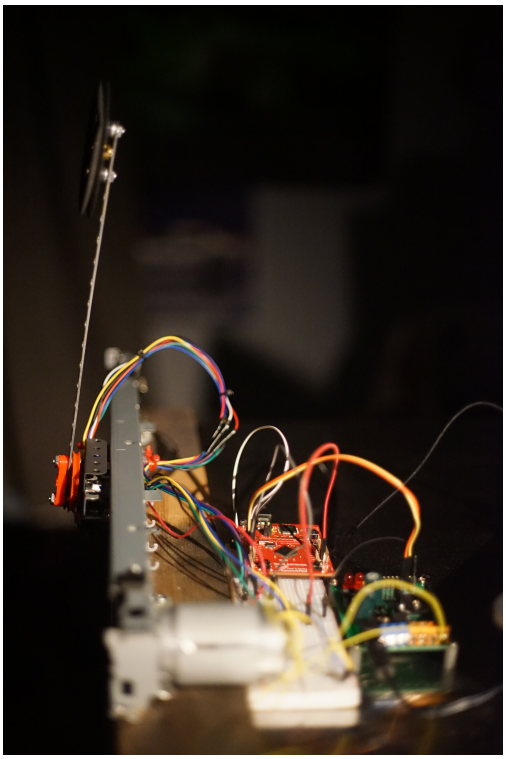
Real-Time Embedded Control System for an Inverted Pendulum

MSE 450
Final Project Report

Self Proposed Project Group 1
Rachel George - 301288581
Ryan Fielding - 301284210

April 10^{th}, 2020



Abstract

This report outlines the project completion for the MSE 450 (Real-time and Embedded Control Systems) course project, which has been developed in conjunction with the MSE 483 (Modern Control Systems) course project. In short, the system implements a state space model observer based 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 initialization 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.

Initialization of the microcontroller and all working components was completed by March 15^{th}, 2020, to allow enough time for state model development and testing. This includes but is not limited to, measurement of θ and x, ensuring that the position increments properly at high speeds, and adequate position control of the DC motor powered cart. At this point, the observer based state feedback controller has been implemented, and vigorously tested. The final control system has been completed and tested by March 30^{th}, 2020, however further gain tuning is required for stable control of the pendulum. Additionally, controller tuning is only comprised of software changes.

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. For this report and project demo video, a simple P controller has been implemented.

Contents

List of Figures

1 Introduction

1.1 Overview

This real time embedded control 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 Control System Background

The controller's commands is dependent on the feedback of the state system; this ultimately entails a closed loop system. As exemplified below (Figure 1), it is the difference between a closed loop system and an open loop system. Simply put, an open loop system has no indication of its perpetuated error that it may be incurring, therefore feedback loops or closed loop systems can respond accordingly to disturbances that deviate the output from the target value.

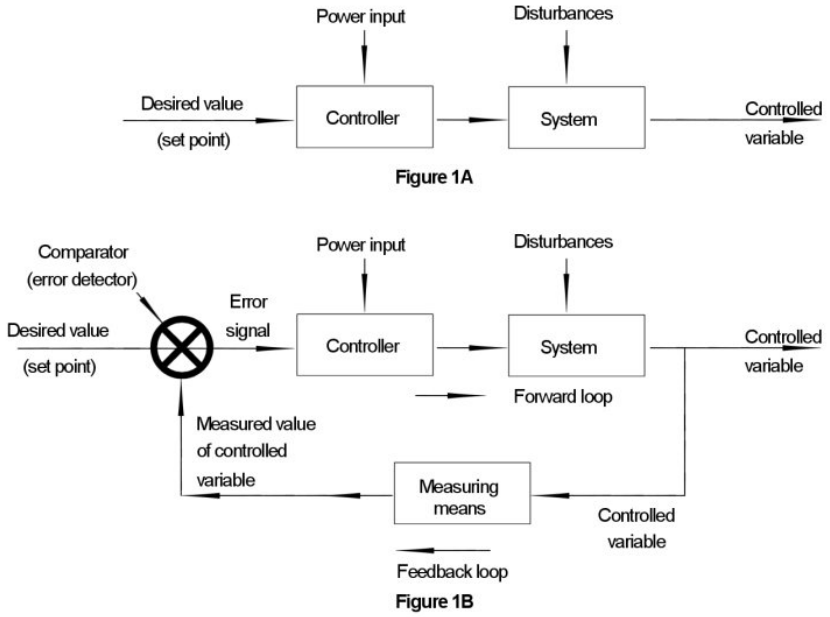


Figure 1: Open Loop System Versus a Closed Loop System [1]

1.3 Problem Investigation

The measure of a successful closed-loop system is merited by its degree to which the measured final value compares to its target value after the system responds to a disturbance. This success can be measured by multiple characteristics of the system response. One such characteristic is the maximum error, a value which corresponds to the initial overshoot of the system response [1]. Another valuable criteria is the response time and the settling time of the system; the time from which an error is detected to when a corrective action is taken, and the time taken for the controller to complete its corrective path, respectively. After the final value is obtained, the steady state error is observable - the absolute difference of the final measured value and the ideal value, see the figure below for this visual representation (Figure 3).

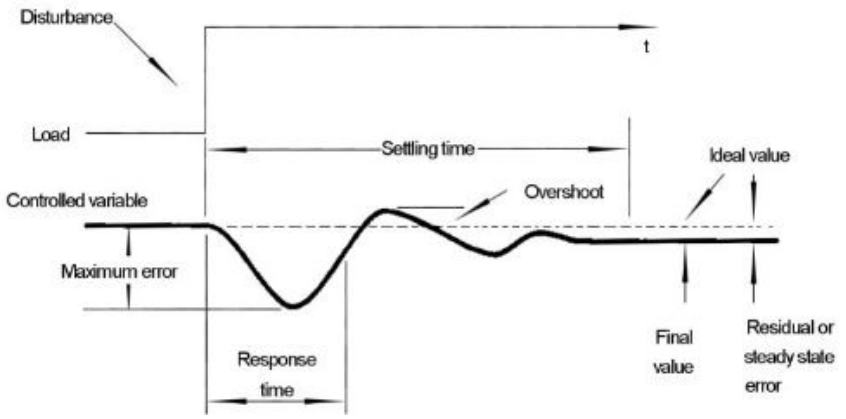


Figure 2: System Measurement - System Response to Disturbances [1]

State Space Modelling is the selected model implementation, ergo using the ”notations of state, inputs, outputs and dynamics to describe the behavior of a system” [5]. 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.

