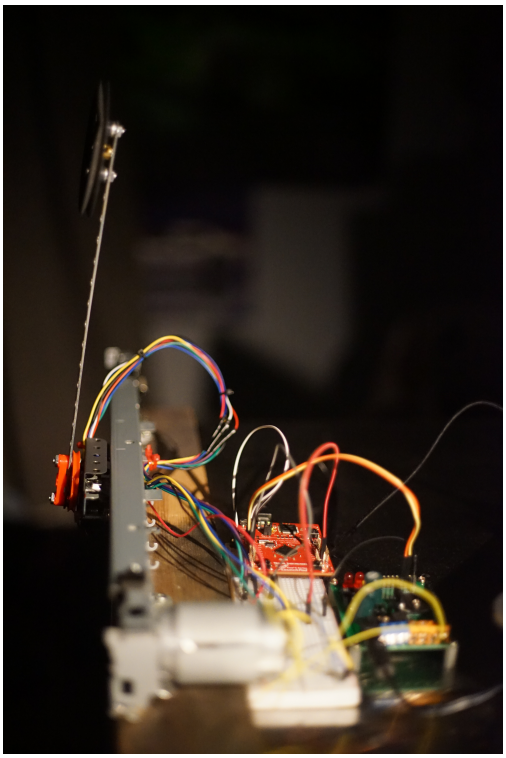
Real-Time Embedded State Feedback Control System for an Inverted Pendulum

MSE 483
Final Project Report

Project Group 4
Ryan Fielding - 301284210
Aarij Saigol - 301270438

April 15^{th}, 2020



Abstract

This report outlines the project completion for the MSE 483 (Modern Control Systems) course project, which has been developed in conjunction with the MSE 450 (Real-time Embedded Control Systems) course project. In short, the system implements a state feedback control system, written in **C**, via the course Tiva C TM4C123GH6PM microcontroller, with the goal of stabilizing an inverted pendulum across various small impulses or step inputs. This inverted pendulum system has been built from the parts of a broken Canon printer donated by a past high school teacher. The main components taken from this printer include the printer cartridge track (for the cart), a DC motor and belt drive, as well as a linear optical encoder. The pendulum is mounted to a potentiometer to measure the angle of the pendulum, θ, which will be mounted to the cart, whose position, x, will be measured (inc/decremented) by the optical encoder. To effectively implement this system, the Tiva C microcontroller initialization has been completed early on, in order to allow for state model controller testing and to ensure a stabilized pendulum by the project deadline. The microcontroller system will consist of:

- 2 digital inputs for the optical quadrature encoder outputs, channels A and B, which will increment and decrement the cart position via the quadrature encoder interface.
- 1 analog input for the angle of the pendulum via a 10kΩ potentiometer and ADC.
- 2 PWM outputs for the actuation and control of the 12V, 1A DC motor through an H-bridge (TA7267BP) interface board, connected to the cart via a belt drive.
- Switch 1 (SW1) digital input to trigger interrupts for running/stopping the system and recording target x, θ.

In addition to the physical development of this system, state space model analysis and modern control systems theory will be analyzed and implemented. To start, a basic dynamic analysis and state space model, as well as linearization about an operating point. Moving forward, a state feedback controller will be implemented, using a linear quadratic regulator to tune the gains. Following, a Luenberger observer based state feedback system will be implemented and attempted to complete by the project deadline. The system will be tested across many gains and pole placements.

In addition to the aforementioned goals and objectives, other features have also been implemented to further enhance this real time control system. Firstly, UART communication through USB to allow for data transmission and plotting of $x, \dot{x}, \theta, \dot{\theta}$ states of the system, in MATLAB. This enables the accurate realization of system response times to different inputs. Additionally, MATLAB functions are being used to place stable system poles, and tune the *Luenberger* observer. After tuning, another function exports an **obsv.h** file containing the system gains and matrices for observer feedback, which is then tested, plotted and analyzed. This process is currently in progress and is nearing completion. All development can be seen on *Github*, linked in the references.

Contents

1 Introduction **5**
- 1.1 Overview . . . 5
- 1.2 Modern Control System Summary . . . 5
- 1.3 Experimental Implementation Approach . . . 5
- 1.4 Expected Outcomes . . . 6

2 System Analysis **6**
- 2.1 Derivation . . . 6
  - 2.1.1 Recording Constants . . . 7
- 2.2 Dynamic Equations . . . 8
- 2.3 Linearization . . . 8
- 2.4 Resulting State Space Model . . . 8
  - 2.4.1 Controllability . . . 9
  - 2.4.2 Observability . . . 9

3 System Design **9**
- 3.1 Requirements . . . 9
  - 3.1.1 Nice to Have's . . . 10
- 3.2 State Feedback Control Model . . . 10
  - 3.2.1 Linear Quadratic Regulator Control . . . 11
  - 3.2.2 Luenberger Observer . . . 11

4 Implementation **13**
- 4.1 Hardware . . . 13
  - 4.1.1 Mechanical . . . 13
  - 4.1.2 Electrical . . . 14
  - 4.1.3 Completed System . . . 15
- 4.2 Software . . . 15
  - 4.2.1 Control System Flowchart . . . 16
  - 4.2.2 LQR Control Implementation . . . 16
  - 4.2.3 Luenberger Observer Implementation . . . 17

5 Results **18**
- 5.1 MATLAB Simulation Results . . . 18
  - 5.1.1 Impulse Responses . . . 18
  - 5.1.2 Step Response . . . 18
- 5.2 SIMULINK Models . . . 19
  - 5.2.1 Full State Feedback . . . 19
  - 5.2.2 Observer Based . . . 20
- 5.3 Experimental Results . . . 20
  - 5.3.1 Gain Tuning Process . . . 20
  - 5.3.2 Basic P Controller - PID Control . . . 21
  - 5.3.3 LQR Full State Feedback Control . . . 22

5.3.4 Observer Based State Feedback Control 24
5.4 Additional System Enhancements 24
5.4.1 Potentiometer Short Circuiting 24
5.4.2 Moving Average Filter for ADC Data 25
5.4.3 Quadrature Encoder Interface 25
5.4.4 UART Data Transmission to MATLAB 26

6 Conclusions **26**

A Timeline **29**

B State Calculation in C **29**

C Observer Function in C **30**

D Observer Header in C **30**

E UART Interrupt Handler in C **31**

List of Figures

1 System Concept - Inverted Pendulum on a Cart [1] 6
2 Measurement of System Constants 7
3 Full State Feedback Control Model [2] 10
4 Observer Based State Feedback Model [2] 12
5 DC Motor Belt and Feedback 13
6 System Setup Development 13
7 Custom Built Low Friction Linear Guided Cart 14
8 Circuit Drawing 14
9 Completed System Electronics 15
10 Built Inverted Pendulum System 15
11 Controller Flowchart 16
12 State Feedback System Impulse Response in MATLAB 18
13 Observer Based State Feedback System Step Response in MATLAB 19
14 SIMULINK State Feedback Controller Model 19
15 SIMULINK Observer Based State Feedback Controller Model 20
16 Tuning Process Flowchart 21
17 Physical System Response 21
18 Cart Oscillation 22
19 MATLAB Pole Placement via LQR 23
20 Continuous vs Discrete Time Pole Placement [3] 23
21 Final System Response to Disturbance Rejection 24
22 Faulty Connection Potentiometer Noise 25
23 ADC Signal Noise Moving Average Filtering 25
24 Quadrature Encoder Interface Signal 26

1 Introduction

1.1 Overview

This real time embedded control system involves the concept of stabilizing an inverted pendulum; an unstable system which if otherwise not intervened with by engineered, calculated actuation, will become unbalanced due to gravity and the displacement of the pendulum's centre of gravity from a set point. This experiment encapsulates a true real-time system which integrates an embedded microcontroller system to control and react within stringent response-time constraints.

Physically the system is built from reused spare parts of a broken Canon printer, donated from one of the team member's previous high school teachers. Parts salvaged include a linear guide rail, a 12V, 1A DC motor and a linear optical encoder. The linear encoder was later replaced as it was discovered to be burnt out. A basic potentiometer is be used to measure the angle of the pendulum. The platform chosen to control this system is the Tiva TM4C123GH6PM microcontroller. It is implemented with an H-bridge motor interface board to drive the DC motor, and the required sensors. Additionally, software development is done in Code Composer Studio written in C.

1.2 Modern Control System Summary

State Space Modelling is the selected means of control model implementation, ergo using the "notations of state, inputs, outputs and dynamics to describe the behavior of a system" [4]. From the perspective of an unstable system, state modelling allows us to predict the behavior of the system. Partner this with real time feedback, and one is able to implement a state feedback controller to achieve a desired objective. In our case, stabilization of an inverted pendulum. After an in depth system analysis and conversion to a state space model, controllability and observability will be assessed prior to the development of the system, to ensure feasibility.

To elaborate on the specifics, the control system will implement state feedback control. Moreover, since not all states of the system can be measured, some will have to be calculated, or estimated. To do this, first we will implement full state feedback control via calculation of immeasurable states. Following, the results of this process, a Luenberger observer based state feedback controller will be tested to assess it's estimation of immeasurable states, rather than calculation.

1.3 Experimental Implementation Approach

Overall, the system is composed of the following components:

Mechanical

- Cart linear guide rail from printer cartridge actuator
- DC motor and belt drive from printer cartridge actuator
- Custom built cart of spare mechanical parts
- Potentiometer mounted to cart

- Pendulum built from mechanical flange and spare parts at the top for weight

Electronics

- TM4C123GH6PM microcontroller
- 10kΩ potentiometer
- Linear optical quadrature encoder and optical strip
- Arduino development kit wires and breadboard for connections

1.4 Expected Outcomes

We expect to obtain a system response and from observing its characteristics previously explained, we plan to develop the system to have good disturbance rejection. This error responds to a system gains, however it is important to balance the increase of gain to the system as it impacts the sensitivity of the response and may additionally result in the increase of the maximum error and/or settling time [5]. A critically damped system with a medium gain is most desirable. This can be observed through an ideal physical system producing minimum oscillation about the set point to counteract the moment created by the movement of the pendulum's center of mass. From this experiment we also aim to form a solidified concept of embedded computer control system programming; designing and building a system, integrating hardware and software for an embedded control application, as well as in depth modern control systems understanding and practical implementation.

