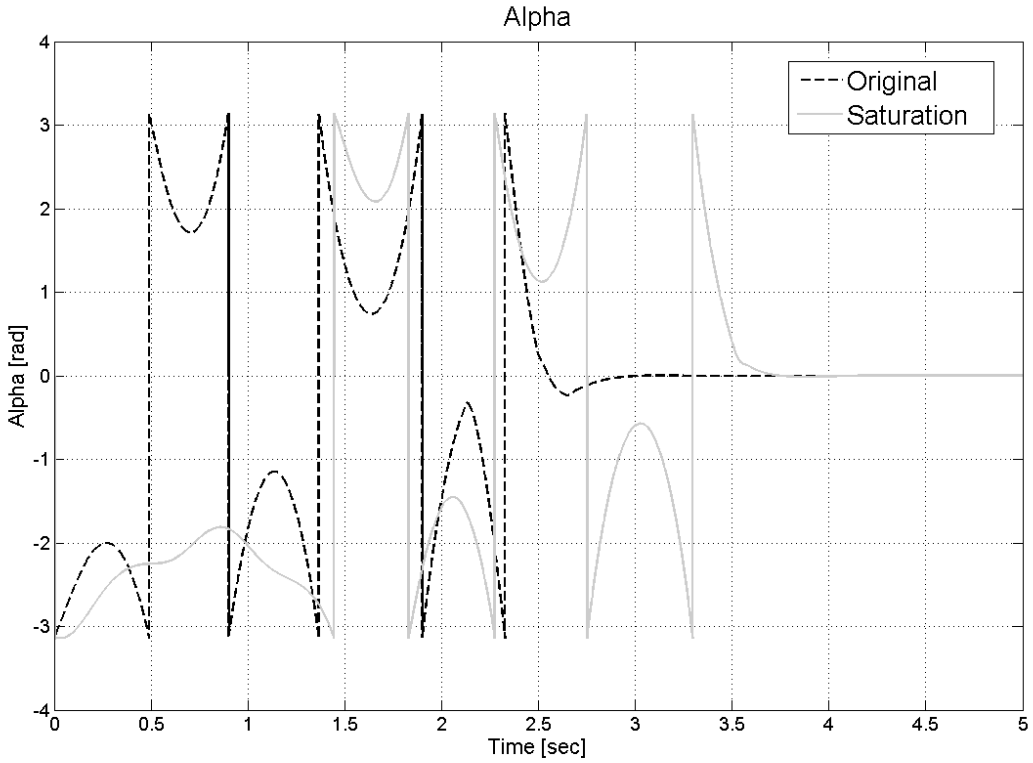

Figure 63: Behavior of α with Saturation

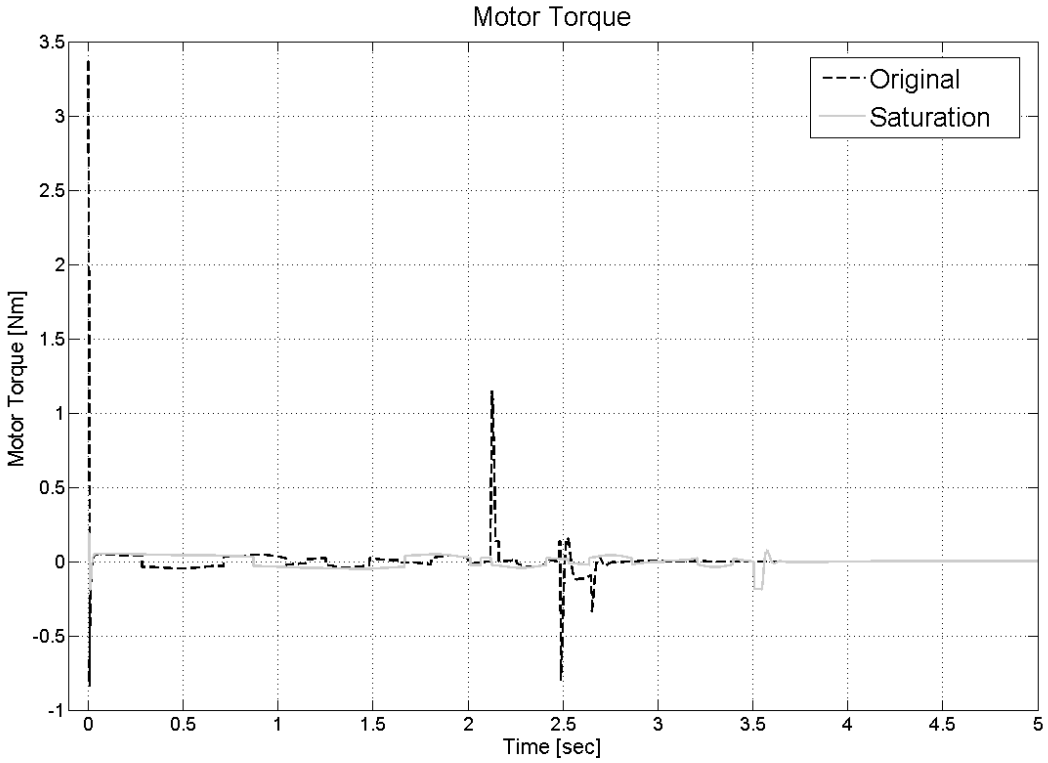

Figure 64: Behavior of Motor Torque with Saturation

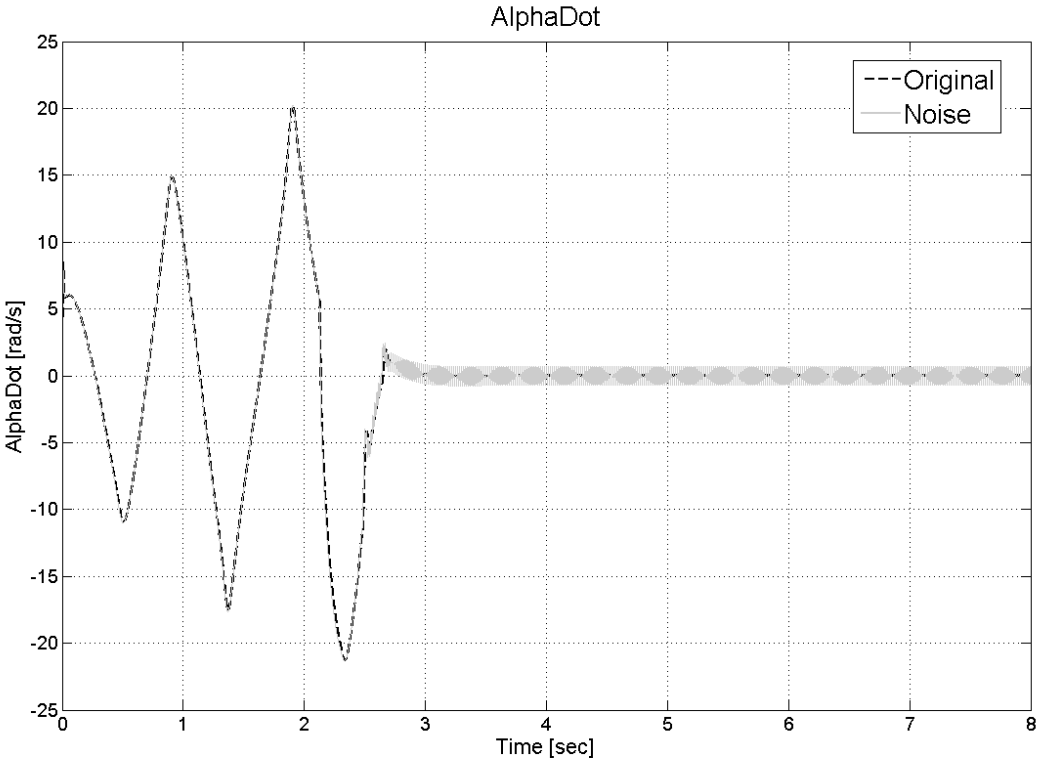

Figure 65: Behavior of $\dot{\alpha}$ with Noise

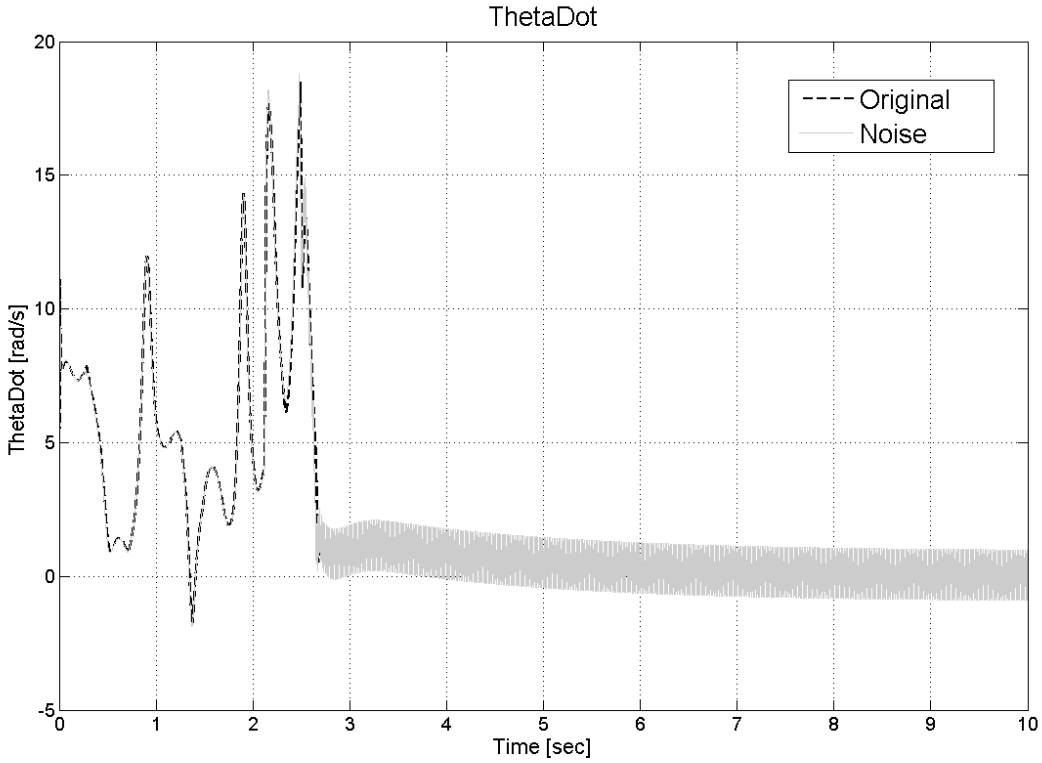

Figure 66: Behavior of $\dot{\theta}$ with Noise

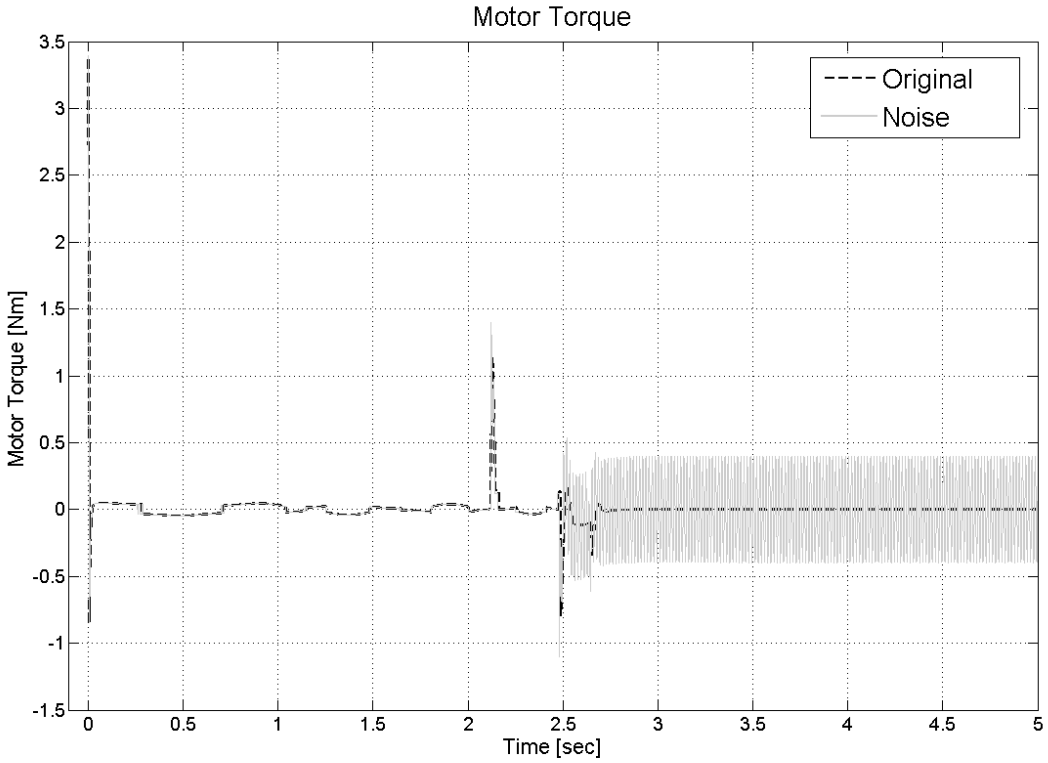

Figure 67: Behavior of Motor Torque with Noise

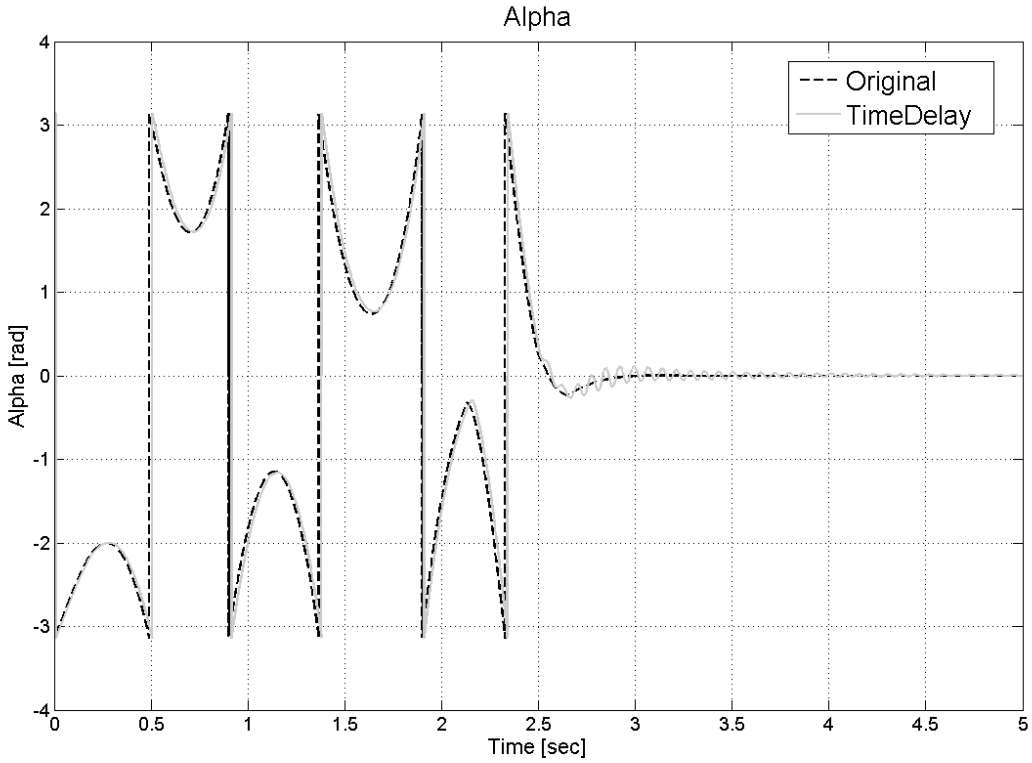

Figure 68: Behavior of α with Time Delay

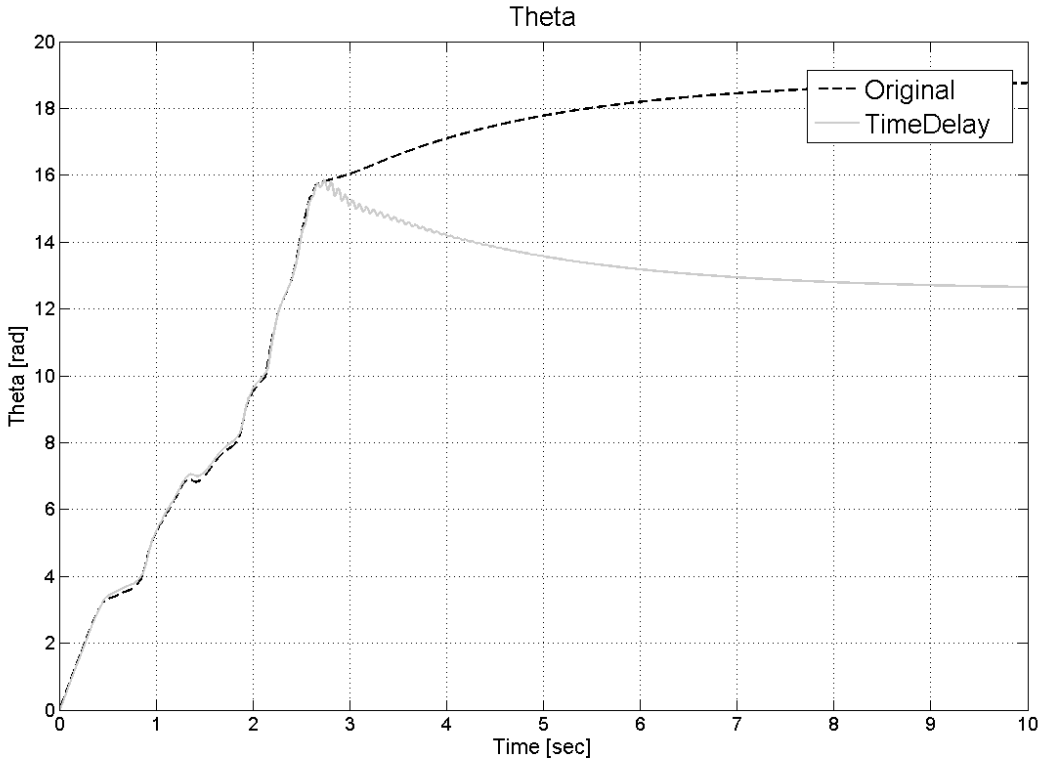

Figure 69: Behavior of θ with Time Delay

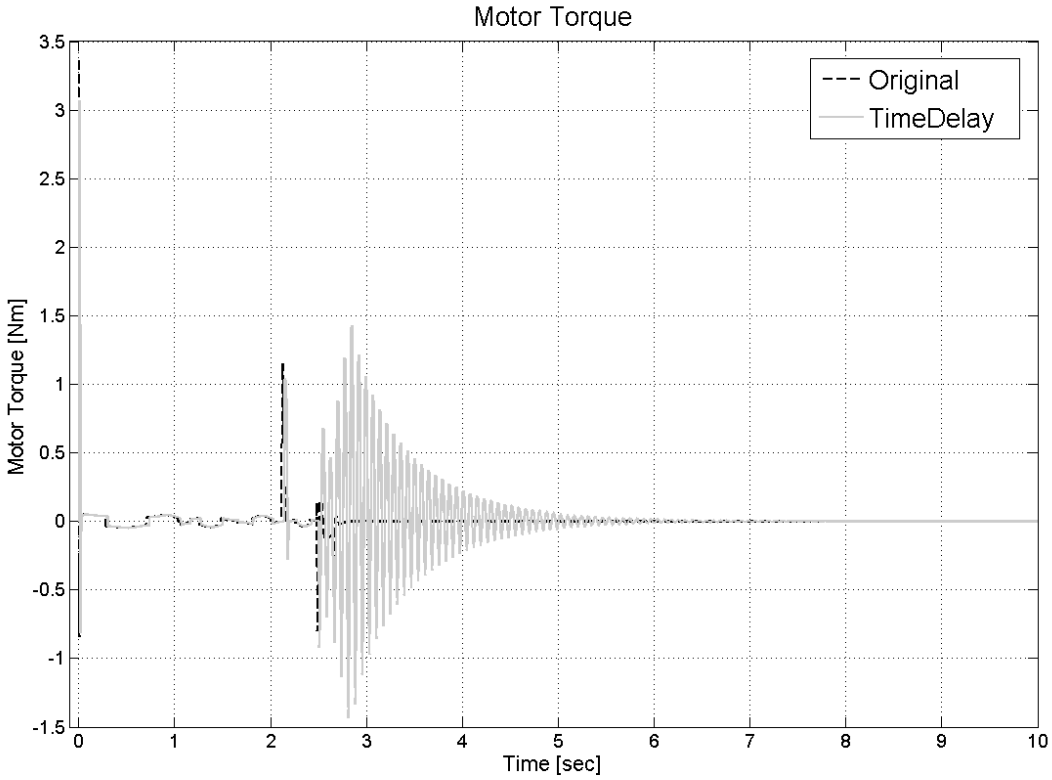

Figure 70: Behavior of Motor Torque with Time Delay

Appendix.

Equation of Motion from Lagrange Method

```
function [tdot_, Tdotdot, adot_, Adotdot]  = eqMotionLagrange(motorTorque, frictionOn , dampingOn ,theta