1.4 Proposed 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

1.4.1 System Integration with PID Control

Once the system setup and software architecture was completed (initialization, handlers, variables, etc.), the first type of controller that was implemented was a proportional-integral-derivative (PID) controller, a basic, practical controller that will enable verification that the entire system is functioning properly. The following figure displays the overall control dynamics of a PID controller.

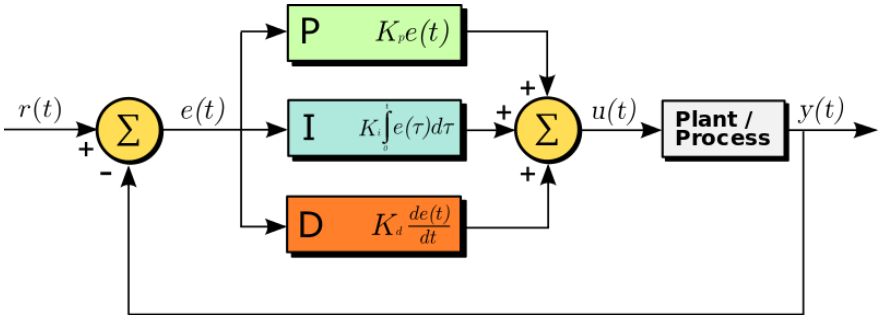


Figure 3: PID Control [2]

Once sensors, handlers and all supporting functions have been tested, we can move on to developing a more complex controller via the course material of MSE 483.

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

$$u = -Kx + r \tag{1}$$

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 3.3.1). For example, the cart weight, pendulum inertia, length and more have significant impacts on the modelling of the system, including the controllability and observability. Section 2.2 outlines the theoretical knowledge for said controller. Once these values are recorded, MATLAB will also be used to tune the gain vector L, also known as the Luenberger observer.

1.5 Proposed Expected Outcomes

We expect to obtain a system response and from observing its characteristics previously explained, we plan to manipulate the system to minimize the steady state error. This error responds to a system gain, 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 [1]. 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.

2 System Design

2.1 Objectives

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 4 below.

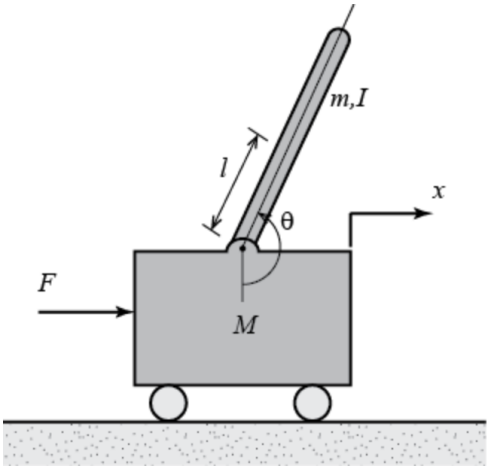


Figure 4: System Concept - Inverted Pendulum on a Cart [3]

Variables

- x the position of the cart
- θ the angle of the pendulum
- F applied forces to the cart
- M, m masses of the cart, pendulum
- l, I length and inertia of the pendulum

2.1.1 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.

2.2 Theory

2.2.1 Control Model

MSE 483 - Modern Control Systems / Intro to State Space Control Systems - effectively outlines the process of implementing a state space model based control system. 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 controls 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.

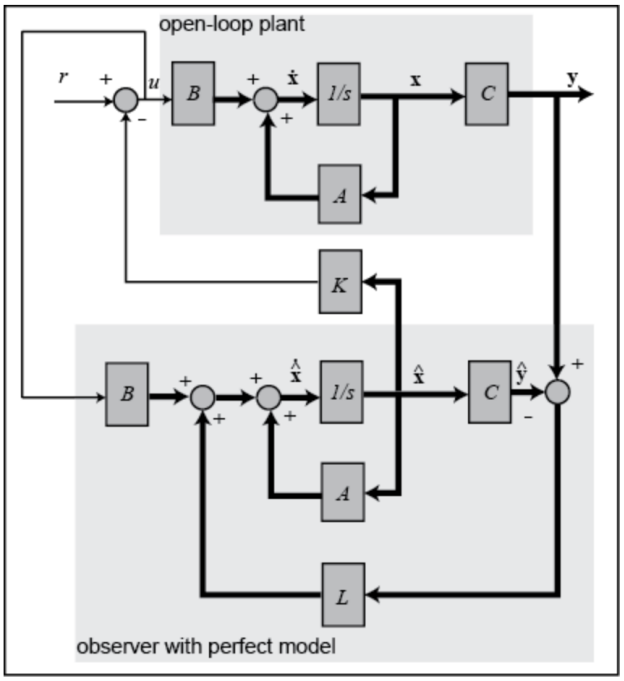


Figure 5: Observer Based State Feedback Model to be Implemented [4]

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:

$$\dot{x} = Ax + Bu \text{ and } y = Cx + Du \tag{2}$$

And for the state estimator:

$$\dot{\hat{x}} = A\hat{x} + Bu + L(y - \hat{y}) \text{ and } \dot{e} = \dot{x} - \dot{\hat{x}} = ... \implies \dot{e} = (A - LC)e \tag{3}$$

Who's poles will be designed as 5-10 times faster than that of the LQR controller, so as to drive $e \implies 0$ as fast as possible.

