Virtually Trained Self-Balancing Pendulum

ECE 445 Design Document

Henry Thompson, Kishore Adimulam, and Mason Ryan
Group 31
TA: Amr Martini
3/6/2019

1. Introduction

1.1 Objective

There is a growing use for virtual reality as a training environment for AI for applications in the real world. Game engines like Unity have even released machine learning tool-kits for researchers and developers to experiment training reinforcement learning algorithms inside games and simulations. There has been past work in translating these simulation-trained models to physical systems, such as the project done by OpenAI which taught a robot to stack different colored blocks in a specific order only after seeing it once in a virtual simulation [1]. However, the use of game engines to perform similar tasks has been limited, since there are no clear workflows for how someone can use AI models trained in an easy-to-use game engine like Unity to perform inference tasks in a physical system.

Our solution would be to create a self-balancing inverted pendulum system, which would be trained as a simulation in Unity and uploaded into a physical system. The system consists of a cart connected to a one-dimensional track, with an attached pendulum on a hinge. The goal of the system would be to move the cart on the track in either left or right, in order to balance the pendulum in a vertical position. We would create a physical system which replicates all of the attributes of a 3D simulation, and train the agent in the simulation to learn to balance the pendulum using the Python API and Tensorflow. The trained Tensorflow model would then be uploaded into a microcontroller, which would then use the control signals of the agent to operate motors to balance the real physical pendulum.The Kalman filter will be used to reduce the noise from the output of the IMU to get a better estimate of the cart's current acceleration.

1.2 Background

The ability of virtual reality to model physics and interactions between materials positions it to become an ideal tool for simulating environments for artificial intelligence. This allows experiments to be carried out on a much larger scale, at a fraction of the time and resources required to carry out physical tests. Once the low-level controls and modeling for a system has been determined, learning how to carry out more advanced tasks, such as training a robot to jump or to drive a vehicle autonomously, could become possible through using reinforcement learning inside simulations. Since the applications for virtual trained AI agents are numerous, it begs the question how easily such systems can practically be implemented using existing software. One of the most popular current game engines, Unity, has made it easy for developers to train their own AI agents using Tensorflow through the ML-Agents toolkit [2]. Furthermore, TensorFlow Lite has made it possible to deploy machine learning models and perform inference tasks on embedded devices such as the Raspberry Pi [6]. We aim to create a solution where a Tensorflow model trained in a 3D simulation in Unity can be deployed into a microcontroller to balance a physical inverted pendulum system.

1.3 High Level Requirements

- The agent in the Unity simulation must be able to balance the pendulum within 10 degrees of vertical given a random initial state also within 10 degrees of vertical for the duration of a test run with a 90% success rate, given 10 second test runs.
- The agent in the physical system must have a 90% success rate of being able to balance the pendulum within 10 degrees of vertical, with no prior experience outside of the simulation and given a random initial state of of within 10 degrees of vertical for the duration of a test run, given 10 second test runs.

2. Design

Our design will consist of four major subsystems: the machine learning subsystem, the control subsystem, the mechanical subsystem, and the power subsystem. The detailed block diagram of our project can be seen in Figure 1.

The machine learning subsystem consists of our Unity simulation as well as the Tensorflow model produced from the Unity simulation. Once the simulation has been trained to balance the pole, we can then generate a Tensorflow file that is ready for interfacing with the Raspberry Pi. This model will be stored in memory on the microcontroller and will take in the current state of the pendulum (velocity of the cart and angle of the pendulum) as inputs and output the required force on the cart.

The Power system will consist of our 6V power supply, a 5V DC step down regulator, and a 6V DC regulator. The output of the 5V DC regulator will be used to power the microcontroller, which will then supply power to the photodiode and the IMU sensor. The output of the 6V regulator will be used to power the motor controller at a constant 6V DC.

The Mechanical subsystem will consist of the motor controller, the motor, the wheel, the cart, the pendulum, the pendulum potentiometer, and the motor encoder. The motor controller will take in two inputs, one binary signal from the microcontroller that determines which direction to rotate the motor, and the other is the duty cycle of the PWM from the microcontroller. The motor controller will take these inputs and output a 6V PWM signal to the motor. The motor will take the input from the motor controller and rotate the motor at a specific RPM depending on the duty cycle of the signal. The rotation of the motor will spin the wheel and cause the cart to move a specific velocity. The angle of the pendulum will change depending on the force applied to the cart. This angle will be measured by the pendulum potentiometer and will be sent to the control subsystem to help determine how the cart should move. The RPM of the motor will also be determined from the motor encoder. This RPM will be used in the control subsystem to calculate the current velocity of the cart, which will be used to help determine how to move the cart.

The control subsystem consists of our microcontroller, a Raspberry Pi, a photodiode, and our backup PID controller. The Raspberry Pi will take in inputs from the photodiode and the pendulum encoder to determine whether the PID controller or the Tensorflow model should be used to balance the pendulum, or whether it should stop and reset the cart. The PID controller will be stored in memory and will be executed when the measured angle of the pendulum is out of the bounds that we have set. The photodiode and pendulum encoder will be used to decide which mode to run the system in. The photodiode will determine if the cart's position is out of bounds that we have set for it. The pendulum encoder will determine the angle of the pendulum with respect to its upright position. The process for determining which model to use can be seen in the controller software design in Figure 3.

We will also have a backup control system plan in the situation in which we determine that the motor encoder cannot accurately be used to predict the velocity of the cart. We will conduct the verifications in the tolerance analysis for the wheel, to determine the max PWM signal we can drive our motor with while also being able to predict velocity and prevent wheel slippage. In the situation we find this approach infeasible, we will use a backup design of an IMU, linear quadratic regulator, and Kalman filter to measure the acceleration of the cart, and have our control system set the velocity accordingly.