% we currently assume differential coulomb friction and damping for theta,
% but not for alpha
%global m2 I1zt Ix2 Iy2 Iz2 Ixz2 g l1 l2c ba btf btr

% Iz2 = 3.646e-4;
% l2c = 0.0856786;
% Iy2 = 3.711e-4;
% l1 = 0.1016;
% Ixz2 = 2.03e-5;
% I1zt = 6.78e-4;
% Ix2 = 8.1e-6;
% m2 = 0.0820142;
% g = 9.81; ba = 6.5E-6; btf = 1.08586e-5; btr = 4.8593e-5;
% %Tca = 7e-6; Tctf = 0.022439; Tctr = 0.0143096;
% Tca = 7e-6; Tctf = 0.00; Tctr = 0.00;

% theta = x(1); tdot = x(2); ct = cos(theta); st = sin(theta);
% alpha = x(3); adot = x(4); ca = cos(alpha); sa = sin(alpha);
tdot_ = tdot;
adot_ = adot;
%ct = cos(theta); st = sin(theta);
ca = cos(alpha); sa = sin(alpha);
%%%%%%%%%%
% Motor
Tm = motorTorque;  % placeholder for motor power term

%%%%%%%%%%
% Damping
% model damping differently forwards and backwards for theta
if dampingOn
    if (tdot > 0)
        Bt = btf;  % baf opposes forwards motion for theta
    elseif (tdot < 0)