2.3 Methods

2.3.1 Experimentation

To evaluate the proposed control models, experimentation will roughly proceed as follows:

1. Power on the microcontroller with latest code developments.
2. Normalize the cart position and pendulum angle (move cart to center of guide rail, hold pendulum vertically).
3. Press SW1 on board to trigger interrupt, thus set run mode to TRUE, release the pendulum and evaluate controller effectiveness / stability.
4. Introduce minor disturbances to the pendulum and the cart, evaluate system response.

2.3.2 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 location, ie. $x = 0cm$.

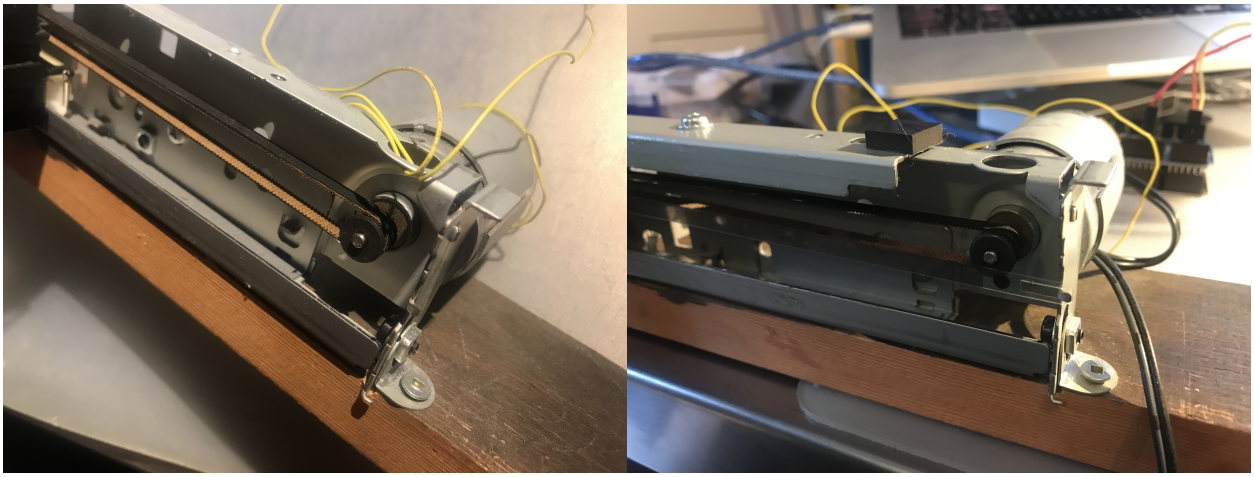
Further requirements may be set for the "Nice to Have's" outlined in section 2.1.1, depending on the project's completion schedule. Assuming all prior requirements are complete, we will investigate said problems and construct design requirements.

3 Implementation

3.1 Hardware

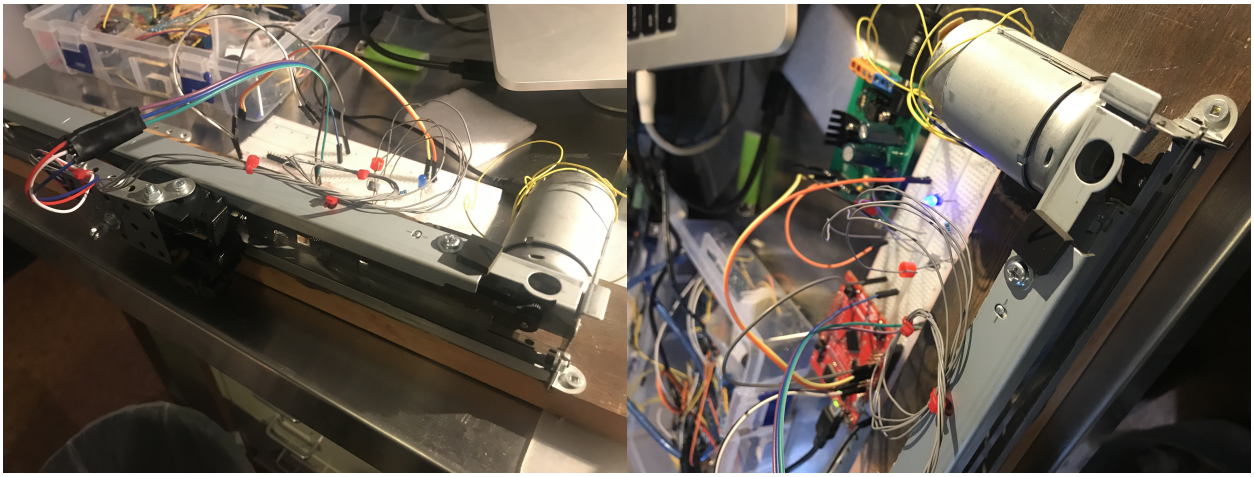
3.1.1 Mechanical

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



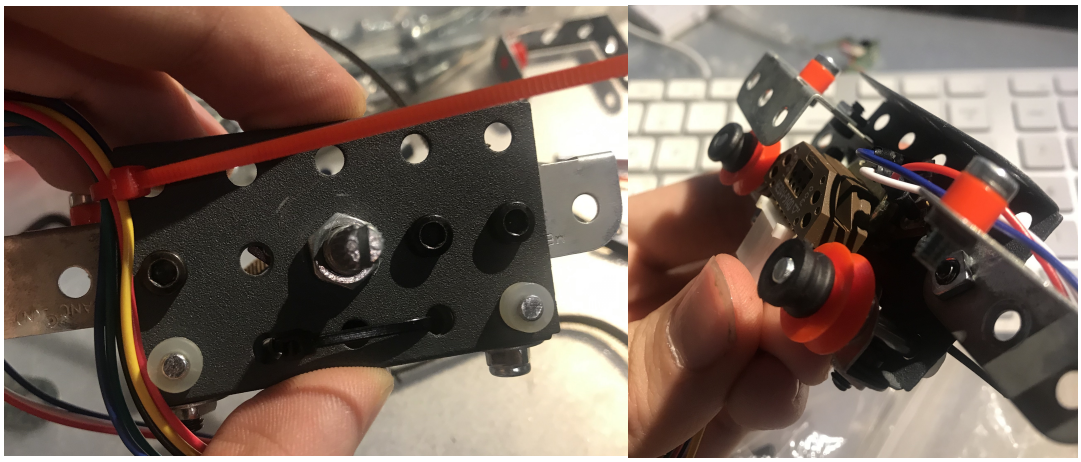
(a) DC Motor Belt Drive (b) Optical Encoder Linear Strip