Within this control system the IMU will calculate the current acceleration of the cart. Since the IMU output is generally noisy, we will pass it through a Kalman filter to reduce some of the noise. The output of the Kalman filter will be fed into the LQR system where it will try to send the right PWM signal to the motor controller to achieve the desired acceleration as calculated from the Tensorflow model. This system will loop until the estimated acceleration produced from the Kalman filter is the same as the desired acceleration. Of course this process will introduce some delay into the system as the control system attempts to ramp up the acceleration to the desired value. However, this as long as this delay is sufficiently small we can account for the lag between the data inputs and outputs in Unity and still be able to balance the pole.

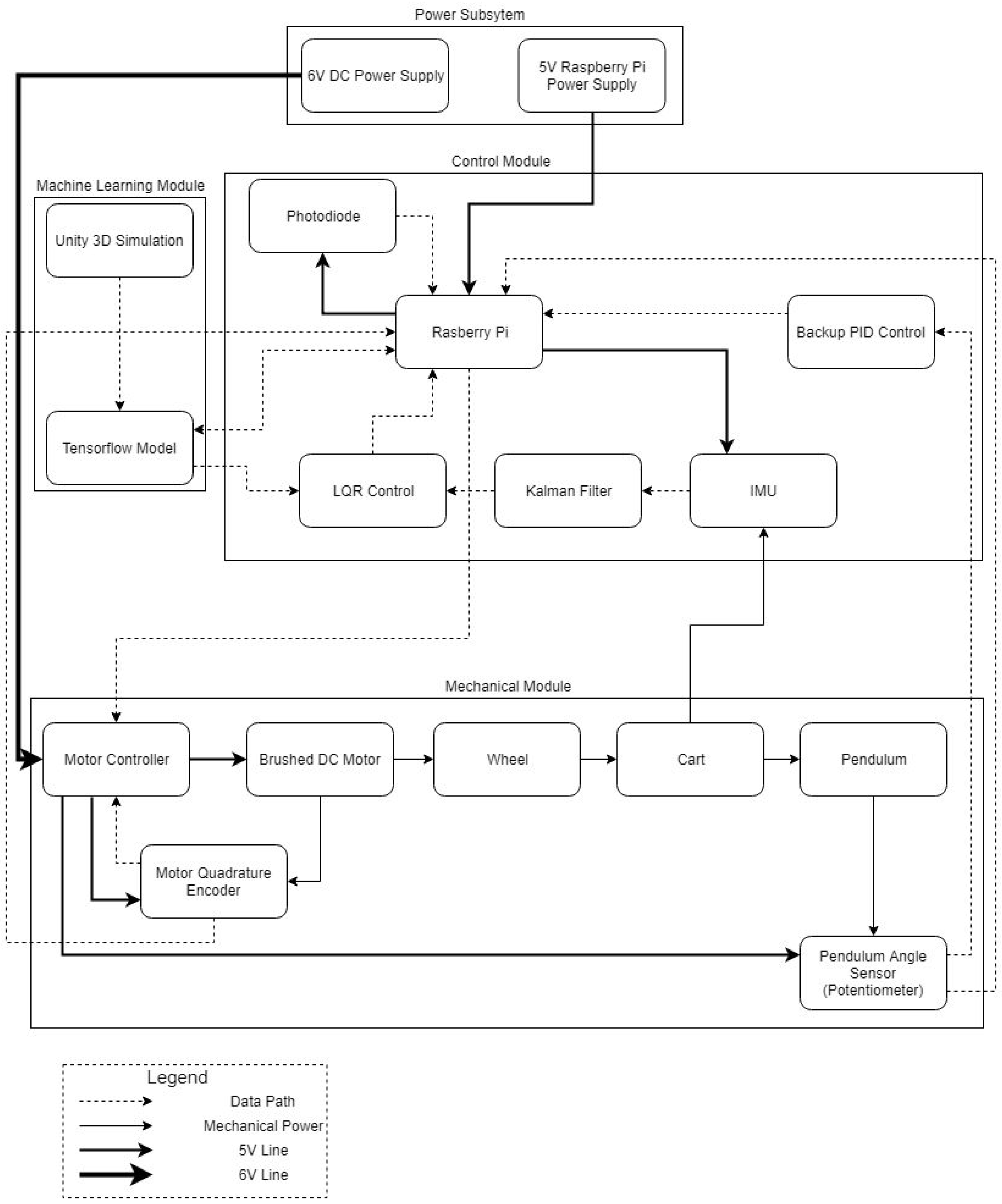

Figure 1: Block Diagram

2.1 Machine Learning Module

For our project we will be training our cart-pole system entirely in a game engine, without any experience directly in our physical system. The simulation will be modeled as closely as possible to our physical system so that that learned model can successfully balance the pendulum in the physical system. From the Unity simulation we will obtain a Tensorflow model which we will upload to the controller.

2.1.1 Unity Simulation

Within the Unity Simulation we can change the necessary parameters to best model the physical system. Once we have replicated the physical design within Unity, we can start to train the system using Unity machine learning toolkits. We we feed the neural network rewards when the angle of the pendulum is within a certain range from the equilibrium and punish it for being too far from the equilibrium. We can fine tune the reward functions until we find the the learned model is properly balancing the pendulum.