        Bt = btr; % bar opposes reverse motion for theta
    else
        Bt = 0;
    end
    Ba = ba;
else
    Bt = 0;
    Ba = 0;
end
% for alpha only a single viscous damping term

%%%%%%%%%%%%
% Friction
if frictionOn
    TCt = coulombFriction(Tctf, Tctr, Tm, tdot);
```

```
    Tg = m2*g*l2c*sa; % torque exerted by gravity on pendulum
    TCa = coulombFriction(Tca, Tca, Tg, adot);
else
    TCa = 0;
    TCt = 0;
end


C1 = Iz2+m2*l2c*l2c;  C2 = Iy2+m2*l2c*l2c;   C3 = m2*l1*l2c-Ixz2;  C4 = l2c*m2*g;

D = (I1zt + sa*sa*C2 + Ix2*ca*ca + m2*l1*l1)*C1 - ca*ca*C3*C3;

Tdotdot = (1/D)*(-2*tdot*adot*sa*ca*(C2-Ix2)*C1 + adot*adot*sa*C3*C1 ...
    - tdot*tdot*sa*ca*ca*(C2-Ix2)*C3 + adot*Ba*ca*C3 - sa*ca*C3*C4 - TCa*ca*C3 ...
    - Bt*tdot*C1 + Tm*C1 + TCt*C1);

Adotdot = (1/C1)*(-Tdotdot*ca*C3 + tdot*tdot*sa*ca*(C2-Ix2) + C4*sa - Ba*adot + TCa);

end
% xdot = [tdot; Tdotdot; adot; Adotdot];

%-------------------------------------------------
function [torque] = coulombFriction(Tff, Tfr, externalTorque, thetadot)
% this friction model is making an assumption that isn't strictly true, we
% have static friction, but only the external torque exerted on each link
% will be considered.  This is not strictly correct, really we should
% consider the constraint forces as well
% it might not hold with the strength that it should

staticThreshold = 1e-6;
staticRatio = 1;

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

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

Linearized Equation of Motion about $\alpha = 0$

```
function [tdot_, Tdotdot, adot_, Adotdot] = eqMotionLinear(Tm, theta,tdot,alpha,adot, m2, I1zt, Ix2, Iy
tdot_ = tdot;
adot_ = adot;

%%%%%%%%%%%
% Linear Damping
% Neglect Friction

C1 = Iz2+m2*l2c*l2c;  C2 = Iy2+m2*l2c*l2c;   C3 = m2*l1*l2c-Ixz2;  C4 = l2c*m2*g;