Figure 6: DC Motor Belt and Feedback



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

Figure 7: System Setup Development



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

Figure 8: Custom Built Low Friction Linear Guided Cart

3.1.2 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.

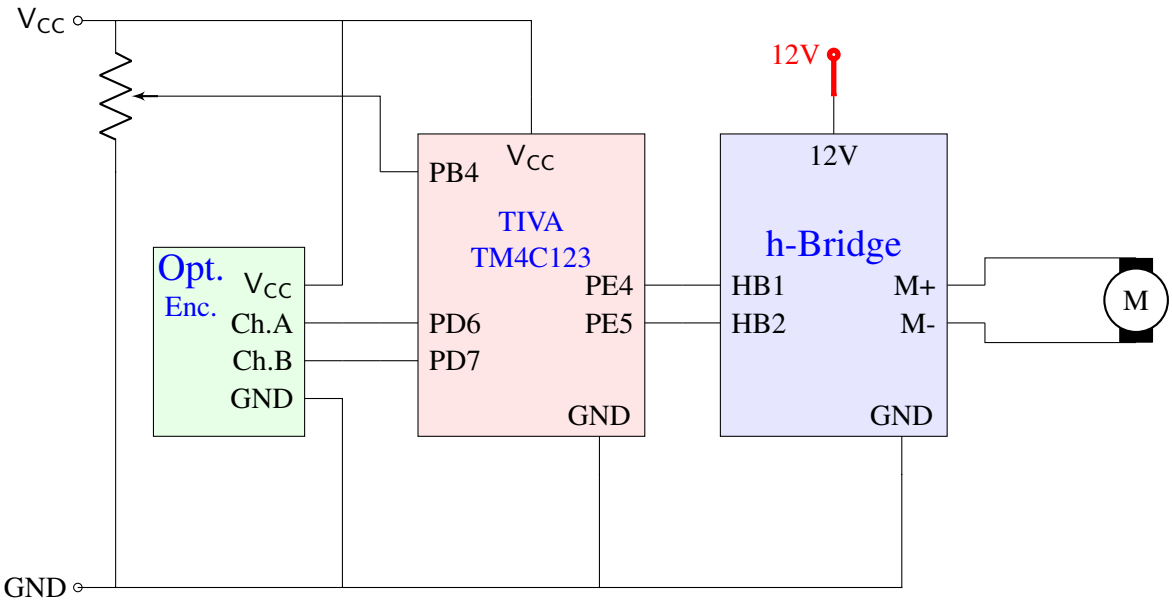


Figure 9: Circuit Drawing

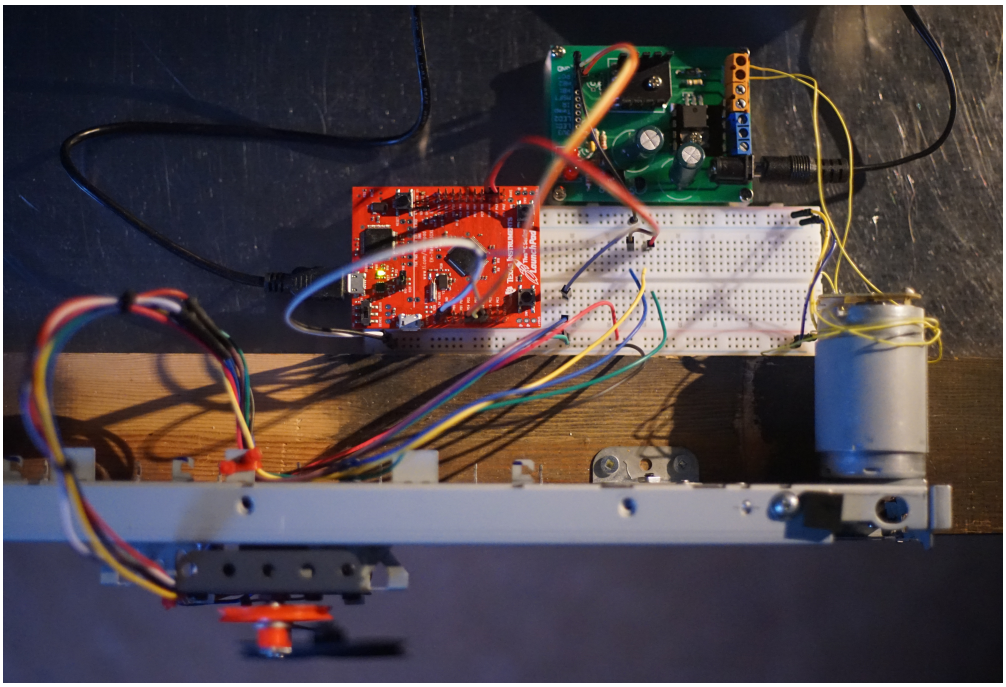
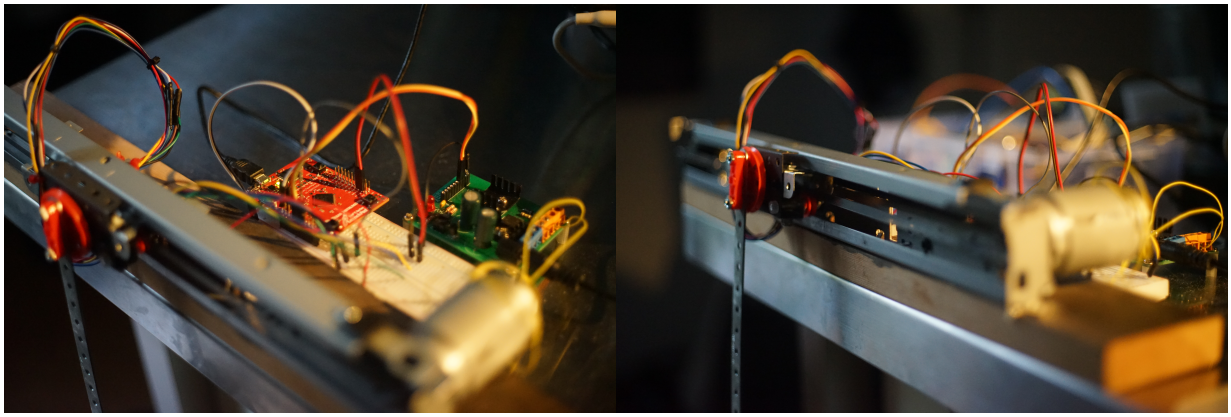