Requirement	**Verification**
Given an initial state of the pendulum within 10 degrees of vertical, the agent in the simulation must be able to maneuver the cart to keep the pendulum balanced.	1. Generate a random initial angle which is between -10 and 10 degrees w.r.t. vertical 2. Set the pendulum angle to the random angle and start the simulation 3. Ensure that the agent can maneuver the cart left and right to balance the pendulum within the bounds for the duration of the test run (10 seconds)
Ensure the force generated by the model needed to balance the pendulum will not surpass max change in acceleration of motor	1. Measure the force applied by agent throughout the course of simulation 2. Give the agent a negative reward and truncate the signal of force if it exceeds the max possible acceleration of the motor 3. Ensure that with the truncated force the simulation still balances for 90% of test runs

2.1.2 Tensorflow Model

The Tensorflow model produced by the Unity simulation will be in the form of a protocol buffer (.pb), and checkpoint (.cptk) files. These will contain information of the graph and variables of our resulting model. The model will be compiled and deployed on our Raspberry Pi to perform control. The model will take in input from our sensors, and output the necessary force needed to apply on the cart to balance the pendulum.

Requirement	**Verification**
Tensorflow model needs to be small enough to be fit on the RAM of our microcontroller	1. Compile the Tensorflow file via uTensor library 2. Ensure that the resulting model file size plus the weights can fit on the available memory
The computation time of the model must be small enough, s.t. the sum of the time plus the rise time of the motor will be less than the sample delay time the NN can be trained with and still be able to work	1. Determine the rise time of the motor, and add it to the computation time of the model 2. Check if the NN can be trained to succeed with balancing at 90% accuracy with a sample delay time larger than the sum of the two given times.

2.2 Control Module

The control module will determine the duty cycle and polarity of the PWM signal to supply to the motor, which will set the velocity of the cart. It will take inputs from the potentiometer and motor encoder to determine the current state of the system. After processing the current state the control module will decide which model to use to balance the pendulum. It will also store the Tensorflow model, and the PID controller for balancing the pendulum in memory. Since our Tensorflow model outputs a desired force to apply on our cart, and our motor controller can only set a desired velocity of the motor, we will perform a conversion to determine the velocity to set the motor given the force applied to the system. We will ensure that this new velocity does not exceed the maximum acceleration allowed by the motor through our verifications on the allowed outputs of the tensorflow model.

2.2.1 Microcontroller

We will be using a Raspberry Pi for our microcontroller. This decision was based on the fact there is good support for deploying the Tensorflow Lite framework on a Raspberry Pi. The inputs for the microcontroller will be readings from our motor encoder and potentiometer to determine the velocity of the cart, as well as the angle of the pendulum. After our NN outputs a required force on the cart, we will have to determine the final velocity that want to set our motor at to simulate the application of force on our cart. This final velocity will be determined by solving the equation for acceleration, below.

$$a = \frac{\left(v_{final} - v_{initial}\right)}{\Delta t}$$

Our Δt will be a pre-computed constant that is greater than the rise time of our motor, since this will allow the motor enough time to get to the final velocity. Once the final velocity is determined, the motor controller will send a PWM signal to the motor to achieve the desired speed.

Requirement	Verification
Must be able to process the data from the sensors and Tensorflow model in a time less than the sample delay time that the NN in simulation can be trained to function with	1. Determine total computation time of tensorflow model plus sensor lag time 2. Check that the NN can be trained in simulation to balance pendulum give the total delay time

2.2.2 Backup PID Control

The backup PID controller will be used for our project when the angle of the pendulum is too far from equilibrium. We have decided that this is beneficial, as we do not want to receive an adverse signal from the neural network for extreme situations. We would rather have a backup method that is already well researched that we can rely on when the state of the pendulum is not easy to balance. The code for this PID control will be stored in memory, and we can tune the values for k_d, k_i, and k_p once we have built our physical system. The PID control should also run faster than the Tensorflow model as we do not need to calculate the acceleration for each state of the system.

Requirement	Verification
The PID controller must be able to balance the inverted pendulum for angles between 10-45 degrees from equilibrium for 10 seconds with a 90% success rate.	1. Hold the pendulum at an angle between 10-45 degrees. 2. Let go of the pendulum at that angle while the PID controller is active. 3. Wait 10 seconds to see if the pole is still balanced. 4. Repeat test enough times to get a good idea of the success rate of the controller.

2.2.3 Photodiode

We will be using a photodiode to check if the cart goes beyond a position on the track that is too close to the end of the track. We will determine the furthest position the cart to go by calculating the maximum velocity of the cart and using the response time of the photodiode. We can multiply these values together to determine where to place sensor far enough from the bumper so that the system will be able to stop the cart before it come into contact with the bumper. When the cart passes the photodiode it will block light from the photodiode and not allow current to pass through. We can measure this change in current on our microcontroller and then promptly shut down power to the motors before the cart runs into the bumper. For our project we will be using the TEFD4300 photodiode for its fast response time, sensitivity within visible light spectrum, and its known application as a position sensor.

Requirement	**Verification**
The photodiode must have a measurable change in current when the cart passes by the sensor.	1. Place the photodiode on its mount on the track. 2. Manually push the cart pass the sensor. 3. Using an oscilloscope measure the change in current as the cart passes the photodiode.

2.3 Mechanical Module

The mechanical module holds all of the hardware of our project. It consists of the motor controller, motor, motor encoder, the cart, the pendulum, and the pendulum potentiometer. The pendulum will be attached to the side of the cart. Check the physical design section for more detailed design. The mechanical module also consists of the sensors that are attached to the physical system, such as the motor encoder and the pendulum potentiometer. These sensors will be used to track the current state of the system, and will send this data to the control unit for processing.

2.3.1 Pendulum Potentiometer Angle Sensor

We will be using a potentiometer angle sensor to calculate the pendulum's current angle. The sensor will output a voltage that can be converted to an angle due to the linear correspondence between the sensor's angular position and resistance.