2 System Analysis

2.1 Derivation

The goal of this controller aims to maintain the pendulum in a state of inversion, that is, an angle of $\theta = 180^o$, constantly and for various impulse forces to the pendulum, by changing the position (x) of the cart. See figure 1 below for a diagram of the system and variables.

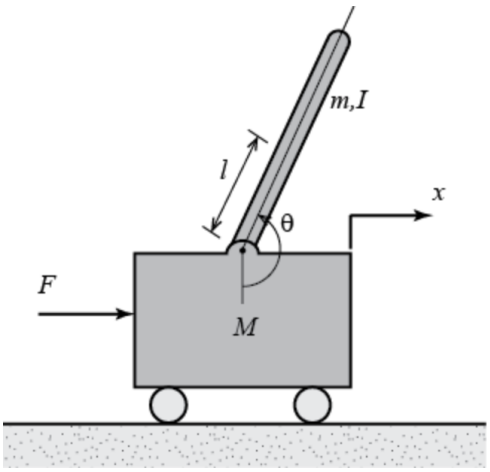
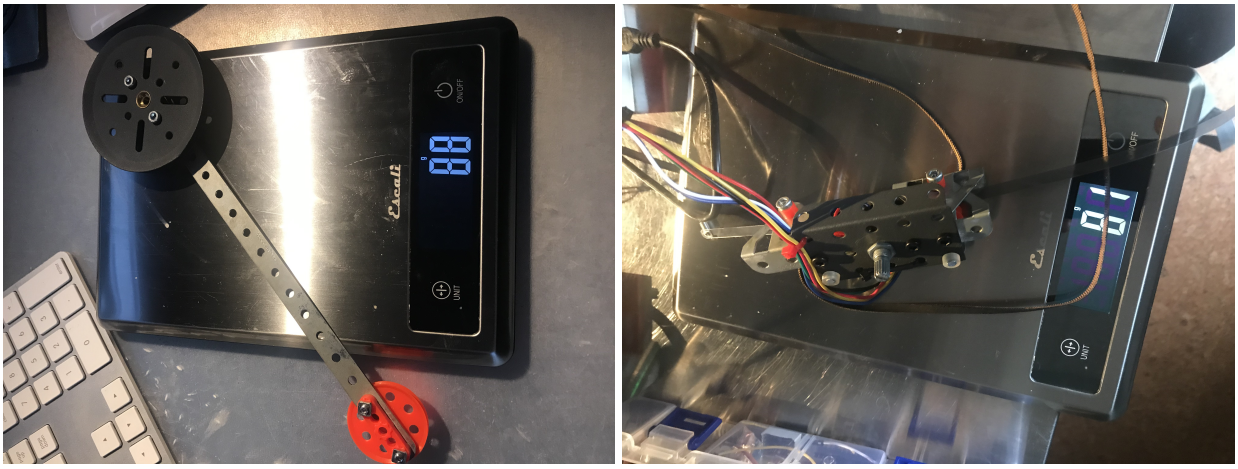


Figure 1: System Concept - Inverted Pendulum on a Cart [1]

2.1.1 Recording Constants

In order to tune the gains of our system effectively, an accurate simulation model had to be created in MATLAB. For this to happen, all characteristics of the system, including the pendulum mass, inertia, cart mass, and more, had to be measured. In the case of cart friction, estimated.



(a) Pendulum Mass (b) Cart Mass

Figure 2: Measurement of System Constants

All resulting MATLAB code can be seen in the *model.m* file on Github [6]. In order to make the derivation and linearization of the model, the variables for the model were defined as shown below:

Model Variables

- M mass of the cart = 0.081kg
- m mass of the pendulum = 0.088kg
- b Cart friction estimate = 0.1N/m/s
- g gravity coefficient = $9.81m/s^2$
- l length of the pendulum = 0.10m

To add weight to the pendulum, a disk has been attached to the top. Without this weight, the pendulum is too light to overcome the static friction of the potentiometer, hence it remains in the same position unless the cart is moved repidly. In order to calculate the inertia of the pendulum, the following equation and variables were used.

- m_{rod} mass of the rod = 0.035kg
- m_{disk} mass of the disk = 0.053kg
- r_{disk} radius of the disk = 0.037m

$$I = (1/3) * m_{rod} * l^2 + 0.5 * m_{disk} * r_{dis} - k^2 + m_{disk} * (2 * l + r_{disk})^2 \quad (1)$$

2.2 Dynamic Equations

After summing up all the forces acting on the cart and the pendulum in figure 4, the dynamic equation are shown below:

$$(M+m)\ddot{x} + bx + ml\ddot{\theta}cos(\theta) - ml(\dot{\theta})^2 sin(\theta) = F \tag{2}$$

$$(I+ml^2)\ddot{\theta} + mglsin\theta = -ml\ddot{x}cos\theta \tag{3}$$

2.3 Linearization

As the system to which the equations will be applied is linear, so the equations need to be linearized. The equations are linearized about vertical equilibrium position and assuming small deviation from the equilibrium position, small angle approximations were made for the nonlinear function for the system.

$$\theta \approx \pi, \phi = \pi - \phi \tag{4}$$

$$cos(\theta) = cos(\pi + \phi) \approx -1 \tag{5}$$

$$sin(\theta) = sin(\pi + \phi) \approx -\phi \tag{6}$$

$$\dot{\theta} = \dot{\phi} \approx 0 \tag{7}$$

The equation for the model after linearization is shown below

$$(M+m)\ddot{x} + bx + ml\ddot{\theta} = u \tag{8}$$

$$(I+ml^2)\ddot{\theta} + mgl\theta = -ml\ddot{x} \tag{9}$$

2.4 Resulting State Space Model

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -(I+ml^2)b/p & m^2gl^2/p & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -(mlb)/p & mgl(M+m)/p & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ (I+ml^2)/p \\ 0 \\ ml/p \end{bmatrix} \text{u}$$

$$\text{y} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \text{u}$$

$$\text{Where } p = I*(M+m) + M*m*l^2 \tag{10}$$

The variables value for our model were plugged into the state space and the resultant matrix is shown below

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -0.6681 & 1.2657 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -1.4661 & 24.31 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ 5.215 \\ 0 \\ 14.66 \end{bmatrix} \text{u}$$

$$\text{y} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \text{u}$$

2.4.1 Controllability

Controllability is the ability of the controller to randomly change the function of the system. To calculate controllability the equation used is shown below

$$P = [B \; AB \; ..... \; A^{n-1}B] \tag{11}$$

The controllability of the system was checked in MATLAB with the formula **ctrb**(A,B), resulting in a value of 4, implying that *P* is full rank, and thus the system is controllable.

2.4.2 Observability

Observability is the ability of the system to decide if the state variable of the system can be measured from the external outputs. To calculate the the observability the equation used is shown below

$$Q = \begin{bmatrix} C \\ CA \\ . \\ . \\ CA^{n-1} \end{bmatrix} \tag{12}$$

The observability of the system was checked in MATLAB with the formula **obsv**(C,A), resulting in a value of 4, implying the system is also observable.

3 System Design

3.1 Requirements

The requirements for this problem are short and sweet - to balance the pendulum vertically as opposed to letting it fall over due to gravity. To elaborate on these requirements, we can set the following parameters:

- $\Delta\theta < 5^o$ for small impulses to the pendulum or cart.
- Settling time of < 4 seconds.
- Cart position should be settled consistently at a constant reference location, ie. $x = 0cm$.

Further requirements may be set for the *Nice to Have's* outlined in section 3.1.1, depending on the project's completion schedule. Assuming all prior requirements are complete, we will investigate said problems and construct design requirements.

3.1.1 Nice to Have's

In addition the the objectives outline in the previous section, there are many sub-features that can be developed in the software of this system that present complex control theory problems, 2 of which are outlined below.

- Side-stepping of cart to a desired position ensuring constant pendulum stability.
  - For example, move cart from $x = 0cm$ to $x = 10cm$ with minimal overshoot, rise time, steady state error.
- Swing-up and swing-down commands
  - Enables the system to move the cart from one state of stability to another, ie. from $\theta = 0^o$ to $\theta = 180^o$ and stabilize, or vice versa with minimum oscillation.

3.2 State Feedback Control Model

The following control model will be implemented assuming full state feedback.

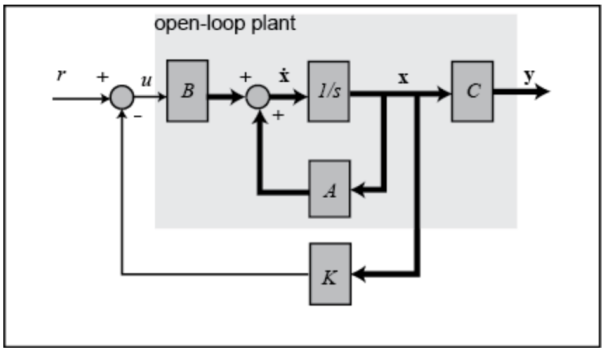


Figure 3: Full State Feedback Control Model [2]

The following equation represents the controller output *u*, which effectively sets the duty cycle of the motor PWM signal, can be seen below.

$$u = -Kx + r \tag{13}$$

Where the values of K will be tuned in MATLAB. In order to model our system accurately in MATLAB, each constant of the system must be measured and recorded (see section 2.1.1). For example,

the cart weight, pendulum inertia, length and more have significant impacts on the modelling of the system, including the controllability and observability. Once these values are recorded, MATLAB will also be used to tune the gain vector *L*, also known as the Luenberger observer, based off the closed loop system poles.