C5 = I1zt + Ix2 + m2*l1*l1;  C6 = C5*C1-C3*C3;

Tdotdot = (1/C6)*(C3*ba*adot - C3*C4*alpha - C1*bt*tdot + C1*Tm);

Adotdot = (1/C6)*(-ba*adot*C5 + bt*tdot*C3 + alpha*C4*C5 - C3*Tm);
end
```

Linearized Equation of Motion about $\alpha = \pi$

```
function [tdot_, Tdotdot, adot_, Adotdot] = eqMotionLinearDown(Tm, theta,tdot,alpha,adot, m2, I1zt, Ix2
% linearized about alpha = pi, theta = 0
tdot_ = tdot;
adot_ = adot;

%%%%%%%%%%%
% Linear Damping
% Neglect Friction

C1 = Iz2+m2*l2c*l2c;  C2 = Iy2+m2*l2c*l2c;   C3 = m2*l1*l2c-Ixz2;  C4 = l2c*m2*g;

C7 = (I1zt + C2*(pi*pi - 2*pi*alpha) + Ix2 + m2*l1*l1)*C1 - C3*C3;

Tdotdot = (1/C7)*(C3*C4*(alpha - pi) - C3*ba*adot - C1*bt*tdot + C1*Tm);

Adotdot = (1/C1)*(Tdotdot*C3 + C4*pi - C4*alpha - ba*adot);
end
```

Envelop Extraction

```
function [T,Uout,Lout] = extractEnvelop(T,X,window)
% extracts lower and upper Envelopes of the given presumably sinusoidal
% function
assert((window > 0),'Window should have been positive');

% get upper and lower
[TUE, UE] = extractUpperEnvelop(T,X,window);
[TLE, LE] = extractLowerEnvelop(T,X,window);

Uout = interpolate(TUE,UE,T);
Lout = interpolate(TLE,LE,T);
```

```
end

function [TE, UE] = extractUpperEnvelop(T,X,w)
UEsub = [];
Tsub = [];
k = 1;
if (X(2) - X(1)) > 0
    increasing = 1;
    UEsub(1) = X(1);
    Tsub(1) = T(1);
else
    increasing = 0;
end
for i = 2:length(X)
    % if going up, keep going
    if (increasing) && (X(i) > UEsub(k))
        UEsub(k) = X(i);
        Tsub(k) = T(i);
    elseif (increasing) % as soon as we stop increasing, stop updating that one
        k = k + 1;
        increasing = 0;
    end

    if (~increasing) && (isIncreasing(X,i,w)) % if we weren't going up, now we are
        increasing = 1;
        UEsub(k) = X(i);
        Tsub(k) = T(i);
    end
end

% if we were still increasing when we quit, then get rid of the last point,
% since it isn't representative
if increasing
   UE = UEsub(1:(length(UEsub)-1));
   TE = Tsub(1:(length(Tsub)-1));
else
    UE = UEsub;
    TE = Tsub;
end

end

function [B] = isIncreasing(X,index,window)
% is the function increasing at that particular timestep

% look backwards if we can
if index - window < 1
    start = 1;
    finish = 1 + window;
else
    start = index - window;
    finish = index;
```

```
end

B = 1;
for i = (start + 1):finish
    if (X(i) - X(i-1)) <= 0
        B = 0;
        break;
    end
end

end

function [B] = isDecreasing(X,index,window)
% look backwards if we can
if index - window < 1
    start = 1;
    finish = 1 + window;
else
    start = index - window;
    finish = index;
end

B = 1;
for i = (start + 1):finish
    if (X(i) - X(i-1)) >= 0
        B = 0;
        break;
    end
end
end

function [TE, LE] = extractLowerEnvelop(T,X,w)
LEsub = [];
Tsub = [];
k = 1;
if (X(2) - X(1)) < 0
    decreasing = 1;
    LEsub(1) = X(1);
    Tsub(1) = T(1);
else
    decreasing = 0;
end
for i = 2:length(X)
    % if going up, keep going
    if  (decreasing) && (X(i) < LEsub(k))
        LEsub(k) = X(i);
        Tsub(k) = T(i);
    elseif (decreasing) % as soon as we stop increasing, stop updating that one
        k = k + 1;
        decreasing = 0;
    end
```

```
    if (~decreasing) && (isDecreasing(X,i,w)) % if we weren't going up, now we are
        decreasing = 1;
        LEsub(k) = X(i);
        Tsub(k) = T(i);
    end
end

if decreasing
   LE = LEsub(1:(length(LEsub)-1));
   TE = Tsub(1:(length(Tsub)-1));
else
    LE = LEsub;
    TE = Tsub;
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
```

Variable Initialization for Linear Simulink model

```
% version 2 of the parameter initialization

% note that positive theta is counter clockwise when viewed from above
```

```
% positive alpha is clockwise when viewed from the end of the arm
% x axis points along pendulum
% z axis along arm towards motor
% y is consistent with right handed coordinate frame

% note that pendulum parameters have been obtained through a combination of
% solid works modeling to provide center of mass and moments of inertia
% and a rough eyeball fit of the friction and damping
% arm damping and friction terms are carried over from the previous
% experiment but this may not be quite the case with an off balance torque
% on the shaft

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% physical parameters

% Motor and Driver Parameters
Kt = 0.0314499;     % motor torque constant
%Kb = 0.04;          % yes technically these should be equal, this is based on 600 rad/s
                    % for a supply voltage of 24 volts only effects driver /
                    % motor model
%Rm = 1.85;          % motor resistance based on data sheet
%Vsupply = 24;       % supp,y voltage
Ka = 1;             % nominal driver voltage to current ratio


% lengths
l1 = 0.10825640;
l2c = 0.0856786;

% mass
m2 = 0.0820142;

% moments of inertia
Ix2 = 8.1e-6;
Iy2 = 3.711e-4;
Iz2 = 3.646e-4;
Ixz2 = 2.03e-5;
I1zt = 6.78e-4 + 3.50514e-5;    % arm moment of inertia obtained from solid works
                                % and rotor inertia obtained from earlier
                                % experiments

% damping
ba = 6.5E-6;        % alpha / pendulum
btf = 1.08586e-5;   % forwards theta / arm
btr = 4.8593e-5;    % reverse theta

% friction is dropped in a linear model
%Tca = 7e-6;             % alpha / pendulum
%Tctf = 0.022439;        % forwards theta / arm
%Tctr = 0.0143096;       % reverse theta
%staticThreshold = 1e-6; % below this velocity we aren't moving
%staticRatio = 1;        % multiplier for static friction over normal friction
```

```
% alternate friction quantities which do not illustrate a limit cycle
%Tctf = 0.0001; Tctr = 0.0001;

% gravity
g = 9.81;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% transfer functions
% this linear form requires us to neglect friction and assume isotropic
% damping

% constants
C1 = Iz2+m2*l2c*l2c;  C2 = Iy2+m2*l2c*l2c;  C3 = m2*l1*l2c-Ixz2;
C4 = l2c*m2*g;        C5 = I1zt+Ix2+m2*l1*l1;   C6 = C5*C1-C3*C3;

btavg = (btf + btr)/2;
bt = btavg;

% with Damping
DenB = [(C5*C1-C3*C3), (C1*btavg+C5*ba), (ba*btavg-C4*C5), (-C4*btavg), 0];
NumBT = [C1, ba, -C4];
NumBA = [-C3, 0, 0];

tfTB = tf(NumBT,DenB);
tfAB = tf(NumBA,DenB);

% without damping
Den = [(C5*C1-C3*C3), 0, -C4*C5, 0, 0];
NumT = [C1, 0, -C4];
NumA = [-C3, 0, 0];

tfT = tf(NumT,Den);
tfA = tf(NumA,Den);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Controllers

% Alpha Inner loop
ACZeros = [-3.75; -7.85];
ACPoles = [-50; 0];
ACK = -3.7203;
[NumAC, DenAC] = zp2tf(ACZeros,ACPoles,ACK);
AController = tf(NumAC,DenAC);

% having the controller changes our theta response
NumTBwAC = conv(NumBT,NumAC);
DenTBwAC = polyCoadd(conv(DenB,DenAC),conv(NumAC,NumBA));
tfTBwAC = tf(NumTBwAC,DenTBwAC);

% Theta Outer Loop will need positive feedback
% just a simple lead controller
TCZeros = [-1];
```

```
TCPoles = [-15];
TCK = 0.01784;
[NumTC,DenTC] = zp2tf(TCZeros,TCPoles,TCK);
TController = tf(NumTC,DenTC);

% Final Closed Loop Transfer Functions to take our root locii from later
% Alpha
NumAF = conv(NumAC,NumBA);
DenAF = conv(DenAC,DenB);
tfAF = tf(NumAF,DenAF);
%rlocus(tfAF);

% Theta
NumTF = conv(NumTBwAC,NumTC);
DenTF = conv(DenTBwAC,DenTC);
tfTF = tf(NumTF,DenTF);
%rlocus(tfTF);
```

Angle Normalization

```
function [tout] = mod2pi(t)
% returns an angle between pi and -pi

if t < 0
    tout = -mod2pi_sub(-t);
else
    tout = mod2pi_sub(t);
end
end

function [tout] = mod2pi_sub(t)
% only good for positive numbers
tout = t - floor(t*0.5/pi + 0.5) * 2*pi;
end
```

Pendulum Equation of Motion

```
function [alphadot, alphadotdot] = pendulumEQMotion(alpha, alphadot)
% Equation of motion for the pendulum by itself
% expects the following variables to be in the workspace
% g - gravity
% Iz2 - moment of inertia of pendulum about center of mass
% m2 - pendulum mass
% l2c - distance from axis of rotation to center of mass
% ba - viscous damping on pendulum
% Tca - torque exerted by static friction

sa = sin(alpha);

Tg = m2*g*l2c*sa;
Tc = coulombFriction(Tca, Tg, alphadot);
```

```
D = Iz2 + m2*l2c*l2c;

alphadotdot = (1/D)*(-ba*alphadot + m2*g*l2c*sa + Tc);
end

function [torque] = coulombFriction(Tf, externalTorque, thetadot)
% this friction model is making an assumption that isn't strictly true, we
% have static friction, but only the external torque exerted on each link
% will be considered.  This is not strictly correct, really we should
% consider the constraint forces as well
% it might not hold with the strength that it should
staticThreshold = 1e-6;
staticRatio = 1.1;

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

else % if we are moving oppose the actual motion with kinetic friciton
    if thetadot > staticThreshold
        torque = -Tf;
    else
        torque = Tf;
    end
end
end
```

Prototype for Pendulum Parameter Regression

```
function [stateC] = pendulumEstimation(Tin, Alphain, xstart, state0, lockedVariables)
% only consider the pendulum by itself, because it is much simpler

% state for simulation
% x = [alpha; alphadot]

% state for optimization
% state = [Iz2 m2 l2c ba Tca]

assert((length(state0) == 5), 'Invalid state length, expected 5');
assert((length(state0) == length(lockedVariables)),'should have been the same length');
assert((length(xstart) == 2), 'We need initial conditions');

global state g staticRatio staticThreshold windowSize TIMES
state = state0;
g = 9.81;
staticRatio = 1.1; staticThreshold = 1e-8; windowSize = 20; TIMES = 3;

global Tobs Alphaobs LPos numUPos
```

```
Tobs = Tin;
Alphaobs = Alphain;
LPos = lockedVariables;

% 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

% nominal state
%state = [3.646e-4; 0.0820142; 0.0856786; 1e-5; 1e-5];
%xstart = [pi/2; 0];

%%%%%%%%%%%%%%%
% Initial
%%%%%%%%%%%%%%%
[T, X] = shootCompare(xstart, state);
stateC = state0;

plotEnvelopes(T,X)
plotCompare(T,X);
%plotEnergy(T,X);

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 = jacobianShootPendulum(xstart,stateC);

    ds = J \ E;

    % going 100% of the dP is causing trouble, going less distance should
    % help
```

```
    stateC = updateState(stateC,ds);

    iter = iter + 1;
end

E = computeError(xstart,stateC);

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

%%%%%%%%%%%%
% Final
[T, X] = shootCompare(xstart, state);

plotComparey(T,X);
plotEnergy(T,X);

end

function [J] = jacobianShootPendulum(xstart,state0)
% compute the jacobian of the error vector in the neighborhood of the
% values P0

global Tobs numUPos LPos

[T0,X0] = shootCompare(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] = shootCompare(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 Alphaobs