Sketch code:

```
int getDegrees()
{
  // Read the raw sensor value
  int sensor_value = analogRead(ROTARY_ANGLE_SENSOR);

  // Convert the sensor reading to degrees and return that value
  float voltage = (float)sensor_value * ADC_REF / 1023;
  float degrees = (voltage * FULL_ANGLE) / Sensor_VCC;
  return degrees;
}
```

Requirement	**Verification**
The angle reading from the sensor and the voltage output of the sensor must have a linear relationship with an r^2 value of above 95 percent.	1. Perform linear regression on the trend between the angle reading of the sensor against the voltage output of the sensor 2. Calculate the r^2 measurement and ensure that it is above 95 percent.

2.3.2 Motor Quadrature Rotary Encoder

We will use a quadrature rotary encoder on the motor's extended motor shaft to count the revolutions the motor turns. We plan to limit the output from the Tensorflow model such that we avoid tire slip so we can use this encoder to keep track of the number of rotations of motor. Then, using the motor's metal spur gearbox gear ratio and circumference of the tire attached to the motor, we can calculate the cart's velocity by dividing the change of the angle by a certain Δt.

Requirement	**Verification**
The encoder must be accurate within $\frac{360}{1024} = 0.36$ degrees of the motor shaft's actual angle.	1. Connect the sensor to our motor and rotate the motor for a known number of steps to designate one full rotation. 2. Ensure that the deviation between the angle recorded from the sensor and the true rotation of the angle is within 0.36 degrees.

2.3.3 Motor Controller

We will be using the Pololu Dual VNH5019 Motor Driver. The motor driver will take a PWM signal input from the Raspberry Pi and take that signal and increase it to the 6V signal needed to run the motor.

Requirement	**Verification**
Must support setting motor speed for velocity based control. The output speed of the motor must be within 0.5 percent of set speed.	1. While motor is stationary, set the speed to a desired value for a given time step needed for one full rotation of the wheel, given the acceleration time of the motor. 2. Measure rotation of the wheel to check that the resulting position of the motor during the given time step was within 0.5 percent of the expected position given our desired set speed..

2.4 Power Module

The power module will consist of our 5V Raspberry Pi power supply, as well as our 6V power supply for the motor controller. We will be using the included power plug for the Raspberry Pi, but need to verify the 6V power supply to ensure that it can interface properly with our motor driver.

2.4.1 6V Power Supply

The 6V power supply will be a wall plug that will supply power for the motor driver.

Requirement	**Verification**
The battery must supply a the motor driver with a constant 5.8-6.2V.	1. Use multimeter to ensure that the supplied output voltage is within the acceptable range of 5.8-6.2V

	when driven with 2.9A for the motor.
Must be able to supply at least 2.9A of current.	1. Run the circuit with a 100% duty cycle PWM signal. 2. Use a current probe to ensure that at least 2.9A of continuous current is produced.

2.5 Tolerance Analysis

One tolerance that we must observe is the delay time of the different components of our physical system, and how it relates to how fast our control system can calculate the next control signal. Specifically, we must determine both the time that the motor encoder can send the velocity of the motor to our microcontroller, and the time the Tensorflow model takes to calculate the next control signal given the current state from the motor encoder and the potentiometer on the pendulum. We know that the delay of the quadrature sensor that will be used on motor shaft is 13 microseconds. Now we must then ensure that our neural net can learn to balance the pendulum at 90% accuracy given that it is sampling the state of the system at discrete time steps, where each time step is greater than the sum of delay times of the 13 microsecond sensor lag and the Raspberry Pi computation lag.

$$t_{sample\ time} > t_{sensor\ lag} + t_{computation\ lag}$$

We can calculate the sensor lag by looking at the datasheets for our different sensors. We can use the maximum rise time of all our sensor's rise times as our true sensor lag time. For the computation lag, we can upload our tensorflow model into our Raspberry Pi and measure the time it takes from inputting a value to receiving an output from the model. With these two times can determine the definite lower bound for sample delay time that our neural network must be trained on. If our neural net can learn to balance the pendulum with a sample delay time greater than this lower bound, we can guarantee that our system will work at our desired accuracy.

Another aspect of tolerance we must account for is the chance of wheel slippage during the rapid velocity changes of our motor. Although we are going to use the motor encoder to determine the velocity of our system and not position, wheel slippage could possibly throw off our calculations significantly. To handle this situation, we will determine the maximum change in speed that the motor can handle without resulting in wheel slippage. The procedure to determine this maximum acceleration will be to incrementally test the motor with increasing PWM signal jumps. We will use calipers to measure our wheel and gain a precise value for circumference. Knowing the motor gearbox ratio and counts the quadrature encoder can provide per revolution, we will be able to correlate a number of counts from the quadrature encoder to a distance traveled. The theoretical distance per count is 0.226 mm as shown by the calculations below, a value much smaller than the track length of 3 feet and expected travel distances during our ten second runs, and should be easily precise enough for our purposes. We will then compare the distance between the expected start and end positions of the cart based on the data from the quadrature encoder to a measured distance between the cart's start and end positions. Through this process, we will determine the maximum change in velocity that will induce wheel slippage, and use this value to train our neural net as well as limit our PID controller to not force accelerations greater than this max acceleration.

$$\text{circumference} = C = \text{wheel diameter} \cdot \pi = 90 \text{ millimeters} \cdot \pi = 282.7 \text{ millimeters}$$

$$\text{counts per revolution of wheel} = \text{gearbox ratio} \cdot \text{encoder counts per revolution}$$

$$\text{counts per revolution of wheel} = 62.5 \cdot 20 = 1250$$

$$\text{distance travelled between counts} = \frac{C = \text{distance travelled per revolution of wheel}}{\text{counts per revolution of wheel}} = 0.226 \frac{\text{millimeters}}{\text{count}}$$