3.2.1 Linear Quadratic Regulator Control

Following the initial implementation of this control system, a more enhanced linear quadratic regulator controller will be implemented. This system, though similar in practicality to a PID controller, employs a more optimal approach to gain tuning. Essentially, gains "are found by using a mathematical algorithm that minimizes a cost function with weighting factors supplied by a human (engineer)" [7]. Said function to minimize, for a finite horizon discrete time system, is shown below.

$$J = x_N^T Q x_N + \sum_{k=0}^{N-1} \left(x_k^T Q x_k + u_k^T R u_k + 2 x_k^T N u_k \right) \tag{14}$$

The performance index *J* is to be minimized. By tuning values in Q and the value of R, the gains and closed loop system poles are optimized according to these requirements. To achieve this function in MATLAB, the **dlqr** command will be used for implementation with a discrete time system.

3.2.2 Luenberger Observer

Physical systems possess many non-linearities and thus more practical models and tuning will need to be implemented. Such as the problem of the nonlinear control of an inverted pendulum, which sees to balance the pendulum from an unstable equilibrium position. Such control methods include Linear Quadratic Regulation (LQR), observer based state feedback control and more. The following flowchart displays the proposed observer based state feedback control model, implementing an observer due to the fact that the system does not provide full state feedback, which would require the purchase of 2 tachometers to measure the rate of change of θ, x, that is, $\dot{\theta}, \dot{x}$. However, we will be implementing a state estimator to estimate the values of these states instead.

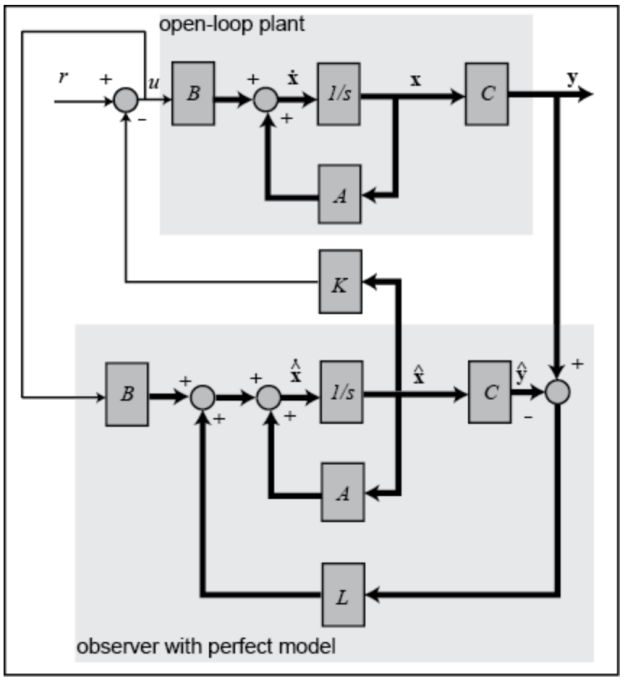


Figure 4: Observer Based State Feedback Model [2]

The above control system will not be elaborated too in depth as it is out of the scope of this report, however it presents the overall concept. The most important variables and what they represent are discussed in the following:

- x the states of the system - $x, \dot{x}, \theta, \dot{\theta}$, measured and estimated through observer based state feedback.
- y the demanded outputs of the controller, x, θ
- u the inputs to the system, which would be the force to the cart, F.
- K, L, the gain vector and Luenberger observer vector, designed and tuned in MATLAB.
- All other matrices, A, B, C, D and so forth, represent the complete state model for the linearized system, full form shown below:

$$\dot{x} = Ax + Bu \text{ and } y = Cx + Du \tag{15}$$

And for the state estimator:

$$\dot{\hat{x}} = A\hat{x} + Bu + L(y - \hat{y}) \text{ and } \dot{e} = \dot{x} - \dot{\hat{x}} = ... \implies \dot{e} = (A - LC)e \tag{16}$$

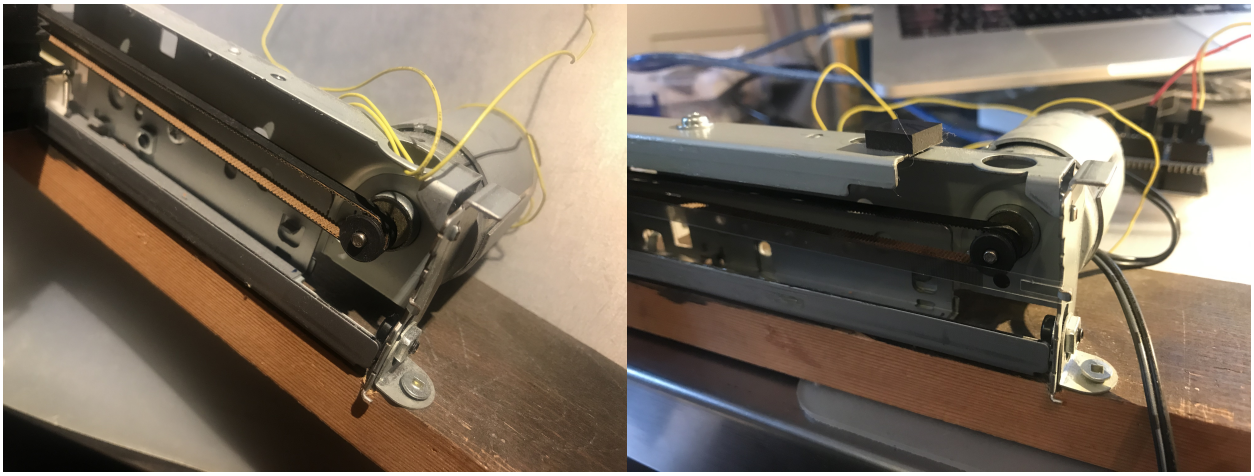
Who's poles will be designed as 5-10 times faster than that of the LQR controller, so as to drive $e \implies 0$ quickly and in a stable manner.

4 Implementation

4.1 Hardware

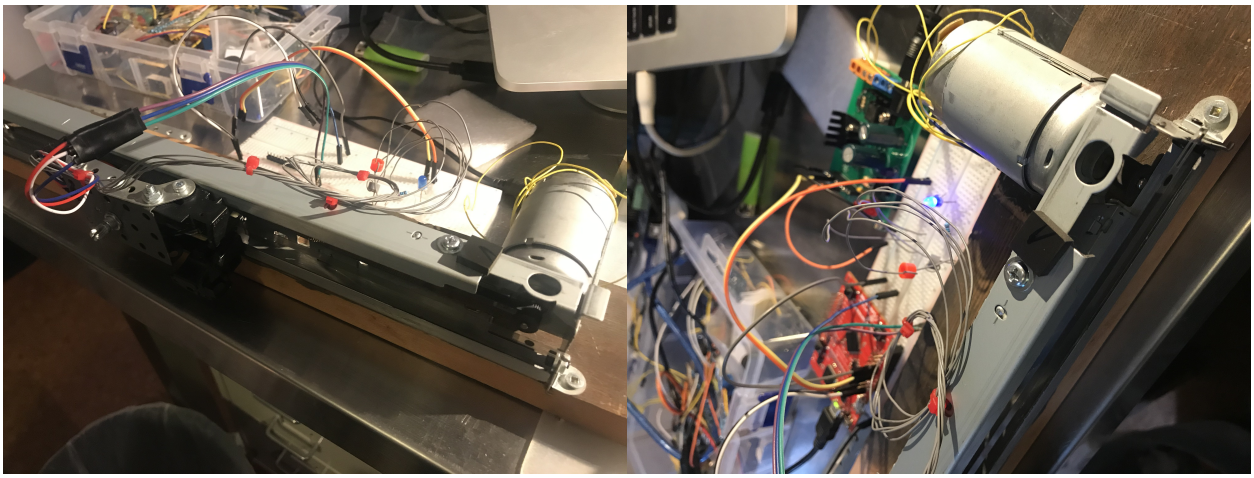
4.1.1 Mechanical

As for the hardware of the inverted pendulum system, the re-purposed Canon printer can be seen throughout the following figures.



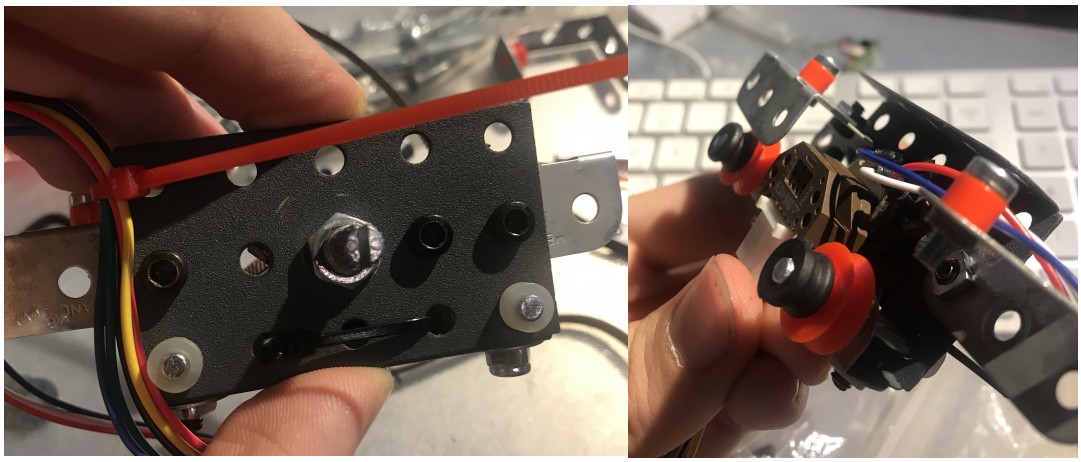
(a) DC Motor Belt Drive (b) Optical Encoder Linear Strip

Figure 5: DC Motor Belt and Feedback



(a) System Setup with Wiring (b) System Electronics

Figure 6: System Setup Development



(a) Potentiometer on Cart (b) Optical Encoder on Cart

Figure 7: Custom Built Low Friction Linear Guided Cart

4.1.2 Electrical

The Tiva TM4C123GH6PM microcontroller from Texas Instruments will be used as the real-time embedded controller. Code Composer Studio 9.3.0 for Mac OS X will be used to program and debug the board. A basic breadboard will be used for all wiring connections, and the course H-Bridge motor driver interface board will be used for PWM 12VDC motor control. A circuit drawing can be seen below, where V_{CC} = 3.3V.