Figure 10: Completed System Electronics

3.1.3 Completed System



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

Figure 11: Built Inverted Pendulum System

3.2 Software

To develop the state space model analysis, software tools MATLAB and SIMULINK will be used. MATLAB to develop the system state model, test and implement observer based feedback control, and SIMULINK to simulate and test the system, as well as fine tune the control gains. Furthermore, the controllability and observability of the system can be assessed in MATLAB, and will be discussed in the final project report.

3.2.1 Control System Flowchart

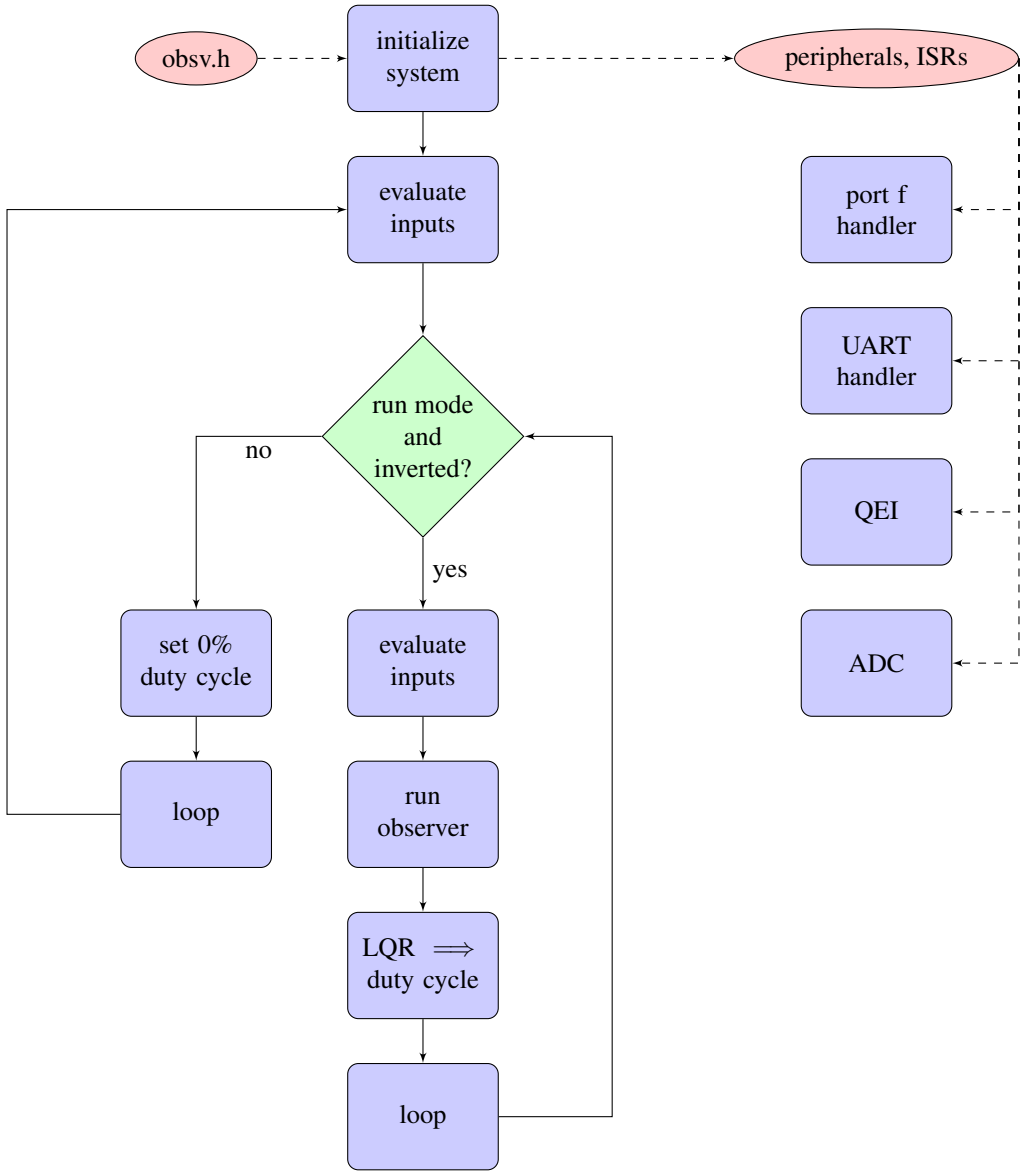


Figure 12: Controller Flowchart

3.2.2 Luenberger Observer Implementation

To elaborate on our control model approach shown in figure 5, 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 1: 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