Another tolerance is being able to accelerate our cart quickly enough to balance the inverted pendulum. Since we are using a motor that can provide far more torque than the motor used in the inverted pendulum project we are basing our design on, we believe achieving the acceleration values necessary will not be a problem. Regardless, we can still calculate an estimate for the maximum acceleration value we will be able to achieve. A high estimate of cart mass is 0.5 kg. Our motor provides a maximum torque at 6V and 2.9 A of stall torque 54 oz-in = 0.3813 Newton-meters. Using a wheel of radius 0.045 m = L, the maximum force we can apply to the cart will be 8.47 N which corresponds to an acceleration of 16.9 m/s^2. Estimating the tire's coefficient of static friction to be 0.5 [9], leads to the cart being accelerated at a maximum of 4.92 m/s^2 as shown by the equations below.

$$F_{\text{cart, max}} = \frac{T}{L} = 8.47 \text{ Newtons}$$

$$\text{Normal Force of cart} = N_{\text{cart}} = m_{\text{cart}} \cdot a_{\text{gravity}} = 4.92 \text{ Newtons}$$

$$\text{Frictional Force tire can provide} = f = \mu_s \cdot N_{\text{cart}} = 2.46 \text{ Newtons},$$

$$\text{since } f < F,$$

$$\text{maximum acceleration of the cart} = a_{\text{max}} = \frac{f}{m} = 4.92 \frac{\text{meters}}{\text{second}^2}$$

2.6 Controller Software

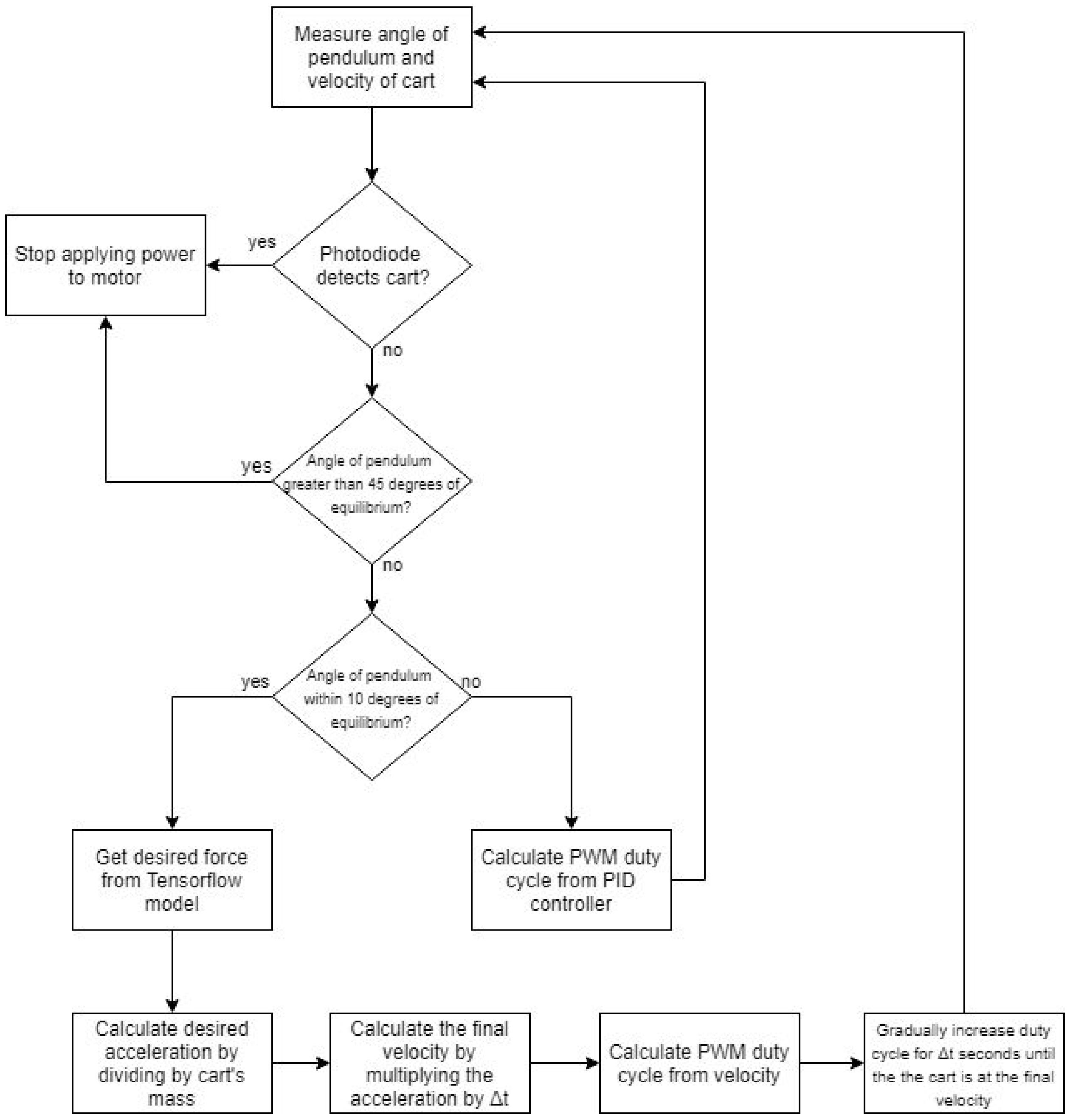
Measure angle of pendulum and velocity of cart
Stop applying power to motor
yes
Photodiode detects cart?
no
yes
Angle of pendulum greater than 45 degrees of equilibrium?
no
yes
Angle of pendulum within 10 degrees of equilibrium?
no
Get desired force from Tensorflow model
Calculate PWM duty cycle from PID controller
Calculate desired acceleration by dividing by cart's mass
Calculate the final velocity by multiplying the acceleration by Δt
Calculate PWM duty cycle from velocity
Gradually increase duty cycle for Δt seconds until the the cart is at the final velocity

Figure 2: Controller Software Flowchart

2.7 Neural Network Model Optimization

Long Short-Term Network (LSTM)

The algorithm we will use to train our agent will be a long short-term network. With an LSTM, our agent can be taught to keep track of past information that could be important. In our design, such a network can be beneficial since remembering past states of the system, including old pendulum angles and velocities can help the agent develop an understand of how the system is changing over time. This can be extremely useful when modeling the lag of the physical system in the training of the network, since the LSTM requires sequences of of experiences to perform training [8].

After we calculate the sum of the lag time of the sensors in the system and the computation lag of the Raspberry Pi, we can set the sequence length of our LSTM to greater than the lower bound of the system lag. This will allow our network to train with a sample delay, and hence allow better control in the physical system. The two most important hyperparameters during training will be the sequence_length and the memory_size. The sequence_length indicates the length of the number of experiences that will be passed through the network while training, while the memory_size indicates the length of an array of floating point numbers keeping track of the past states of the system. Since we only need to infer the velocities of objects in the system, these parameters can be relatively low.

2.8 Circuit Schematics

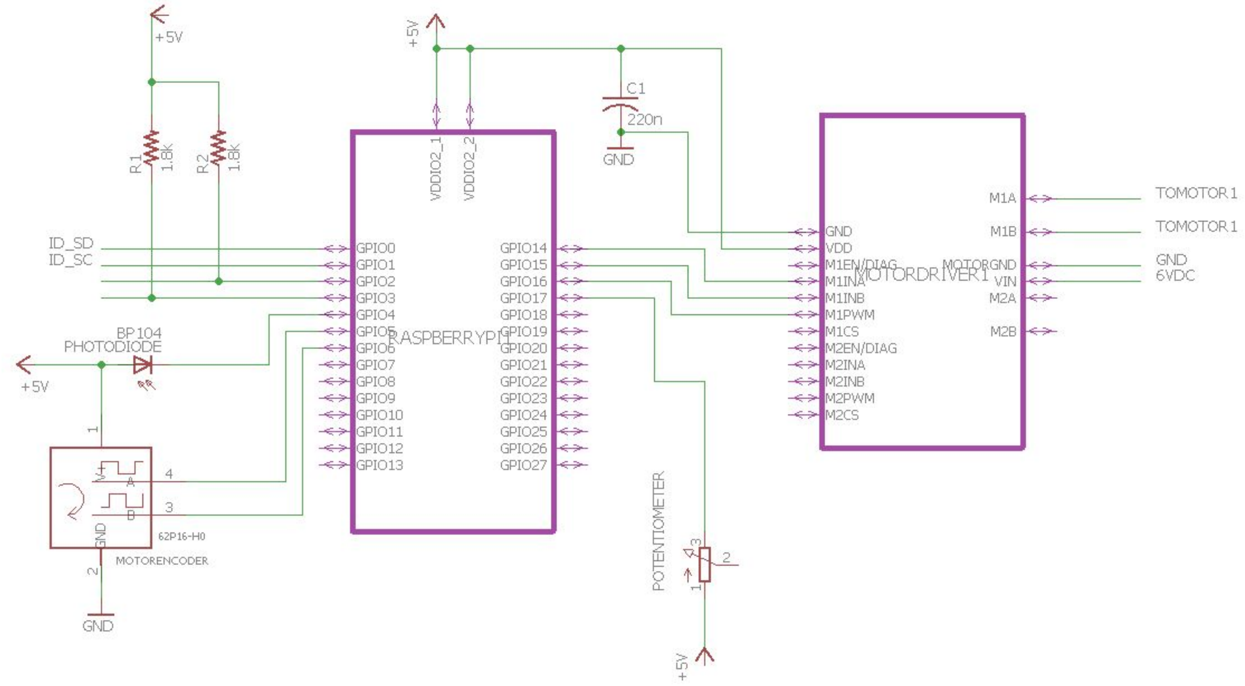

Figure 3: Circuit Schematic

2.9 Physical Design

Our physical design will consist of both 3D printed parts, as well as motors and sensors purchased online. The main cart chassis will be connected to both the track and motor below, as well as the potentiometer connected to the pendulum above. Our design is based off the parts and assembly described by [7]. The simulation in Unity will be modeled such that the masses, lengths, and other physical properties of the system match as closely as possibly. All the below figures have dimensions listed in inches.

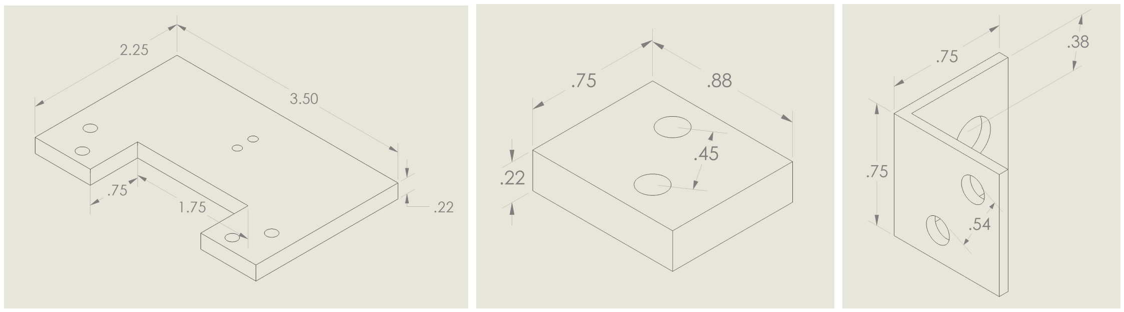

Figure 4: Cart Chassis, Rail Spacer, and Track Holder

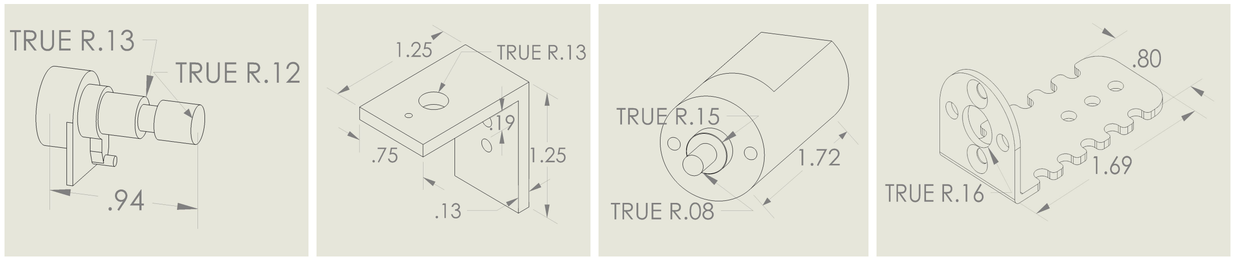

Figure 5: Potentiometer, Potentiometer Holder, Motor, and Motor Bracket

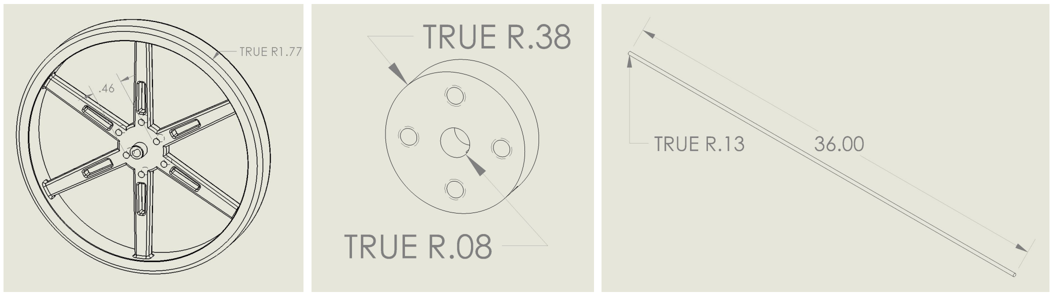

Figure 6:, Wheel, Wheel Hub, and Track

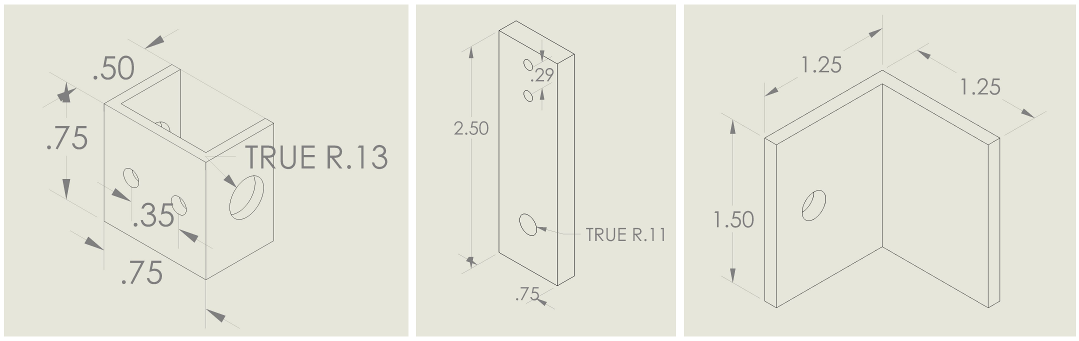

Figure 7: Pendulum Holder, and Potentiometer Connector, and Outer Track Holder

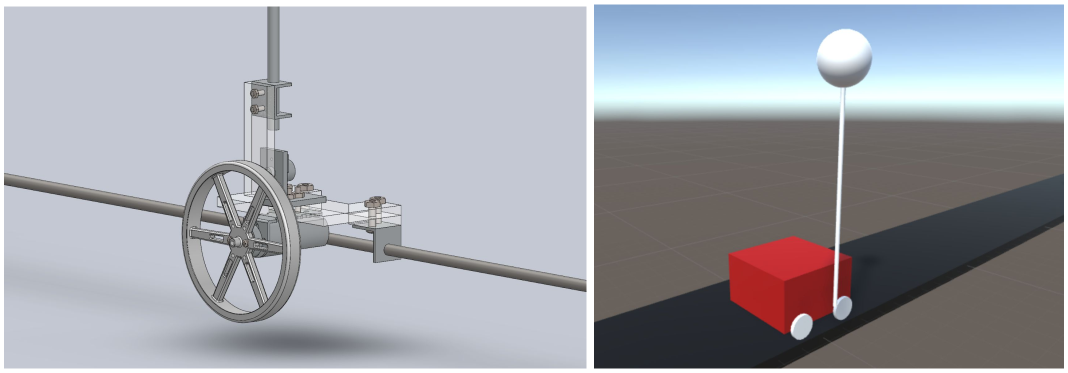
Figure 8: Prototype 3D Model and Unity Simulation

3. Cost and Schedule

3.1. Cost

Our cost comes from labor and parts. For labor, we have 3 people working an estimated average of 10 hours per week each for the 16 week semester at a wage of $35.00 per hour per person.

$$\text{cost of labor } = 3 \text{ people} \cdot 12 \frac{\text{hours}}{\text{week}} \cdot 16 \text{ weeks} \cdot \frac{\$\ 35.00}{\text{hour}} \cdot 2.5 = \$\ 50,400$$

For parts:

Part Description	**Part**	**Cost ($)**
6V DC motor	Pololu 63:1 Metal Gearmotor 20Dx43L mm 6V CB with Extended Motor Shaft, #3714	22.95
Motor Encoder	Pololu Magnetic Encoder Pair Kit, #3499	8.95
Motor Brackets	Pololu 20D mm Metal Gearmotor Bracket Pair, #1138	6.95
Motor Driver	Pololu Dual VNH5019 Motor Driver Shield	49.95
Wheel	Pololu wheel (2 pack), #3691	7.95
Wheel Mount	Pololu Universal Aluminum Mounting Hub for 4mm Shaft, 2 Pack, #1081	6.95
6V Power supply	Triad Magnetics	12.45

	WSU060-3000 6 V, 3 A Power Supply	
Photodiode	TEFD4300-ND	0.74
Cart Chassis	Custom chassis	20.00
Raspberry Pi	Raspberry Pi RASPBERRY PI 3 MODEL B+	35.00
Potentiometer	Potentiometer angle encoder Vishay Spectrol #357	44.48
	Total	216.37

Grand total = cost of parts + cost of labor = $50,616.37

3.2 Schedule

Week	Kishore	Henry	Mason
2/18	Design review prep	Design review prep	Design review prep
2/25	Improve Unity Simulation, develop prototype Tensorflow model to be uploaded to Raspberry Pi microcontroller	Determine all necessary components and chassis design, order components	Determine all necessary components and chassis design, order components
3/4	Further development of Unity simulation and Tensorflow model	Verify that each component fulfills our needs, order chassis	Verify that each component fulfills our needs, order chassis
3/11	Adjust Unity simulation to replicate our physical system	Build physical system	Build physical system
3/18	Spring Break	Spring Break	Spring Break
3/25	Make minor adjustments to simulation to improve Tensorflow model	Build physical system	Build physical system
4/1	Finalize Unity simulations. Obtain final working Tensorflow model.	Make sure each subsystem in physical design is working properly.	Make sure each subsystem in physical design is working properly.
4/8	Prepare for mock demo. Make sure each subsystem is working	Prepare for mock demo. Make sure each subsystem is working.	Prepare for mock demo. Make sure each subsystem is working
4/15	Prepare for final/mock presentation and test project for mock demo.	Prepare for final/mock presentation and test project for mock demo.	Prepare for final/mock presentation and test project for mock demo.
4/22	Prepare for presentation and poster session, complete final paper	Prepare for presentation and poster session, complete final paper	Prepare for presentation and poster session, complete final paper

4. Ethics and Safety

The ethical and safety issues in our project mainly pertain to the safety of the different moving mechanical components in the design. Since there is movement of a cart on a track and the attached swinging pendulum, this poses a safety risk for anyone standing too close or putting their body parts in the system. During operation, the cart and pendulum could cause injury to someone too close to the system. Additionally, our electrical components could cause harm to someone upon being mishandled.

The safety precautions we would take to handle these situations refer to #9 on the IEEE Code of Ethics, to “avoid injuring others, their property, reputation, or employment by false or malicious action“ [4]. To prevent the cart from hurting someone’s fingers, we would add a rubber bumper to each side of the cart, as well as bumpers to the end of the track to prevent the cart from flying off. We would also ensure that the heavy end of the pendulum is rounded and not sharp, to avoid the risk of serious injury if it was to strike anyone standing too close. We will also have a safety mechanism to prevent any outputs of the control system which ask for too high voltages or forces.   In such a situation, we will ignore the output of the model and choose a more appropriate signal.
In addition, since we are dealing with relatively high voltages to power the motor, we need to make sure the circuits are properly insulated so that no can get hurt by accidentally coming into contact with the system.

References

[1] Clark, Jack. “Robots That Learn.” *OpenAI Blog*, OpenAI Blog, 28 Nov. 2017, blog.openai.com/robots-that-learn/.

[2] Juliani, Arthur. “Introducing: Unity Machine Learning Agents Toolkit – Unity Blog.” *Unity Technologies Blog*, 19 Sept. 2017, blogs.unity3d.com/2017/09/19/introducing-unity-machine-learning-agents/.

[3] OpenAI. “Proximal Policy Optimization.” *OpenAI Blog*, OpenAI Blog, 20 July 2017, blog.openai.com/openai-baselines-ppo/.

[4] “IEEE Code of Ethics.” *IEEE - Advancing Technology for Humanity*, www.ieee.org/about/corporate/governance/p7-8.html.

[5]”Average Entry-Level Electrical Engineer Salary”, Payscale, www.payscale.com/research/US/Job=Electrical_Engineer/Salary/6fd28da9/Entry-Level

[6] “TensorFlow Lite | TensorFlow.” *TensorFlow*, www.tensorflow.org/lite.

[7] “Inverted Pendulum Project.” *Andy's Log*, 23 Jan. 2016, andreweib.wordpress.com/inverted-pendulum-project/.

[8] Unity-Technologies. (n.d.). Unity-Technologies/ml-agents. Retrieved from https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Feature-Memory.md

[9] Friction and Friction Coefficients. (n.d.). Retrieved from https://www.engineeringtoolbox.com/friction-coefficients-d_778.html