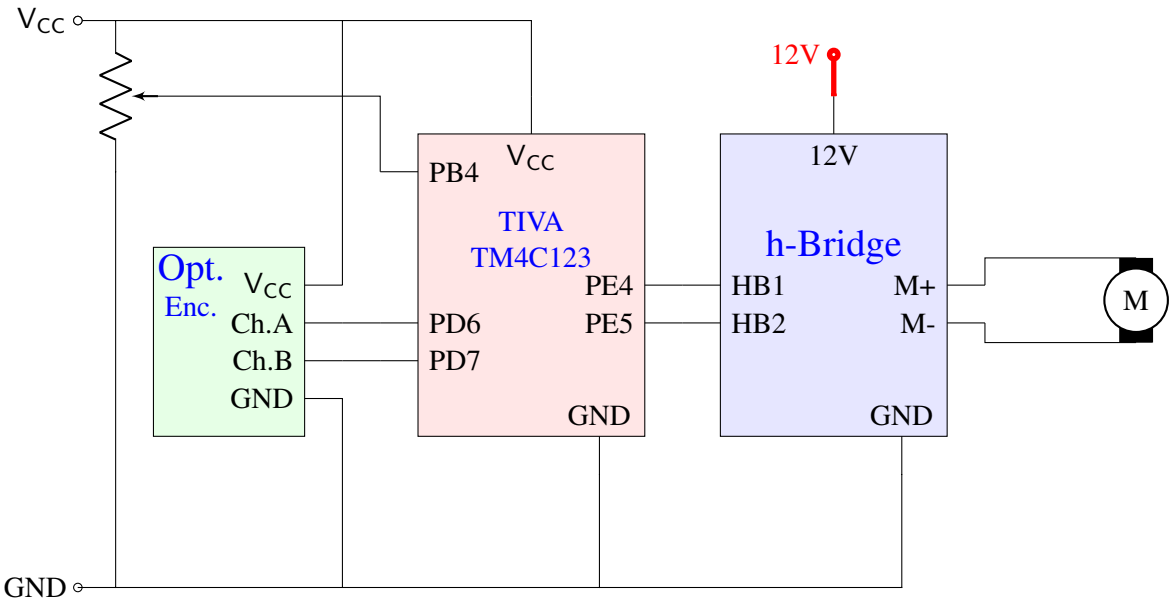


Figure 8: Circuit Drawing

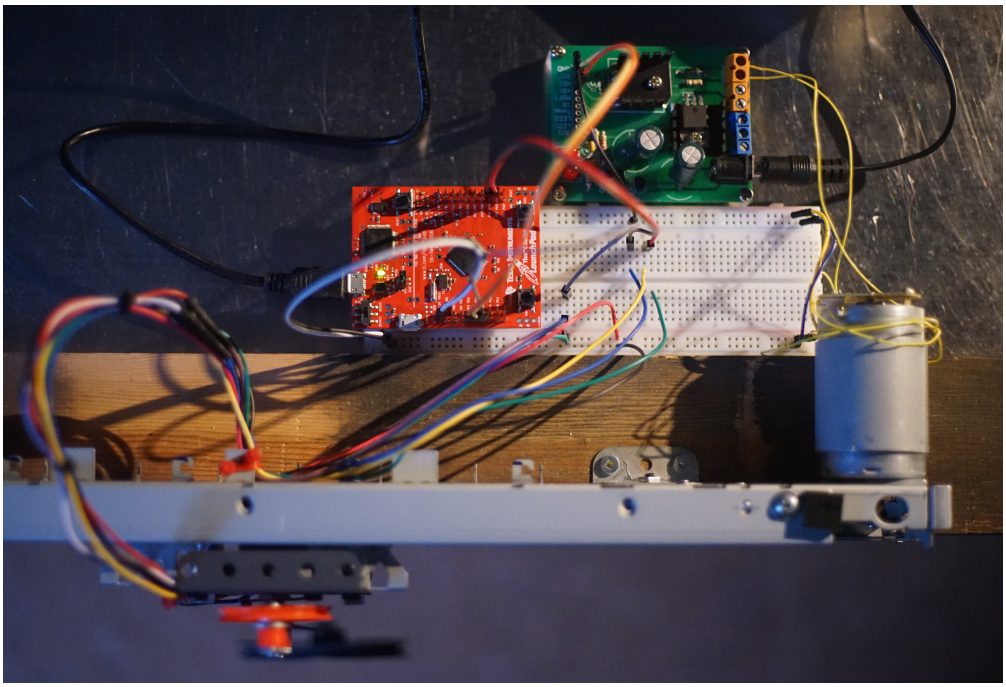
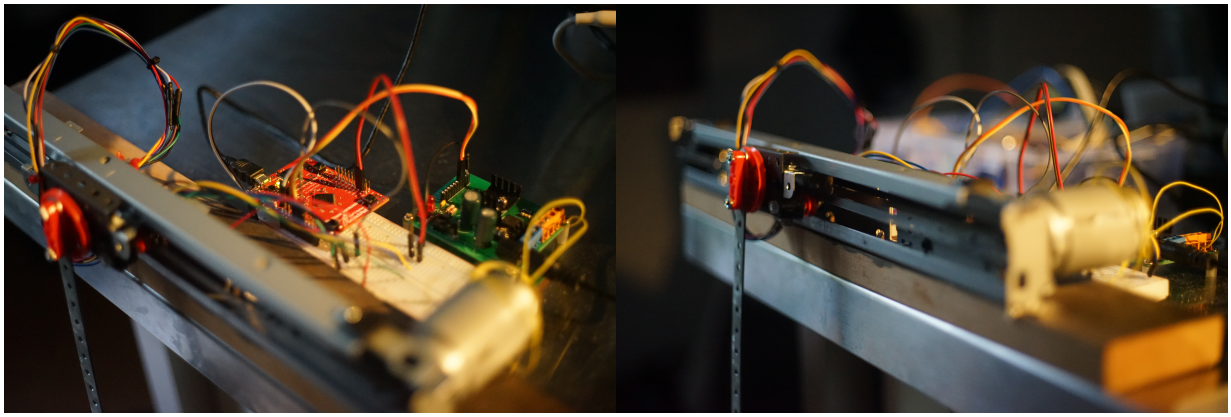


Figure 9: Completed System Electronics

4.1.3 Completed System



(a) Complete System (b) Linear Cart with Optical Strip

Figure 10: Built Inverted Pendulum System

4.2 Software

To develop the state space model analysis, software tools MATLAB and SIMULINK will be used. MATLAB to develop the system state space model, test and implement state feedback control, and SIMULINK to simulate and observe the system, as well as fine tune the controller gains. Furthermore, the controllability and observability of the system will be assessed in MATLAB.

4.2.1 Control System Flowchart

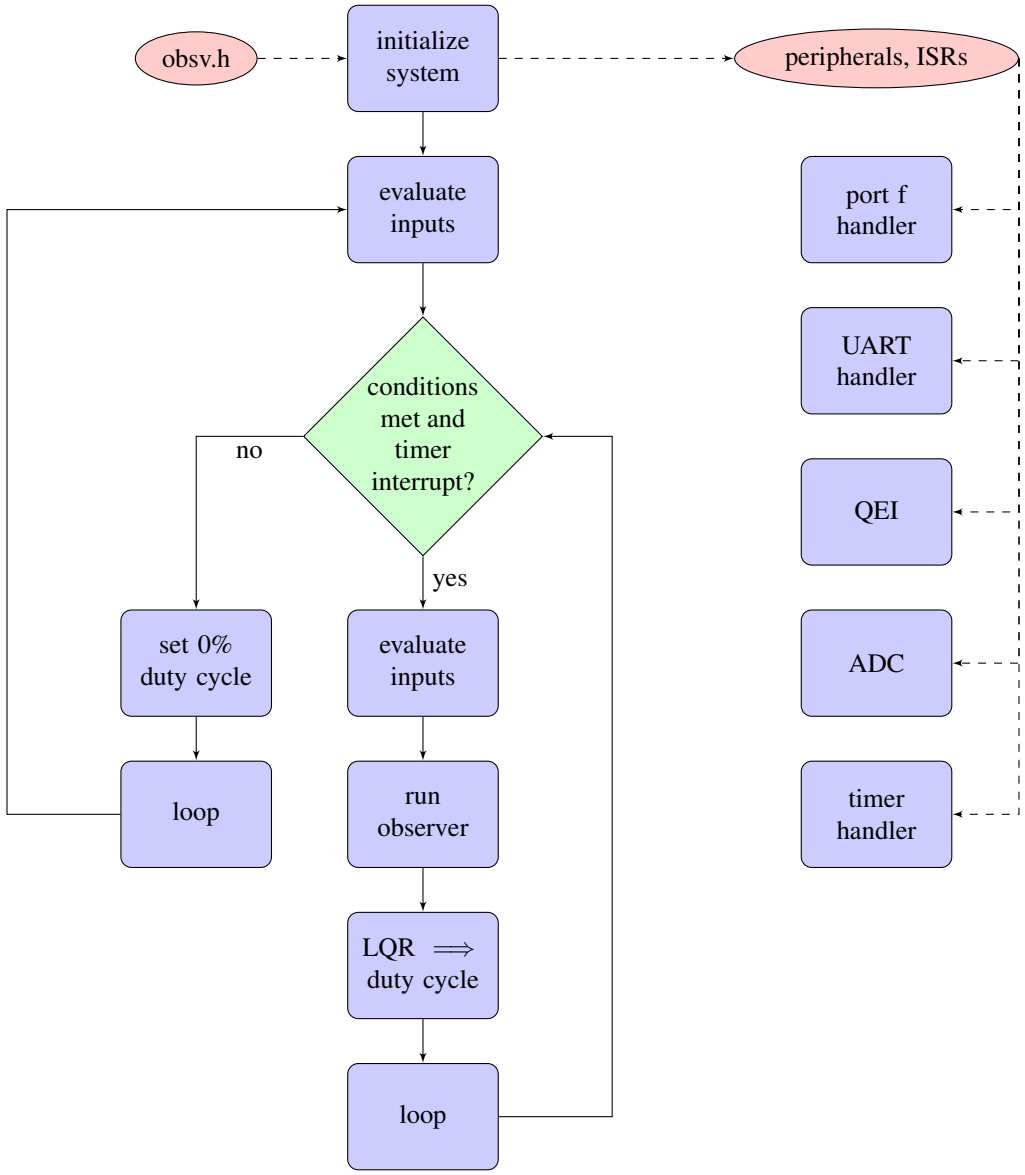


Figure 11: Controller Flowchart

4.2.2 LQR Control Implementation

In order to effectively run the full state feedback controller, calculation of u must be done at a specified frequency, rather than at the variable CPU crystal frequency of 80MHz, which is far too fast. The algorithm can be seen below, with exponential filters applied to smooth the noisy data and derivatives.

Algorithm 1: Full State Calculation Algorithm

Input : Measured states, x
Output: Estimated states, $\hat{x}$
Calculates $\dot{x}$ and $\dot{\theta}$ based off measurements of x, θ and exponential smoothing
rPosDot = 0.05
rThetaDot = 0.02
rTheta = 0.15

$\hat{x}.X = (x - x_{ref})scale_x$
$\hat{x}.Y = (1 - rPosDot)\dot{x}_{k-1} + rPosDot(\hat{x}.X - x_{k-1})/dt$
$\hat{x}.Z = (1 - rTheta)\theta_{k-1} + rTheta(\theta_{max} - readADC() - \theta_{ref})scale_\theta$
$\hat{x}.W = (1 - rThetaDot)\dot{\theta}_{k-1} + rThetaDot(\hat{x}.Z - \theta_{k-1})/dt$
$x_{k-1} = \hat{x}.X$
$\dot{x_{k-1}} = \hat{x}.Y$
$\theta_{k-1} = \hat{x}.Z$
$\dot{\theta_{k-1}} = \hat{x}.W$

This algorithm implementation in C can be seen in appendix B. Following this calculation, the duty cycle of the motor is set via this equation.

$$u = -Kx \implies dc = -\alpha_{dc} K\hat{x} \tag{17}$$

Where α_{dc} is a tuned constant that scales the input *u* to the required duty cycle levels, which range from -50000 to 50000.

4.2.3 Luenberger Observer Implementation

To elaborate on our control model approach shown in figure 4, the following algorithm will be implemented in C to estimate the immeasurable states, $\dot{x}, \dot{\theta}$, the velocity of the cart and the rotational velocity of the pendulum.

Algorithm 2: Luenberger Observer Algorithm

Input : Measured states, x
Output: Estimated states, $\hat{x}$
Update observer state estimator global variables $\dot{x}, \dot{\theta}$
Based off measurements of x, θ

$dt = 1/CtrlFreq$
$y = Cx$
while $\|e\| \geq \tau$ **do**
    $\hat{y} = C\hat{x}$
    $e = y - \hat{y}$
    $\dot{\hat{x}} = (A - BK)\hat{x} + Le$
    $\hat{x_n} = \hat{x} + \dot{\hat{x}}dt$
    $\hat{x} = \hat{x_n}$
end

This algorithm implementation in C can be seen in appendix C.

5 Results

5.1 MATLAB Simulation Results

After completing the state model for the linearized system, the model was then implemented and simulated in MATLAB. This allows for software-in-the-loop testing of the system, without the concerns of breaking hardware, sensor noise, non-linearities and more. However, this can pose as a risk as the system simulation accuracy cannot be exact, due to said non-linearities that only exist in a physical system, such as friction, motor back EMF, and more.

5.1.1 Impulse Responses

An example plot of the system's impulse response can be seen below. This plot was output upon every tune of the gain matrix K to ensure stable disturbance rejection.

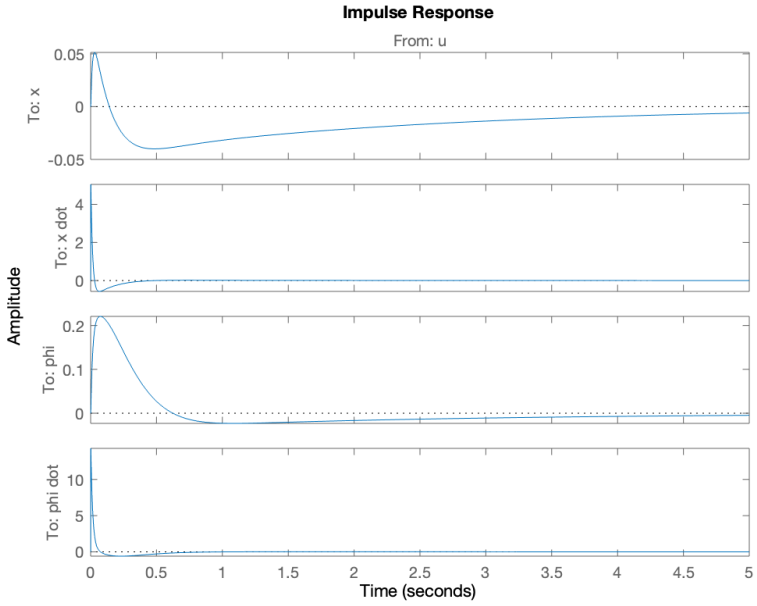


Figure 12: State Feedback System Impulse Response in MATLAB

5.1.2 Step Response

Figure 15 shows an example of the simulated discrete time system response for a cart position step input of *20cm*.

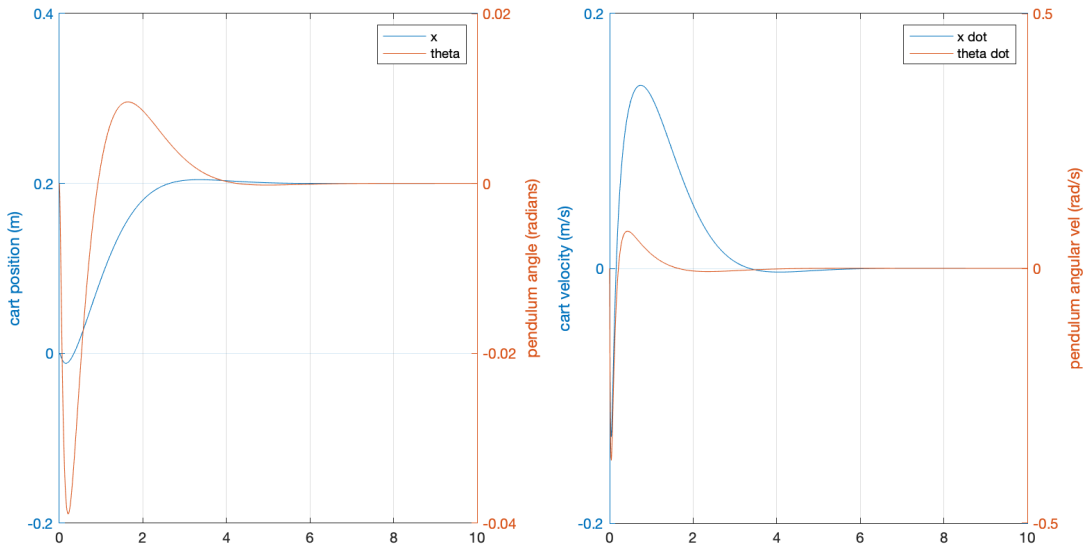


Figure 13: Observer Based State Feedback System Step Response in MATLAB

5.2 SIMULINK Models

SIMULINK was also used to build simulation models, in addition to MATLAB's functions. This was to ensure feasible testing of tuning prior to implementation on the physical system.

