Sim2Real - Reinforcement Learning to Hardware Platform for Cartpole Environment

Grace Kim

Abstract

A major benefit of using Sim2Real (Simulation to Real) methodology is the significantly reduced cost of being able to test and train hardware platforms. This is through the use of simulations rather than real-world experimental setups to continuously generate data and knowledge to train a hardware platform. However, if the simulation environment used to generate this data is flawed, there is bound to be a disconnect when trying to feed this new knowledge back into the hardware setup. These misrepresentations of hardware within simulation is what leads to the Sim2Real gap, which makes it difficult to move forward with implementing Sim2Real for specific systems. In this work, the Sim2Real methodology was applied onto the cart pole problem, to explore the dynamics and tradeoffs between preserving simulation performance and modeling as closely to the hardware as possible. Through a student built hardware platform, a DQN Reinforcement learning agent was trained to mimic as closely as possible the hardware platform (both its pros and faults) to bridge the gap between Sim2Real.

1 Introduction / Background

The concept of simulation to reality, or shortened to Sim2Real, has been an exciting topic applied to multiple areas within robotic in-hand manipulation, computer vision, autonomous vehicles, and many more [1]. The principle is based on using various forms of artificial intelligence, often training simulated environments through reinforcement learning, to learn skills that can be transferred into real world hardware or applications. However, despite the incredible advancements of numerous deep learning algorithms and simulation platforms, there are still many areas of disconnect and difficulty to bridge the gap between simulation and reality [2].

Several main pain points identified in the literature include the difficulties modeling the complexities found within physics, such as non-linear dynamics caused by factors like friction and deformation [3]. In addition, various simulations are often times based off of discrete time elements, rather than trying to emulate a continuous scope similar to their hardware twin setups[2]. Some have tried to address this using hybrid simulation learning, or using imitation learning from real life examples.

It is proposed to tackle the lack of model complexities by incorporating as many "faults" found within the physical hardware into the simulation model itself. For this project, the cart pole environment will be explored to apply this experimentation.

First, the hardware platform will be developed, and discrepancies between the simulation and hardware will be identified. The main differences would lie in the imperfections of the hardware system, which will need to be emulated in the simulation environment as closely as possible. By introducing these hardware defects into the simulation's perfect environment, it may be possible to work towards bridging the sim2real gap that exists in many applications today.

Second, the simulation environment will be thoroughly analyzed and iterated over to find the best hyper parameters for the environment setup we have. With the best hyperparameters, and then the incorporation of the flaws found within the hardware (latency issues, sensor misreadings, etc) the

Preprint. Under review.

simulation environment can then be inputed back into the hardware for testing and completing the sim2real gap.

1.1 Cart Pole

For this study, the cart pole problem was selected to test the Sim2Real pipeline. Large precedence for hardware cart pole designs can be found within the realm of controls, as the task is often used to test classic control algorithms such as Proportional–Integral–Derivative (PID) controllers [4]. In addition, many libraries exist such as OpenAI gymnasium that provide cart pole as a reference simulation environment for training reinforcement learning algorithms [5].

The general cart pole problem can be summarized as follows. An inverted pendulum, in this case a pole, is set to balance above a cart. This cart can only move horizontally back and forth in a linear motion, and the motion is generally set to be frictionless. A simplified diagram of the problem can be found in Figure 9 in the appendix. The important parameters in the simulation space are also defined in Table 1 in the appendix.

The appendix also carries several notes on the equations of motion used for both the hardware and simulation platforms.

2 Methodology

2.1 Hardware Development

To design the hardware of the cart pole problem outlined in background section 1.1, a systems diagram was first developed, shown in Figure 1, to determine all the electronic components necessary to build the system. Starting from the sensors, a MPU5060 6-Degree of Freedom (DOF) Inertial Measurement Unit (IMU) module was used to determine the angle of the pendulum pole θ, and the angular velocity $\dot{\theta}$ of the pole itself. Having as precise of a measurement for θ and $\dot{\theta}$ were crucial, especially as those dictated the next state of the cart pole in the equations of motion 1 and 2. However, the MPU5060 unit only provided gyroscope and accelerometer data, which corresponds to having only our our angular velocity $\dot{\theta}$. In order to calculate and determine the angle of the pole itself, an integration needed to be performed over the angular velocities to obtain our angle θ.

The distance of our cart in x and our linear velocity $\dot{x}$ was then determined by using two US-100 Ultrasonic Distance Sensors. One was positioned on each side of the cart, both for redundancy, and also to check if the cart was reaching the end of the railing/platform. As the US-100 sensor provides direct readings in millimeters of how far away the end of the platform was in x, the linear velocity $\dot{x}$ was calculated by performing a derivative over the consecutive x values over periods of 50 milliseconds.

Initially, all of these readings were planned to be inputted directly to the Raspberry Pi 4 Model B, so that there could be one on board computer that could operate the entire cart pole system. However, after completing multiple initial trials, it was clear that the Raspberry Pi was not suited to handle the high volume and precise sensor readings needed of the system. Many noisy spikes of readings could be found at sporadic moments in time when operating both the MPU5060, and the US-100. In order to ensure accurate and consistent data readings from the sensors, an Arduino Uno was used as an intermediary for sensor readings due to it's adeptness in interfacing with analog inputs [6]. Every 300 milliseconds, the Arduino Uno will collect sensor inputs from the MPU5060 and both US-100s, arrange them in a pre-formatted string and send the sensor data continuously over a direct serial line connection to the Raspberry Pi. This was the shortest possible latency that could be managed with the current setup, due to the time needed to calculate the various integrations and derivatives for $\dot{x}$ and θ.

Then, after receiving the sensor inputs, the Raspberry Pi can load in a pre-trained RL algorithm and feed in the current state s of the hardware to obtain an action a to perform on the current hardware to balance the pole. Receiving this action a from the reinforcement learning algorithm, the Raspberry Pi can then feed this action to the motor, which will then move the cart in the direction specified to balance the pendulum pole. This is a cyclic process, working to keep the pole balanced for as long as possible. However, the termination cases are if the pole falls past a threshold angle, or it hits the physical boundary railings of the cart setup itself.

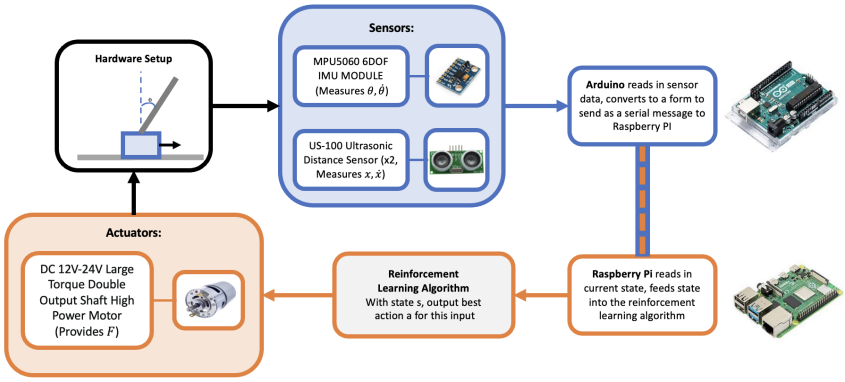


Figure 1: Systems diagram of the final hardware setup developed for the cart pole system. The diagram can be seen to first start at the hardware sensor readings, determining the current values of θ, $\dot{\theta}$, x, and $\dot{x}$. These are read in through an Arduino, which sends the state data through a direct serial connection into the Raspberry Pi. The Raspberry Pi then feeds the state s into a pre-trained RL algorithm, receives an action a, and then feeds this action command into the motor to move the cart and balance the pole. This cycles over and over until the pole falls, or reaches the limits of the physical barriers.

It is noted that these specific electronic components were selected as these were the sensors and actuators already available at the student's residence for this project.

In addition, below in the appendix, setup photos of the hardware system and various components can be found, detailing a bit further into the hardware design.

2.2 OpenAI Environment and Reinforcement Learning

After having the completed hardware setup, a Deep Q Learning (DQN) technique was used for the Reinforcement Learning (RL) algorithm. The general process flow for training the DQN agent is shown in Figure 2, followed from [7]. There are two neural networks, one main policy network used for major action selection and optimization processes, and a target network that gets updated with soft versions of the new policy weights each episode. Most work is completed on the policy neural network, where in each around an action is chosen based on the state given. Depending on the progress of the training, the action selection should be more exploratory in the beginning, and exploitative towards the end. With this action, the simulation steps through the dynamics of what would occur when applying this action, and save this within the replay memory. Once there are enough samples within the replay memory, full batches will be sent to the optimizer to optimize the weights of the policy's neural network. The use of this replay memory allows for a more data driven approach rather than goal driven manner. For the optimizer itself, Huber loss is used as the loss function within the regression process. When the error is small, the Huber loss exhibits similar behavior to the mean squared error loss function, but for large errors instead takes the approach of mean absolute error. This provides robustness when outliers or Q in general becomes noisy [7].

All of this is trained on the cart pole environment on the OpenAI gymnasium library. Within the cart pole environment, several key things must be noted. First, this is a continuous reinforcement learning problem as the agent's reward function is based on how many incremental time-steps the pole can balance on the cart for. As long as the simulation does not terminate, a reward of +1 is provided for every step taken. The conditions for terminating the simulation are similar to the hardware, so that there are two conditions of termination. First, if the pole falls too far from the center, further from the specified angle of $12\,\mathrm{deg}$, the simulation fails. In addition, if the cart moves too far from the center of the platform, past 2.4 meters, the simulation also fails.

In order to find the best assortment of hyperparameters for this setup, a parameter study of four different variables were looped upon: γ the discount, ϵ_{start} the starting rate of decay for level of explorative actions we take, ϵ_{stop} the stoping rate of decay for level of explorative actions we take, and the learning rate. The results of the parameter study can be found in Figure 3.

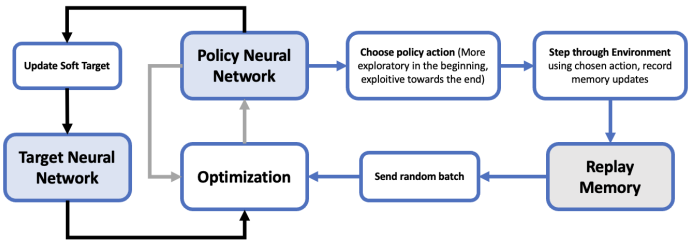


Figure 2: Training flow diagram of the DQN reinforcement learning agent. Starting from the untrained policy neural network, a policy action is chosen, and sent to the environment to step through and run save within the replay memory. Then, a random batch from this replay memory is then sent to the optimizer to train the policy neural network using a Huber loss function. Once optimized, the Policy Neural Network sends its optimized state dictionary into the Target Neural Network, which gets soft updates dictated by a hyperparameter τ.

These hyperparameters were chosen to be tuned due to references such as Andrychowicz [8], which state the discount factor and learning rate as the most important parameters to tune in a reinforcement learning simulation. As the learning rate determines how fast the simulation will learn the optimal weights, too fast of a rate may cause the agent to miss better solution spaces. For the discount factor, we can ensure the rewards are adjusted over time, so that the agent does not only seek immediate rewards but instead try to work towards the long term goal.

Along with these two hyperparameters, the values of ϵ_{start} and ϵend were also investigated to find the effects of having a more exploitative V.S. explorative approach to learning. Although the action space available to the cart pole contains only two states each round: move left or to move right, the incremental buildup of these actions over time creates limitless number of possibilities within this action space, making decisions much more complex. The specific choices the agent makes over these initial decisions can therefore be approached in either a exploratory or exploitative manner by changing ϵ_{start} and ϵ_{end}. A larger ϵ for either the start or finish indicates a more explorative manner of action selection in the simulation, and a smaller ϵ indicates a more exploitative behavior.

2.3 Integration of Simulation to Hardware

Now, having developed both the hardware and simulations, the two can now be linked to complete the sim2real pipeline. However, when approaching this, several key differences between the hardware and simulation were found that needed to be resolved.

First, time scales of the simulation to hardware were not consistent, introducing a form of latency within the system. