Virtually Trained Self-Balancing Pendulum

ECE 445 Design Document

Henry Thompson, Kishore Adimulam, and Mason Ryan
Group 31
TA: Amr Martini
2/17/2019

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 AI 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, there has been no past projects translating AI models trained in the Unity Game Engine to real physical systems.

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 which then would gain the ability to balance the pendulum. Specifically, 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.

1.2 Background

The ability of virtual reality to model physics and interactions between materials positions it to become an ideal tool for stimulating 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. Such an agent trained in a virtual simulation could be uploaded into an autonomous vehicle or factory robot, for example. 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]. We aim to create a solution where the Tensorflow model of an agent trained on a 3D simulation in Unity can be deployed into a microcontroller and perform in a physical replica of the system.

1.3 High Level Requirements

- The agent in the Unity simulation must be able to successfully learn to balance the pole
- The agent in the physical system must be able to balance the pole with no prior experience outside of the simulation
- The control signals in the physical system must be bounded in some way to prevent excessive or unsafe voltages/forces on on the motor

2. Design

There are four major subsystems in our project: the machine learning module, the control module, the mechanical module, and the power supply module as shown in figure 1. The machine learning module contains the Unity 3D simulation and the Tensorflow model produced from the simulation. The model will be uploaded to the control module where it can determine the appropriate control signals to send based off of this data. The control module consists of our microcontroller, on-chip memory, position sensor, and our rotary sensor. The microcontroller will take in the sensor and Tensorflow model data, and will then determine the PWM rate it should supply to the motor in the physical system. The physical system consists of motor, cart and pendulum. It will be controlled by the microcontroller, which will feed it a PWM signal to the motor through the voltage amplifier. The pendulum on the cart will then move according to the cart's velocity. The power supply module will supply power to the microcontroller, sensors, and the voltage amplifier. The amplifier is needed to provide enough voltage for the motor, since the microcontroller has only a limited power output.

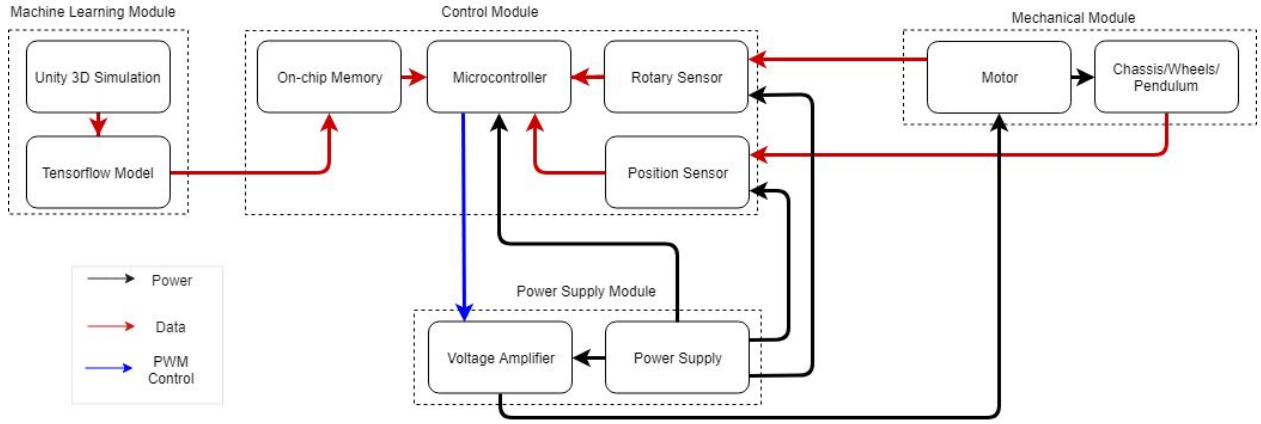

Figure 1: Block Diagram

(Input PWM rate into the microcontroller, not voltage to motor)

The physical design, shown in figure 2, will consist of a pendulum attached to a cart. The cart will be restrained to a one dimensional track. The Raspberry Pi and rotary sensor will be attached to the cart and they will be powered externally. The system can be modeled by a typical cart-pole control equation, Eq.1, where γ is the viscous friction, J is the total angular momentum of the system, F is the force applied to the cart, $\boldsymbol{\theta}$ is the angle of the pole from its equilibrium position, l is the length of the pole, and g is the acceleration due to gravity.