5.2.1 Full State Feedback

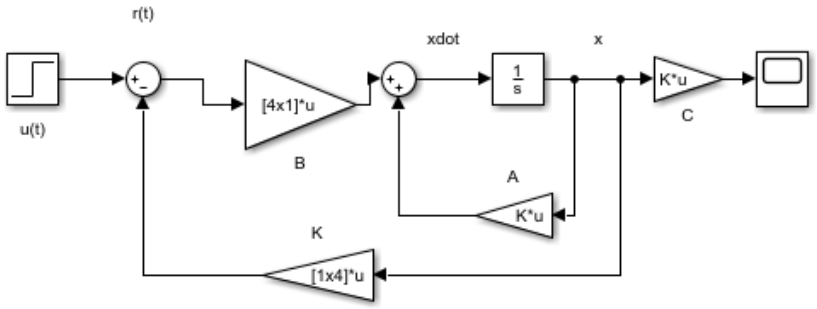


Figure 14: SIMULINK State Feedback Controller Model

5.2.2 Observer Based

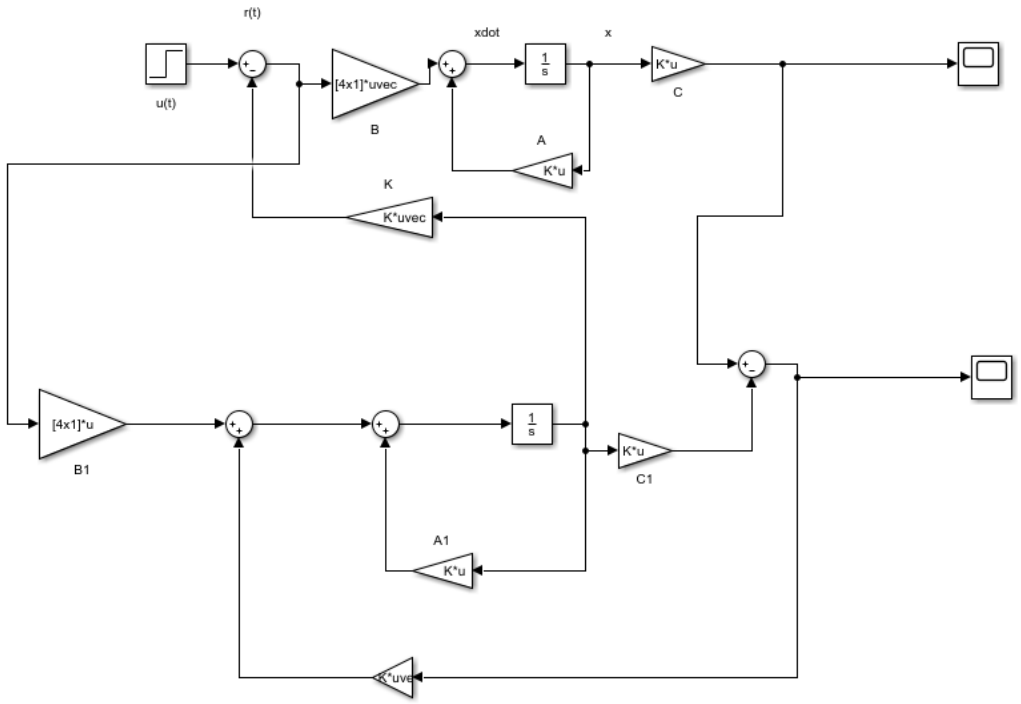


Figure 15: SIMULINK Observer Based State Feedback Controller Model

5.3 Experimental Results

Overall, the physical system is very controllable. Adequate sensors and a powerful motor enable the robust, real time embedded control of the inverted pendulum system. However, there are limitations to tuning the physical system based off a state space model of the system whose variables, though measured, may not account for all the non-linearities that exist in a practical, physical control system. Nonetheless, the following sections discuss the overall system tuning results. Responses have been plotted in MATLAB through serial UART communication with the tiva microcontroller for a more in depth analysis.

5.3.1 Gain Tuning Process

Once the system was completely setup and tested with PID control, development and testing of the observer had to be streamlined. Rather than tuning in MATLAB, copying and pasting system matrices, a function was developed in MATLAB to export the *obsv.h* file, containing said constants. This file can be seen in appendix D.

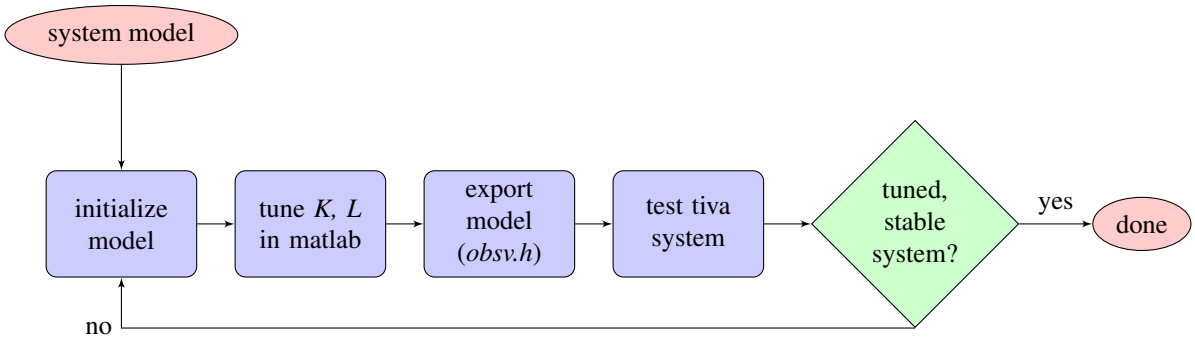


Figure 16: Tuning Process Flowchart

5.3.2 Basic P Controller - PID Control

To observe functioning and successful system integration, a basic P controller was first implemented. Figure 17 displays the measured states x, θ, of the system through UART communication with MATLAB as the pendulum responds to minor impulses. Let it be noted that a second impulse returned the cart to the reference positon, $x = 0cm$.

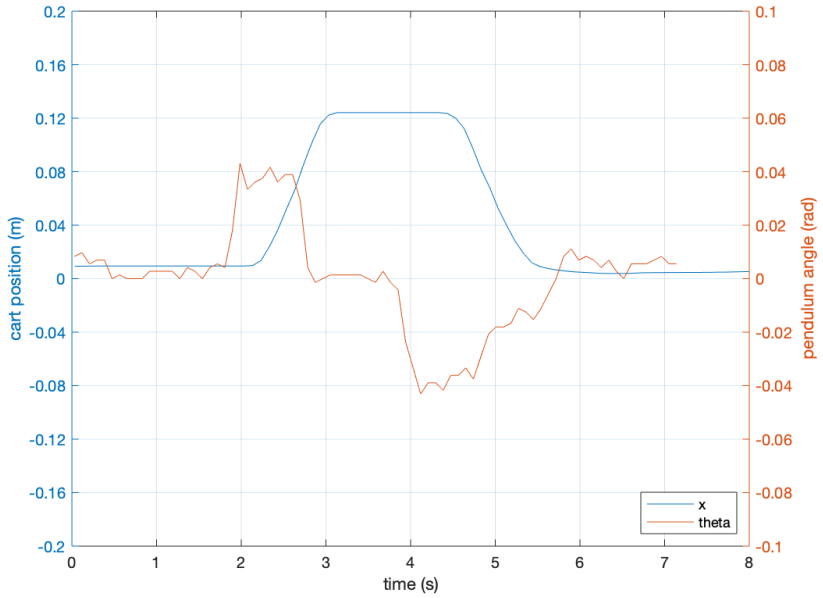


Figure 17: Physical System Response

It can be seen above that as the pendulum angle moves further from 0, the cart's position is adjusted by the motor in order to ensure the pendulum remains stable. At most, the pendulum angle moves 0.08 radians, approximately 4.58^o, which satisfies the controller requirements. Moreover, a settling time of ≤ 4 seconds is clear, however there exists steady state error with respect to the cart's position. After a quick analysis of controllability of an inverted pendulum using PID control, it can be seen that there will always exist steady state error for the cart position [8], which will also be very unstable, hence the demand for a state feedback type controller.

After the manual tuning method was implemented, the system ended up with a relatively satisfactory proportional gain for a P controller. As the gain for P (K_P) increases, the most observable effects are the decrease in rise time, increase in overshoot, decrease in steady state error, and a degrade in the stability of the system with an overall small change in settling time. After the manual manipulation of K_P, the settled upon gain after a few attempts was a value of 800.

5.3.3 LQR Full State Feedback Control

After initial implementation of a state feedback controller, tuned via LQR methods in MATLAB, it was found that the pendulum was quite stable, however the cart would easily oscillate about it's set point (reference) upon any disturbances, even just sensor noise would cause it to oscillate. Figure 18 displays the cart's position in red, upon a minor disturbance to the pendulum.

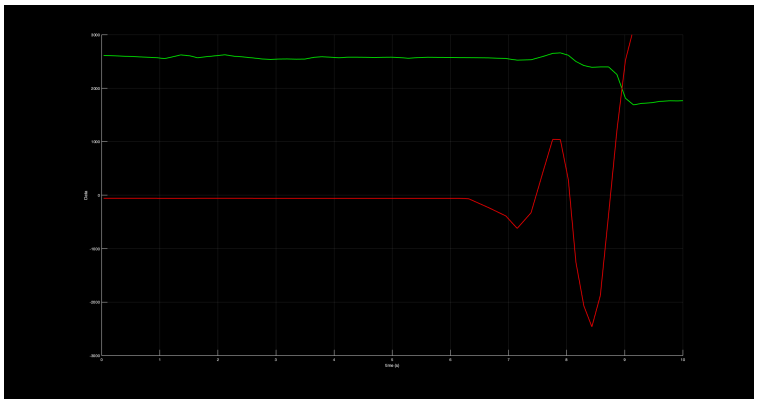
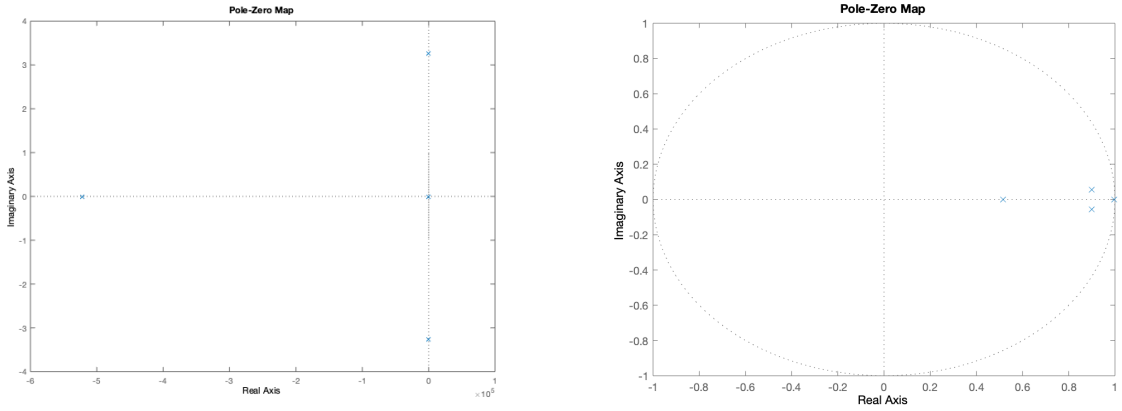


Figure 18: Cart Oscillation

While tuning the system in MATLAB using the **lqr** function, it was found that the system tuning for a continuous time system, while in reality our physical system is implemented on a microprocessor, hence it is a discrete time system. Thus, use of the **dlqr** command on the state model improved placement of the closed loop poles to within the unit circle or the s-plane.