$y = Cx$
while $e \geq \tau$ **do**
    $\hat{y} = C\hat{x}$
    $e = y - \hat{y}$
    $\dot{\hat{x}} = (A - BK)\hat{x} + Le$
    dt = stopTimer();
    $\hat{x_n} = \hat{x} + \dot{\hat{x}}dt$
    $\hat{x} = \hat{x_n}$
    startTimer();
end

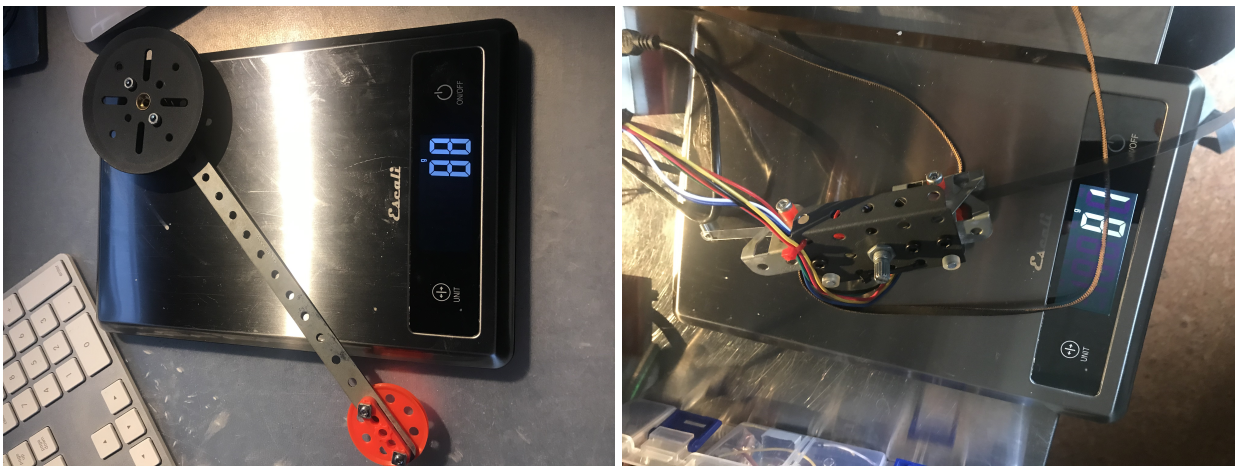
This algorithm implementation in C can be seen in appendix B.

3.3 Gain Tuning

There are various methods for tuning that are proven and that range in trial-and-error, iterative behaviour, aggressiveness of tuning, process time etc. For the sake of simplicity and the completion of our proof of concept we resorted to a manual tuning method.

3.3.1 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 13: Measurement of System Constants

All resulting MATLAB code can be seen in the *model.m* on Github [6].

3.3.2 Modelling System Responses

After completing the state model for the linearized system, as discussed in section 2.1, 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. Figure 14 shows the simulated system response for a cart position step input of *20cm*.

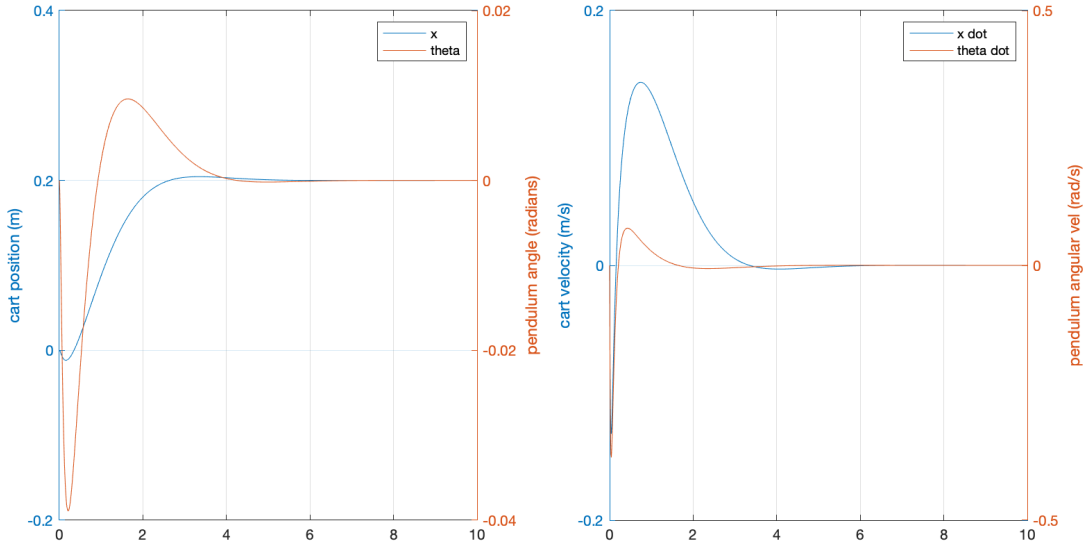


Figure 14: Observer Based State Feedback System Simulated Response in MATLAB

4 Results

4.1 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 C.

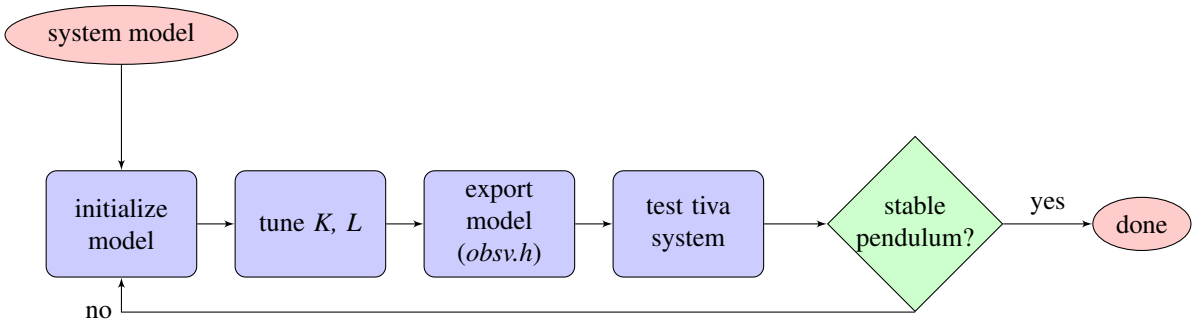


Figure 15: Tuning Process Flowchart

4.2 Stability and Response Characteristics

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, the issue lies in the implementation of the observer. As discussed earlier and seen in figure 5, the Luenberger observer is a complex system that takes time to tune and test, and is out of the scope of this report. Currently, the error variable, e (see equation 3), diverges to infinity, and consulting with the MSE 483 course TA is underway.

Instead, the basic response characteristics and results will be observed for the basic PID controller. Figure 16 displays the states of the system, x, θ, as the pendulum responds to minor impulses.

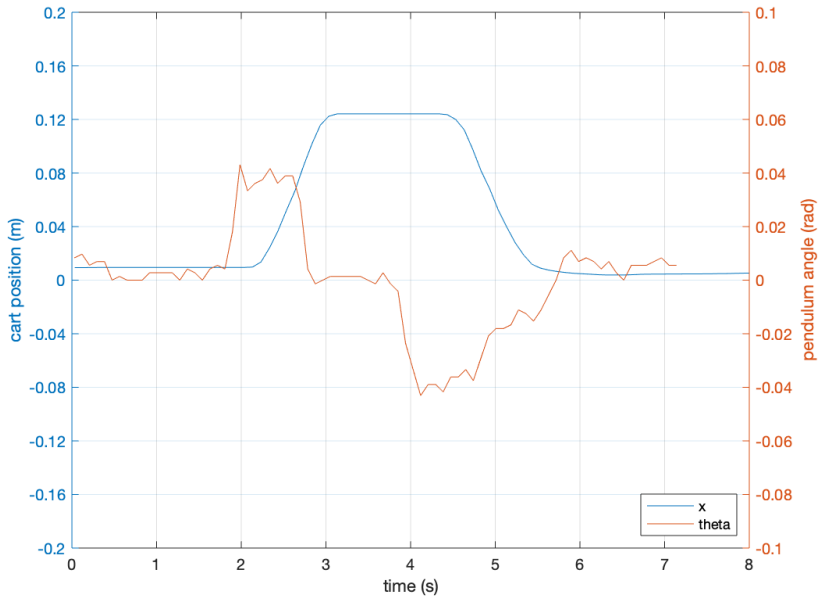


Figure 16: Physical System Response

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 [7], 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.

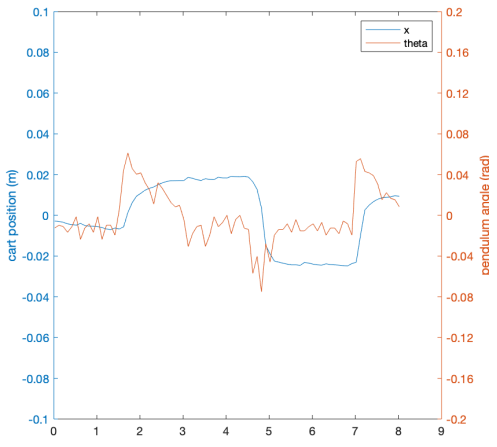
4.3 Additional System Enhancements

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

4.3.1 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



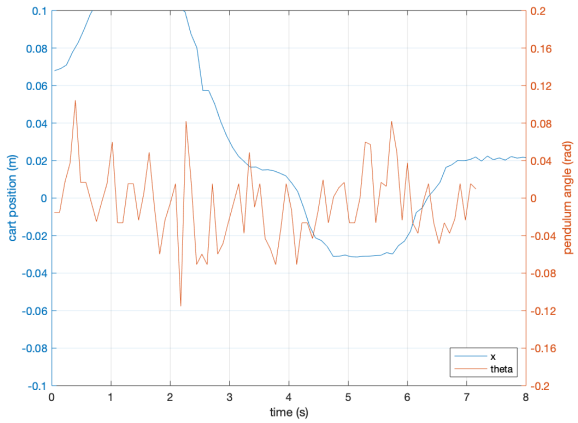
(b) Resulting Noise

Figure 17: Faulty Connection Potentiometer Noise

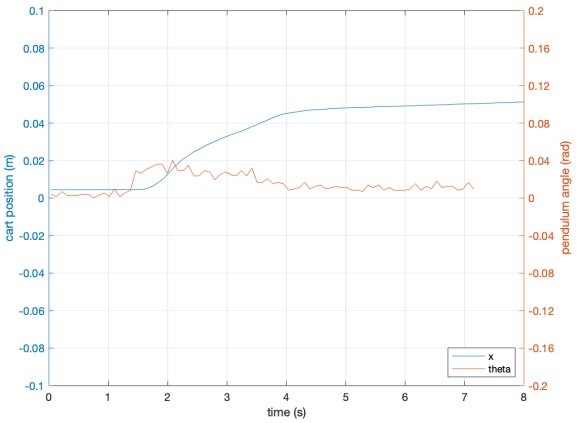
4.3.2 Moving Average Filter for ADC Data

