Virtually Trained Self-Balancing Pendulum

Group 31: Kishore Adimulam, Mason Ryan, and Henry Thompson
TA: Amr Martini

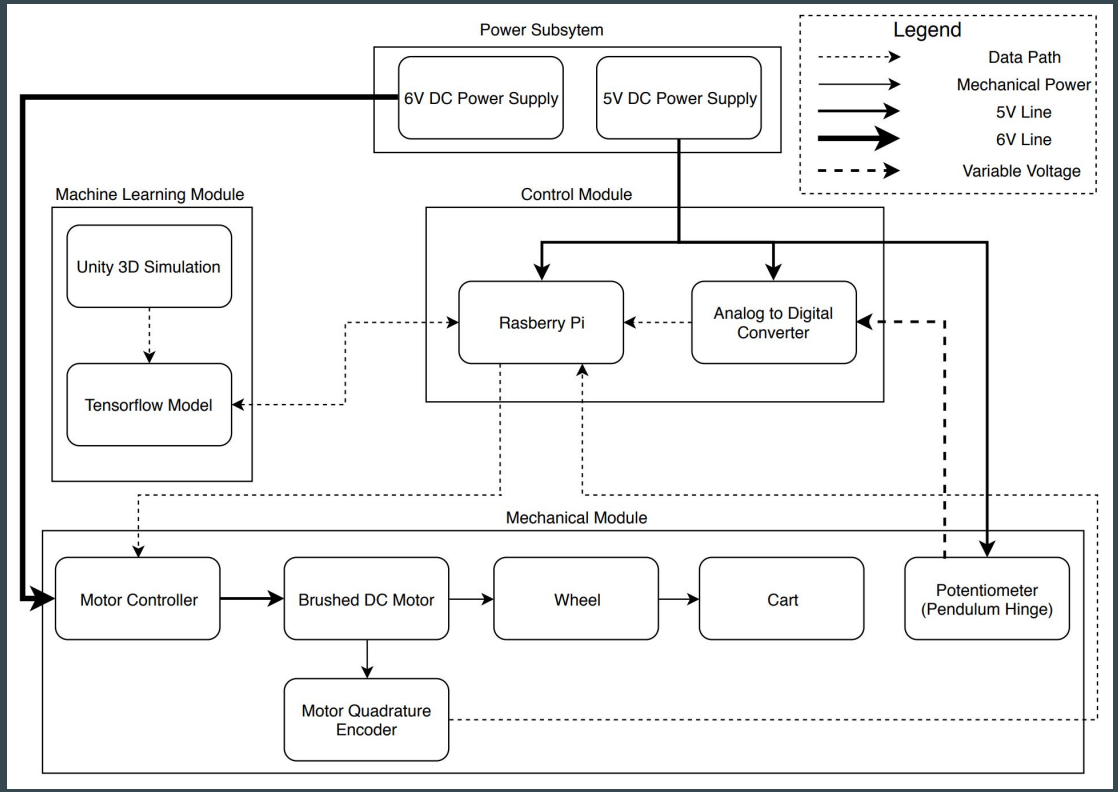
Introduction

- There is a growing use for virtual reality as a training environment for AI for real world applications
- Some game engines have machine learning tool-kits for training reinforcement learning algorithms inside games and simulations
- Not much work in translating game engine simulation-trained models to physical systems has been done

Objectives

- Create a self-balancing inverted pendulum system which uses a control algorithm generated by ML Agents in Unity
- Establish a workflow 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

Block Diagram

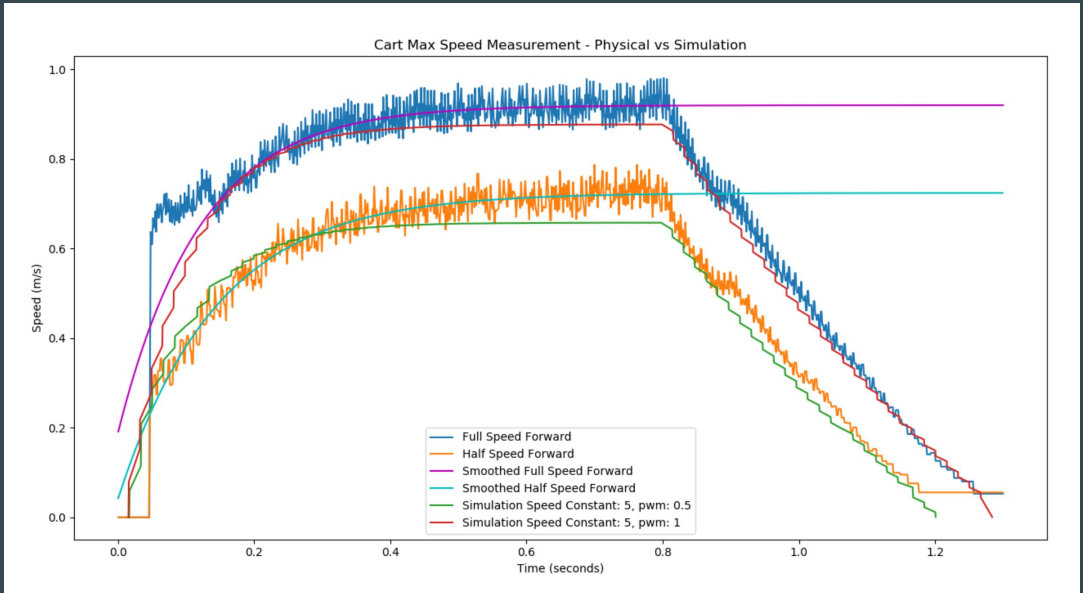


Modeling Physical System Dynamics

- **Cart motion**
  - Apply max speed to motor in physical system and simulation to determine force constant and air drag constant needed for cart
- **Pendulum hinge friction**
  - Drop the pendulum from 5 degrees from vertical in physical system and simulation to determine hinge friction constant

Since all constants determined are based off imperfect measurements, apply random normal noise in order to train agent to balance on a range of different values.

Modeling Physical System Dynamics - Speed Test



- Best fit line to exponential function

$$y = A - Be^{-Cx}$$

  for rising velocity of motor, fit using scipy optimize function

- Applied resistance torque in motor shaft to cart in simulation to induce linear velocity decrease

Conclusions

- Sensors have accurate readings and motor has reliable control of system
- Back PID control system works well to balance the pendulum given small disturbances while at equilibrium
- Unity simulation works well to balance the pendulum given random normal noise for all measured system constants

Since the Tensorflow model outputted from Unity cannot be uploaded directly into Raspberry Pi, secondary neural net needs to be trained to output more accurate floating-point values for precise motor control (currently at 65% accuracy for learning output rounded to 1 decimal point).