(a) Pole Placement with **lqr**

(b) Pole Placement with **dlqr**

Figure 19: MATLAB Pole Placement via LQR

As seen in figure 19, the continuous time system on the left has an incredibly aggressive pole below $-5x10^5$, whereas the digital tuning has practical poles. Physically, this resulted in a much more stable system. Minor impulses to the cart did not have as great of an effect as before. This is because pole placement theory is different for continuous and discrete time cases, as shown in the figure below.

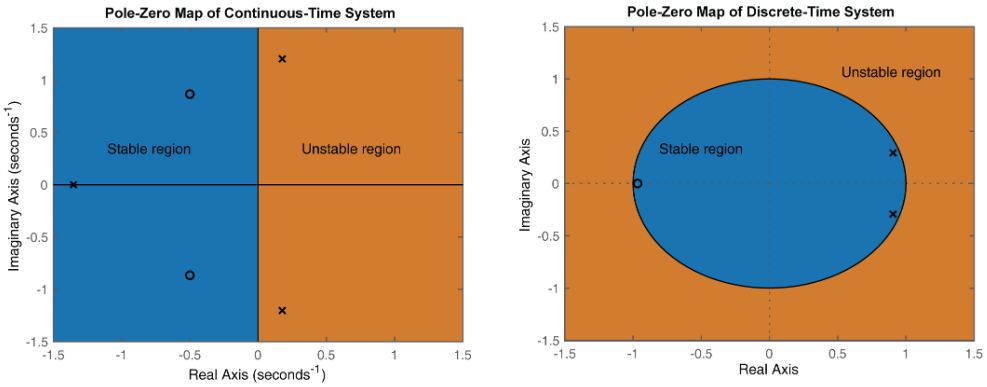


Figure 20: Continuous vs Discrete Time Pole Placement [3]

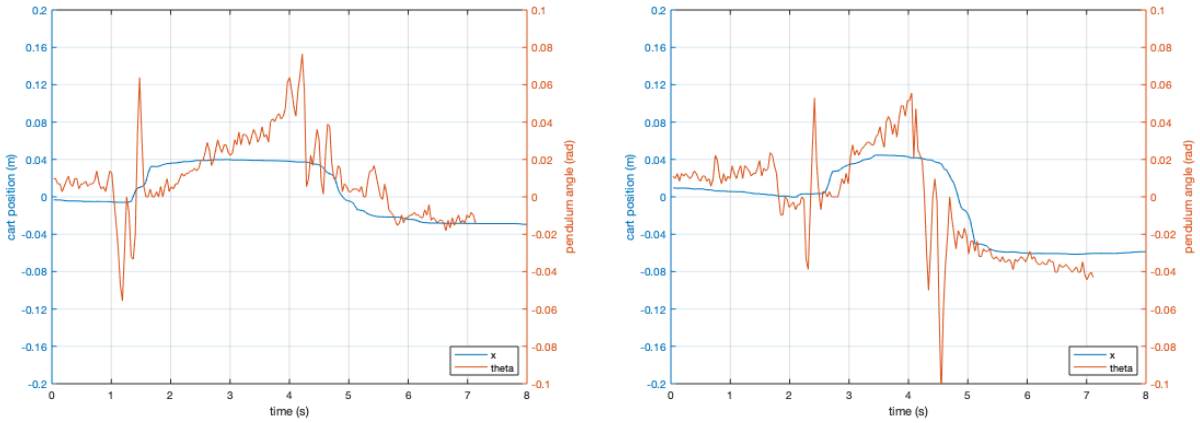
Testing with MATLAB functions **place** and **acker** proved to result in incredibly large values of K, that also placed far too much control effort on the cart position x, hence these commands could not be used. After some testing with the **dlqr** function, the following optimal values were obtained for K.

$$Q = \begin{bmatrix} 1000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 5000 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, R = 1 \implies K = [-27.7730, -9.4171, 66.4014, 5.0086] \quad (18)$$

However, these values for K are still not completely optimal, but are a great base for K. Even still, the cart seems to oscillate about it's reference point, while keeping the pendulum inverted,

similar to that of figure 18. After some manual tuning of K, the following values optimized the performance of the system.

$$K = [-25.0, -20.0, 40.0, 5.0] \tag{19}$$



(a) Decent Tune

(b) Better Tune

Figure 21: Final System Response to Disturbance Rejection

However, there is still too much oscillation of the cart. This is due to the gain k_2, which amplifies the cart velocity, $\dot{x}$, hence it was turned down just enough. The response can be seen above in figure 21b.

$$K = [-25.0, -18.0, 40.0, 5.0] \tag{20}$$

Clearly, the cart did not finish at it's reference position of $x = 0cm$, instead it shot past by $6cm$. Steady state error can be eliminated by adding a pre-compensator, an integrator, or a Luenberger observer, which is discussed in the following section.

5.3.4 Observer Based State Feedback Control

Unfortunately due to project time constraints, the observer was never properly implemented. After hours of testing, gains could not be tuned to drive $e \implies 0$. Thus, the system was very unstable. Tuning was done in MATLAB, with placement of observer poles 5 times faster than the system closed loop poles. All implementation is present, however a proper *L* must be found to achieve a functioning observer.

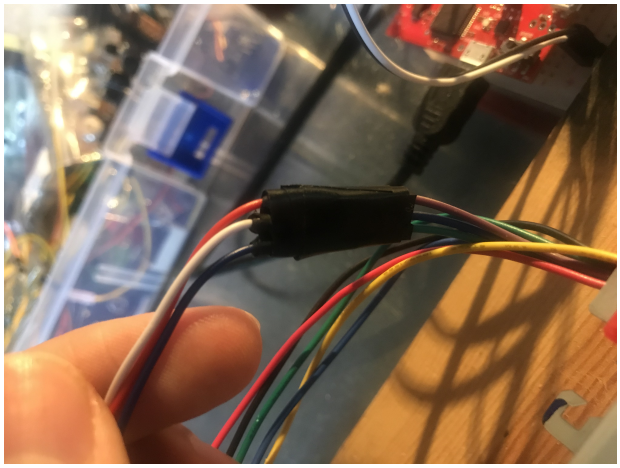
5.4 Additional System Enhancements

This section outlines the modifications that were made to the system to solve problems that were discovered throughout the development of the project.

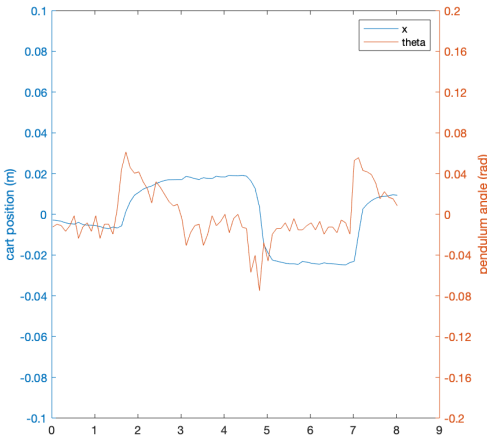
5.4.1 Potentiometer Short Circuiting

After tireless troubleshooting it was found that there was a faulty connection in the wiring for the potentiometer. The following figures display the noise induced by this error, and it's effect on the

system performance. It can be seen that the cart was incredibly 'jumpy' prior to the re-wiring of the sensor.



(a) Faulty Connection

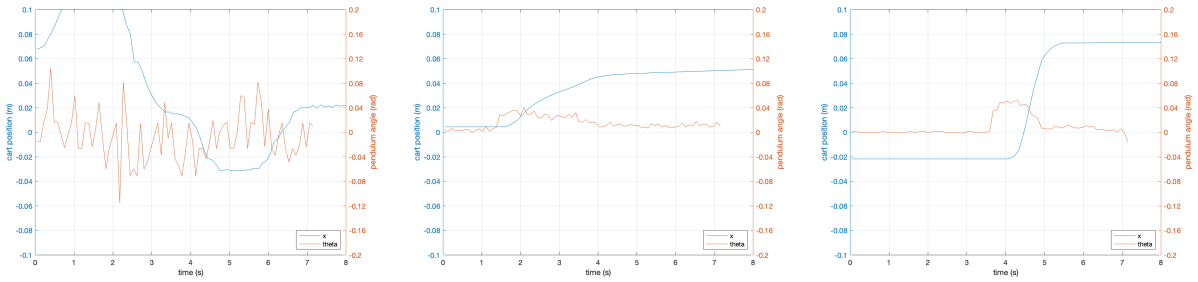


(b) Resulting Noise

Figure 22: Faulty Connection Potentiometer Noise

5.4.2 Moving Average Filter for ADC Data

When using an analog to digital converter, there is often a large amount of signal noise present, especially when using a cheap Arduino kit potentiometer. Thus, signal filtering is required. Rather than implementing an RC filter and delaying the signal with too large of a capacitor, we implemented a simple moving average filter. The following images display the before and after signal conditioning results.



(a) No Filter (b) Moving Average, Length 10 (c) Moving Average, Length 25

Figure 23: ADC Signal Noise Moving Average Filtering

5.4.3 Quadrature Encoder Interface

The Quadrature Encoder Interface (QEI) module provides an interface for which incremental encoders - such as the one used in this project- is able to obtain the mechanical position of the data [9]. Below is an image which illustrates the signals produced by a QEI (figure 24). These encoders are

also known as optical encoders and characteristically have two signals which are out of phase (one leading and one lagging). Since the digital interrupt protocol for QEI outputs were not fast enough to increase or decrease the cart's position, the on board QEI peripheral was implemented. This was able to essentially read the cart's position separate from the main CPU.

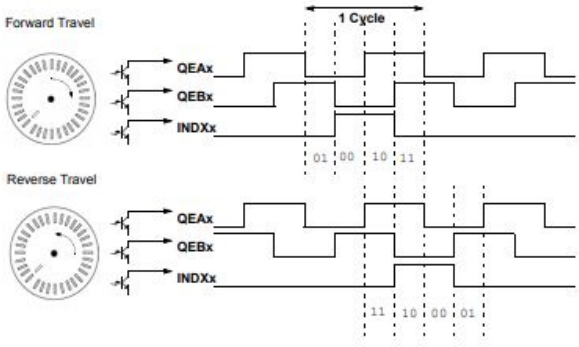


Figure 24: Quadrature Encoder Interface Signal

5.4.4 UART Data Transmission to MATLAB

Rather than using Code Composer Studio's low quality graphical interface, that does not allow for recording of data, plot editing and more, UART communication was implemented on the board to enable plotting and analysis in MATLAB. Upon interrupt trigger, the board sends the current readings for x and θ, rather than constantly sending the data, even if no devices are listening. After a few lines of code, a loop allows MATLAB to poll these values from the board over a set period, convert them from hex to decimal, and plot. Code for the UART interrupt handler function can be seen in appendix E.

6 Conclusions

Overall, this system is completely feasible, controllable and observable. Within this control system design project we were able to provide thorough testing that was completed with all proposed components and ensured satisfactory working requirements to implement the state feedback controller. Issues that arose included the mechanical development and construction of the system requiring the complete deconstruction of the cart to be rebuilt in order create smoother movement. Furthermore, the optical encoder needed to be replaced as the old component salvaged from the printer was burnt out. Difficulties with protocol implementation included UART and PWM. UART was difficult to implement at first but was soon remedied after further research, and the PWM load cycle took a while to integrate during our experimentation but was able to run the motor successfully.

Hours were spent tuning the system using MATLAB's **dlqr** function, which proved to produce a viable starting point *K*. Finely tuning these gains enabled the system to respond quicker, and oscillate less. As for the implementation of the Luenberger observer state feedback controller, the tuning is nearing completion. Once this is completed, the system can be further tuned for faster response time, 0 steady state error and more. State space modelling in MATLAB has enabled the

proper assessment of controllability and observability of the system, as well as tuning of the LQR controller.

Overall, the project successfully implemented a state feedback controller. Of course if we had more time, an integrator would have been added to the LQR controller to eliminate steady state error, and the observer would have been completed - this will still be worked on by Ryan as a hobby project in the coming weeks of COVID-19 quarantine. All to date development can be seen on Github [6], on branch **master**. Gain tuning and Luenberger observer testing can be seen on branch **gain-tuning**, and will be merged once completed. Folder **matlab/** discusses system analysis, controllability and observability, as well as contains functions that were used for tuning the system. Folder **tiva/** contains all **C** code for the microcontroller board in Code Composer Studio.

References

[1] *Inverted Pendulum: System Modelling*. [Online]. Available: http://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum&section=SystemModeling

[2] *Inverted Pendulum: State-Space Methods for Controller Design*. [Online]. Available: http://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum&section=ControlStateSpace

[3] *Pole-zero plot of a dynamic system*. [Online]. Available: https://www.mathworks.com/help/control/ref/pzmap.html

[4] *Chapter 2: System Modelling*. [Online]. Available: https://www.cds.caltech.edu/~murray/courses/cds101/fa04/caltech/am04_ch2-3oct04.pdf

[5] *OPEN LOOP AND CLOSED LOOP SYSTEMS*. [Online]. Available: https://www.pesquality.com/blog/open-loop-and-closed-loop-systems

[6] R. Fielding, *Inverted Pendulum Github Repository*. [Online]. Available: https://github.com/ryanfielding/inverted-pendulum

[7] *Linear-quadratic regulator*. [Online]. Available: https://en.wikipedia.org/wiki/Linear\OT1\textendashquadratic_regulator

[8] *Inverted Pendulum: PID Controller Design*. [Online]. Available: http://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum&section=ControlPID

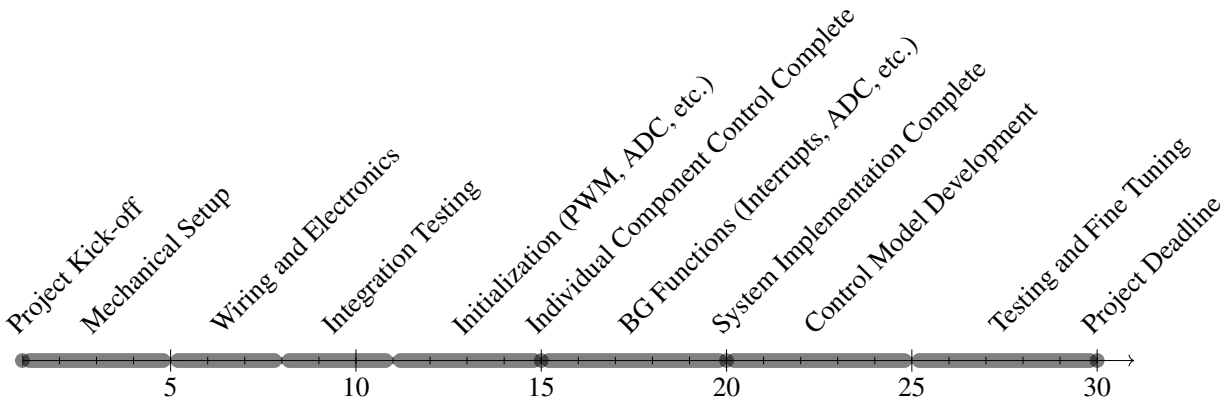
[9] *Using Hardware QEI on Tiva Launchpad*. [Online]. Available: https://forum.43oh.com/topic/7170-using-harware-qei-on-tiva-launchpad/

[10] *Quadrature Encoder Overview*. [Online]. Available: https://www.dynapar.com/technology/encoder_basics/quadrature_encoder/

[11] *PID Controller*. [Online]. Available: https://en.wikipedia.org/wiki/PID_controller

A Timeline

It has been decided upon that the project duration will last approximately 1 month, to ensure completion of the working control system before the due date. The following timeline shows the approximate estimated dates for each major phase of the project. It should be noted that some tasks will be completed in parallel rather than in series, when possible. The remaining dates of April 1 - 10 will be used for tuning the controller with MATLAB and testing.



Project Timeline for the Month of March, by Day

B State Calculation in C

```
void lqr(void){
    //xHat = pos posdot theta thetadot
    //need to estimate the derivative states, then apply exp avg filter
    float rPosDot = 0.05;
    float rThetaDot = 0.02;
    float rTheta = 0.15;

    xHat.X = (pos - ref.X)*scalePos;
    xHat.Y = (1-rPosDot)*posDotPrev + rPosDot*(xHat.X - posPrev)/dt;
    xHat.Z = (1-rTheta)*thetaPrev + rTheta*(4096.0 - read_ADC() - ref.Z)*scaleTheta;
    xHat.W = (1-rThetaDot)*thetaDotPrev + rThetaDot*(xHat.Z - thetaPrev)/dt;
    posPrev = xHat.X;
    posDotPrev = xHat.Y;
    thetaPrev = xHat.Z;
    thetaDotPrev = xHat.W;

}
```

C Observer Function in C

```
void obsv(void){
    y.X = (pos - ref.X)*scalePos;
    y.Y = (theta - ref.Z)*scaleTheta;

    yHat.X = HMM_DotVec4(C1, xHat);
    yHat.Y = HMM_DotVec4(C2, xHat);

    e = HMM_SubtractVec2(y, yHat);

    xHatDot.X = HMM_DotVec4(ABK1, xHat) + HMM_DotVec2(L1, e);
    xHatDot.Y = HMM_DotVec4(ABK2, xHat) + HMM_DotVec2(L2, e);
    xHatDot.Z = HMM_DotVec4(ABK3, xHat) + HMM_DotVec2(L3, e);
    xHatDot.W = HMM_DotVec4(ABK4, xHat) + HMM_DotVec2(L4, e);


    dt = stopTimer();

    xHatNext = HMM_AddVec4(xHat, HMM_MultiplyVec4f(xHatDot, dt));

    xHat = xHatNext;
    startTimer();

}
```

D Observer Header in C

It should be noted that these actual gains were not implemented, only present at time of report development.

```
//Gains for K, A-B*K, L from MATLAB
float k[4] = {-10,-10.7925,46.0882,7.94368};
float abk2[4] = {52.14475,55.60931,-239.0601,-41.42211};
float abk4[4] = {146.6096,156.7628,-651.391,-116.462};
float l1[2] = {430.0024,-162.9998};
float l2[2] = {5372.1669,-2736.4126};
float l3[2] = {-127.0637,177.8561};
float l4[2] = {3473.1059,110.9461};
```

E UART Interrupt Handler in C

```
void UARTIntHandler(void)
{
    uint32_t ui32Status;
    // Get the interrupt status.
    ui32Status = UARTIntStatus(UART0_BASE, true);
    //
    // Clear the asserted interrupts.
    //
    UARTIntClear(UART0_BASE, ui32Status);

    //UARTSend((uint8_t *)"nah\r\n", 5);
    //
    // Loop while there are characters in the receive FIFO.

    while(UARTCharsAvail(UART0_BASE))
    {
        // Read the next character from the UART and write it back to the UART.
        UARTCharPutNonBlocking(UART0_BASE, UARTCharGetNonBlocking(UART0_BASE));
    }
    send_u32(theta_target - theta);
    send_u32(pos);
}
```