$$\ddot{\theta} + \frac{\gamma}{J}\dot{\theta} - \frac{mgl}{J}\sin(\theta) - \frac{l}{J}F\cos(\theta) = 0 \qquad \text{Eq.1}$$

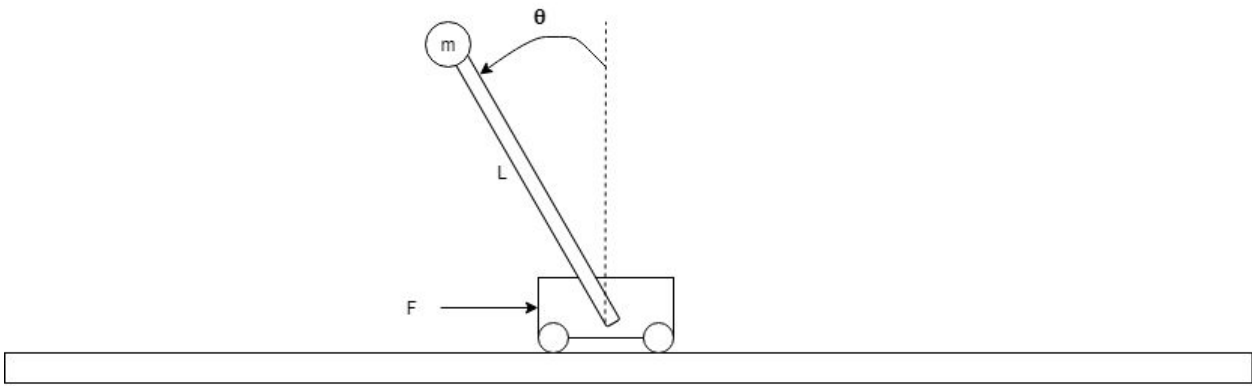

Figure 2: Physical design sketch

2.1 Machine Learning Module

2.1.1 Unity Simulation

Requirement	Verification
● Simulation must hit hit a target performance rate ● Ensure the force generated by the model needed to balance the pendulum will not surpass max voltage input for the motor (12V)	● Given a particular microcontroller τ, with the upper and lower bounds on the deviation $\|\Phi\tau - \Phi^*\|$, (where $\Phi\tau$ is the state space of the pendulum angle on the system using the microcontroller and Φ^* is the equilibrium angle when the pendulum is upright), check that $\|\Phi u - \Phi^*\|$ (where Φu is the state space of pendulum angle in the Unity simulation) is sufficiently small enough that the added deviations $\|\Phi u - \Phi^*\| + \|\Phi\tau - \Phi^*\|$ still results in a stable system ● Measure the force applied by agent throughout the course of simulation, check that the resulting voltage given the force doesn't exceed 12V

2.1.2 Tensorflow Model

Requirement	Verification
● Tensorflow model needs to be small enough to be run under $\boldsymbol{\tau}$ delay and fit in RAM	● Check that the model can run fast enough to generate control output through some function $d\Phi/dt = f(\Phi,u,\boldsymbol{\tau})$. Simulate in Unity or MATLAB how sensitive the system is to delay and check if bound of deviation $\|\Phi - \Phi^*\|$ are finite for all time t.

2.X Tolerance Analysis

One tolerance that we must obverse is the delay of the Raspberry Pi, τ, as it relates to how fast it can calculate the next control input. We can determine how sensitive our system is to a given delay by simulating our system with some function $d\Phi/dt = f(\Phi,u,\tau)$, in Unity or MATLAB, where Φ is the current pendulum angle, u is the control signal, and τ is the delay.

We must also ensure that accumulating sources of error in our measurements of position and velocity of the cart, as well as measurements in the angle and angular velocity of the pendulum don't lead to our model unable to correctly predict the right control signal.

Finally, we must observe the tolerance for when we will accept values output from the model as true. If the model asks for a force or voltage that is too high, we must override those decisions with a safety mechanism and choose a more appropriate action.

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 motor from hurting someone's fingers, we would add a rubber bumper to each side of the motor, as well as bumpers to the end of the track to prevent the motor 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.