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

E = Alphaobs - X(:,1);
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 [T, X] = shootNormal(xstart,tend, state_)
global state
state = state_;

options = odeset('RelTol',1e-6,'AbsTol',1e-6);
[T, X] = ode45(@EQMotion,[0 tend],xstart,options);
end

function [T, X] = shootCompare(xstart, state_)
global Tobs state TIMES
state = state_;
%options = odeset('RelTol',1e-6,'AbsTol',1e-6);
%[T, X] = ode45(@EQMotion,Tobs,xstart,options);
Tint = interpolateTime(Tobs,TIMES);
Xint = ode5(@EQMotion,Tint,xstart);
X = reduceTime(Xint,TIMES);
T = Tobs;
end

function [Tout] = interpolateTime(T,times)
Tout = zeros((length(T)*times - (times-1)), 1);
```

```
for i = 1:(length(T)-1)
    dT = T(i+1) - T(i);
   for j = 0:(times-1)
       Tout((i*times-(times-1))+j) = T(i) + j*dT/times;
   end
end

Tout(end) = T(end);

end

function [Tout] = reduceTime(T,times)
% times is the times that was applied with interpolate
Tout = zeros((length(T)+(times-1))/times,1);

for i = 1:length(Tout)
   Tout(i) = T(i*times-(times-1));
end
end

function [xdot] = EQMotion(t,x)
% equation of motion for just the pendulum not all that many parameters
global state g

Iz2 = state(1); m2 = state(2); l2c = state(3); ba = state(4); Tca = state(5);

alpha = x(1); alphadot = x(2); sa = sin(alpha);

Tg = m2*g*l2c*sa;
Tc = coulombFriction(Tca, Tg, alphadot);

D = Iz2 + m2*l2c*l2c;

alphadotdot = (1/D)*(-ba*alphadot + m2*g*l2c*sa + Tc);

xdot = [alphadot; alphadotdot];
end


function [torque] = coulombFriction(Tf, externalTorque, thetadot)
% this friction model is making an assumption that isn't strictly true, we
% have static friction, but only the external torque exerted on each link
% will be considered.  This is not strictly correct, really we should
% consider the constraint forces as well
% it might not hold with the strength that it should
global staticThreshold staticRatio

if (abs(thetadot) < staticThreshold) % if not moving, friction opposes attempted motion
    % if external torque does not exceed static friction
    if ((externalTorque < Tf) && (externalTorque >= 0)) || ((externalTorque > -Tf) && (externalTorque <
        FF = abs(externalTorque);
```

```
    else
        FF = Tf;
    end
    torque = - staticRatio * FF * sign(externalTorque);

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

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

Xout = zeros(length(Tout),size(Xin,2));

i = 1; % keep track of positon in the original
for o = 1:length(Tout)
    for j = 1:size(Xin,2)
        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,j) - Xin(1,j)) / dt;
            dt = Tout(o) - Tin(1);
            Xout(o,j) = Xin(1,j) + slope * dt;
        elseif (i > length(Tin))
            % linearly extrapolate - subject to noise, be careful
            dt = Tin(end) - Tin(end - 1);
            slope = (Xin(end,j) - Xin(end-1,j)) / dt;
            dt = Tout(o) - Tin(end);
            Xout(o,j) = Xin(end,j) + slope * dt;
        else
            dt = Tin(i) - Tin(i - 1);
            slope = (Xin(i,j) - Xin(i-1,j)) / dt;
            dt = Tout(o) - Tin(i-1);
            Xout(o,j) = Xin(i-1,j) + slope * dt;
        end
    end
end
end


function [] = plotCompare(T, X)
global Tobs Alphaobs
% plot the system
figure
hold on
```

```
plot(T,X(:,1),'k');
plot(Tobs,Alphaobs,'b')
legend('Alpha','Alpha Observed');
xlabel('Time (seconds)')
ylabel('Angle (radians)')
title('Passive Pendulum Behavior')
end

function [] = plotTrajectory(T, X)
% plot the system
figure
hold on
plot(T,X(:,1),'k');
plot(T,X(:,2),'b');
legend('Alpha','Alphadot');
xlabel('Time (seconds)')
ylabel('Angle (radians)')
title('Passive Pendulum Behavior')
end

function [] = plotEnergy(T,X)
% plots energy of the system

PE = potentialEnergy(X);
KE = kineticEnergy(X);
Total = PE + KE;

figure;
hold on;
plot(T,PE,'g');
plot(T,KE,'r');
plot(T,Total,'k');
xlabel('Time (seconds)');
ylabel('Energy');
legend('Potential','Kinetic','Total');
title('Energy');
hold off

end

function [] = plotEnvelopes(T,X)
global Tobs Alphaobs windowSize
[Tout,UA,LA] = extractEnvelop(Tobs,Alphaobs,windowSize);
[Tout,UX,LX] = extractEnvelop(T,X,windowSize);
figure
hold on
plot(Tout,UA,'b')
plot(Tout,LA,'b')
plot(Tout,UX,'k')
plot(Tout,LX,'k')
xlabel('Time (seconds)');
ylabel('Angle (radians');
```

```
title('Trajectory Envelope');
hold off
end

function KE = kineticEnergy(X)
% what is the kinetic energy of the system
global state
Iz2 = state(1); m2 = state(2); l2c = state(3);

KE = 0.5 * (Iz2 + m2*l2c*l2c) * X(:,2) .* (X(:,2));

end

function PE = potentialEnergy(X)
% what is the potential energy of the system
global state g
m2 = state(2); l2c = state(3);

PE = l2c*m2*g*(cos(X(:,1)) - 1);

end

function [] = test(T)
last = -1;
for i = 2:length(T)
    if (T(i) - T(i - 1)) < 0
        disp('we went down!')
        i
    end
end
end
```

Helper Function

```
function [pSum] = polyCoadd(p1, p2)
% returns sum of two arrays representing polynomial coefficients

if length(p1) >= length(p2)
    pSum = p1;
    for i = length(p2):-1:1
        pSum(length(pSum) - i + 1) = pSum(length(pSum) - i + 1) + p2(length(p2) - i + 1);
    end
else
    pSum = p2;
    for i = length(p1):-1:1
        pSum(length(pSum) - i + 1) = pSum(length(pSum) - i + 1) + p1(length(p1) - i + 1);
    end

end
end
```

Motor supply voltage saturation modeling

```
function Im = saturation(Vcmd, Vsupply, thetadot, Kb, Ka, Rm)
% determines motor current as a function of the command voltage, supply
% voltages, angular speed, and motor back emf constant
% Vsupply are the supplied votages to the power supply
% thetadot is the current angular speed of the motor
% Kb is the back emf constant
% Ka is the nominal motor driver gain
% R is nominanal motor resistance