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

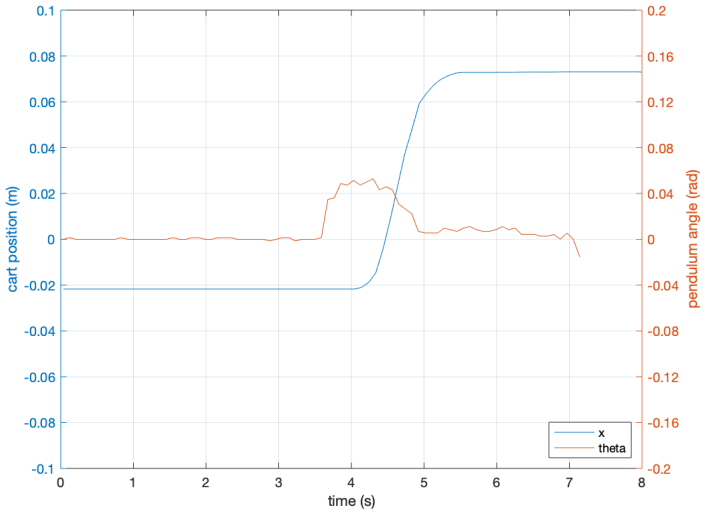
than implementing an RC filter and delaying the signal with too large of a capacitor, we implemented a simple moving average filter. The following images display the before and after signal conditioning results.



(a) No Filter



(b) Moving Average, Length 10



(c) Moving Average, Length 25

Figure 18: ADC Signal Noise Moving Average Filtering

4.3.3 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 [8]. Below is an image which illustrates the signals produced by a QEI (figure 19). 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 hardware was implemented. This was able to essentially read the cart's position separate from the main CPU.

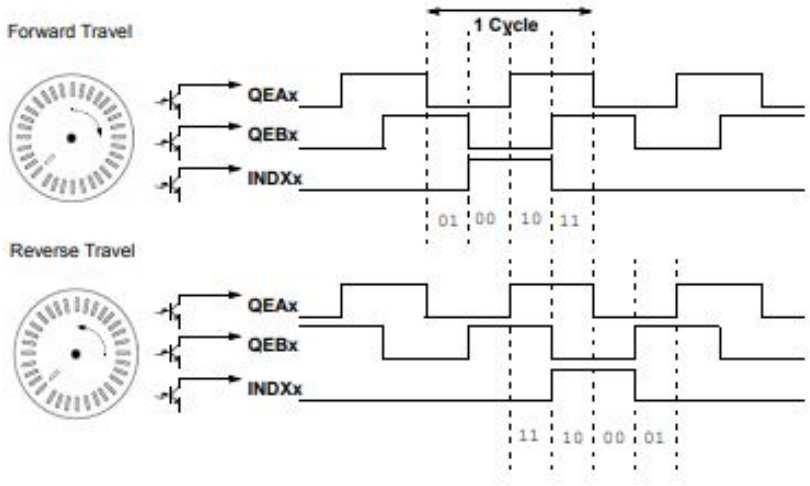


Figure 19: Quadtrature Encoder Interface Signal

4.3.4 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 D.

5 Conclusions

Overall, this system is completely feasible, controllable and observable. Within this 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 controller. Furthermore, proposed GPIO for the microcontroller has been completed and checked in the course lab experiments, hence all the required knowledge is possessed. 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 inevitably 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 successfully by the end.

As for the implementation of the Luenberger observer state feedback controller, the tuning is nearing completion. An online session has been scheduled with the MSE 483 TA to ensure the convergence of the error variable, e, to 0. Once this is completed, the system can be further tuned for quick 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. Pole placement is key in control system development, and functions such as **place** and **acker** enhance this process.

In terms of tuning the PID controller, a more standardized method of tuning could have been implemented such as the *Zielgler-Nicolas method* as opposed to finding an experimentally sufficient gain. Our approach was able to find a well-fitting proportional "P" gain, however we did not implement a fully developed gain for a proportional, derivative and integral "PID" controller. Regardless our main design goals were ultimately met, this is rather a suggestion for bettering the experiment in the future.

Finally, the timeline earlier proposed (appendix A), was for the most part kept on schedule and successfully implemented. From this experiment, topics of embedded systems thoroughly explored were interrupts, ADC, PWM, and integration of design - to name a few.

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. Folder **tiva/** contains all **C** code for the microcontroller board and Code Composer Studio.

References

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

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

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

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

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

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

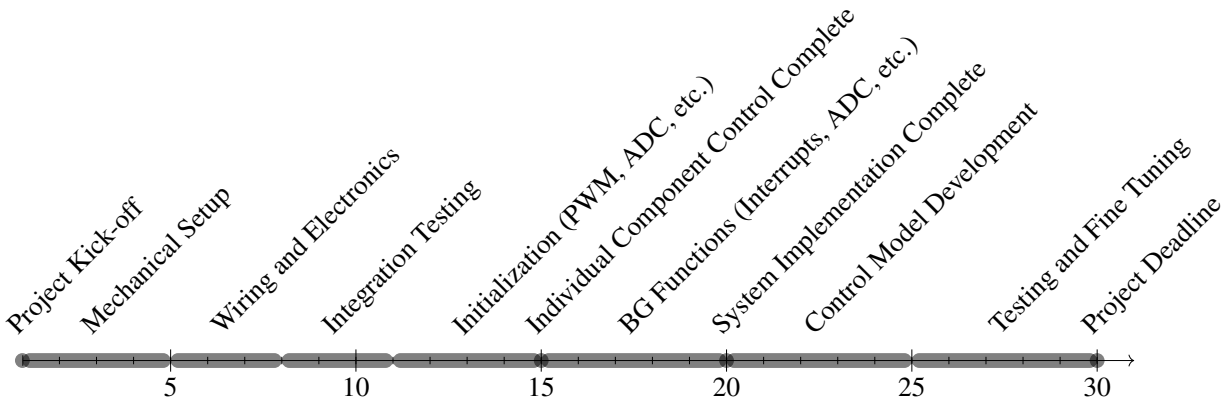
[7] *Inverted Pendulum: PID Controller Design*. [Online]. Available: http://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum&section=ControlPID

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

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

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 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();

}
```

C 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};
```

D 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);
```

```
21      send_u32(pos);
22  }
```