% for thetadot no load = 600 rad/s then Kb = 0.04

% for the given command current, how much voltage would we need
Vb = Kb*thetadot;
Icmd = Ka * Vcmd;
Vrequired = Vb + Rm * Icmd;

if abs(Vrequired) < Vsupply
    Im = Icmd; % if we can supply enough voltage, then we can send this much current
else
    % otherwise, need to figure out what current we are limited to
    if Vcmd > 0
        Im = (Vsupply - Vb) / Rm;
    else
        Im = (-Vsupply - Vb) / Rm;
    end
end
end
```

Variable Initialization for State Space Controller

```
% version 2 of the parameter initialization

% note that positive theta is counter clockwise when viewed from above
% positive alpha is clockwise when viewed from the end of the arm
% x axis points along pendulum
% z axis along arm towards motor
% y is consistent with right handed coordinate frame

% note that pendulum parameters have been obtained through a combination of
% solid works modeling to provide center of mass and moments of inertia
% and a rough eyeball fit of the friction and damping
% arm damping and friction terms are carried over from the previous
% experiment but this may not be quite the case with an off balance torque
% on the shaft

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% physical parameters

% Motor and Driver Parameters
Kt = 0.0314499;     % motor torque constant
Kb = 0.04;          % yes technically these should be equal, this is based on 600 rad/s
```

```
                    % for a supply voltage of 24 volts only effects driver /
                    % motor model
Rm = 1.85;          % motor resistance based on data sheet
Vsupply = 24;       % supp,y voltage
Ka = 1;             % nominal driver voltage to current ratio


% lengths
l1 = 0.10825640;
l2c = 0.0856786;

% mass
m2 = 0.0820142;

% moments of inertia
Ix2 = 8.1e-6;
Iy2 = 3.711e-4;
Iz2 = 3.646e-4;
Ixz2 = 2.03e-5;
I1zt = 6.78e-4 + 3.50514e-5;    % arm moment of inertia obtained from solid works
                                % and rotor inertia obtained from earlier
                                % experiments

% damping
ba = 6.5E-6;        % alpha / pendulum
btf = 1.08586e-5;   % forwards theta / arm
btr = 4.8593e-5;    % reverse theta

% friction
Tca = 7e-6;             % alpha / pendulum
Tctf = 0.022439;        % forwards theta / arm
Tctr = 0.0143096;       % reverse theta
staticThreshold = 1e-6; % below this velocity we aren't moving
staticRatio = 1;        % multiplier for static friction over normal friction
% alternate friction quantities which do not illustrate a limit cycle
%Tctf = 0.0001; Tctr = 0.0001;

% gravity
g = 9.81;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% transfer functions
% this linear form requires us to neglect friction and assume isotropic
% damping

% constants
C1 = Iz2+m2*l2c*l2c;  C2 = Iy2+m2*l2c*l2c;  C3 = m2*l1*l2c-Ixz2;
C4 = l2c*m2*g;        C5 = I1zt+Ix2+m2*l1*l1;   C6 = C5*C1-C3*C3;

btavg = (btf + btr)/2;
bt = btavg;
```

```
% with Damping
DenB = [(C5*C1-C3*C3), (C1*btavg+C5*ba), (ba*btavg-C4*C5), (-C4*btavg), 0];
NumBT = [C1, ba, -C4];
NumBA = [-C3, 0, 0];

tfTB = tf(NumBT,DenB);
tfAB = tf(NumBA,DenB);

% without damping
Den = [(C5*C1-C3*C3), 0, -C4*C5, 0, 0];
NumT = [C1, 0, -C4];
NumA = [-C3, 0, 0];

tfT = tf(NumT,Den);
tfA = tf(NumA,Den);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% State Space
% where x = [theta; thetadot; alpha; alphadot];
% xdot = A*x + B*T

A = [  0,  1,              0,              0;
        0,  (-C1*btavg/C6), (-C3*C4/C6),    (C3*ba/C6);
        0,  0,              0,              1;
        0,  (C3*btavg/C6),  (C4*C5/C6),     (-C5*ba/C6)];

B = [0; C1/C6; 0; -C3/C6];

t = 0:.01:9.99;
C = [0 0 1 0];
D=0;

sys = ss(A,B,C,D);
x0 = [0 0 .1 0]';
u=zeros(size(t));

% 10 10 100 40 R = 5
Q = diag([10 40 120 60]);
R = 6;

K = lqr(A,B,Q,R)
K1 = K(1); K2 = K(2); K3 = K(3); K4 = K(4);

Ac = (A-B*K);
Bc = B;
Cc = C;
Dc = D;

sys_cl = ss(Ac,Bc,Cc,Dc);

% [y,x] = lsim(A,B,C,D, u, t, x0);
% plot(t,x)
```

```
% title('Uncontrolled');
% figure
% [Y,T,X] = lsim(sys_cl,u,t, x0);
% plot(T,Y);
% title('Controlled')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Other Controller Parameters
CONTROLLER_SWITCH_POINT = .30;  % switch over to balancing controller here
tau = .01;                      % time constant for derivative filter
```

System Energy Computation

```
function [energy, KE, PE]  = totalEnergy(theta, tdot, alpha, adot, Iz2,l2c,Iy2,l1,Ixz2,I1zt,Ix2,m2,g)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% compute potential energy of the system only cares about the pendulum
ca = cos(alpha); sa = sin(alpha);
ct = cos(theta); st = sin(theta);

PE = (ca - 1)*l2c*m2*g;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% compute the kinetic energy of the the system
thetadot = tdot;  alphadot = adot;

% rotational component for arm
KE1 = 0.5*I1zt*thetadot*thetadot;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% translational component for pendulum
J = zeros(3,2);

J(1,1) = -l2c*sa*ct - l1*st;
J(1,2) = -l2c*st*ca;

J(2,1) = -l2c*sa*st + l1*ct;
J(2,2) = l2c*ct*ca;

J(3,2) = -l2c*sa;

% now that we have the jacobian
V = J*[thetadot; alphadot];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
KE2T = 0.5*m2*(V'*V);

% rotaional component
KE2R = 0.5*(thetadot*thetadot*(Ix2*ca*ca + Iy2*sa*sa) - 2*Ixz2*thetadot*alphadot*ca ...
    + Iz2*alphadot*alphadot);
```

```
KE = KE1 + KE2T + KE2R;

energy = KE + PE;
```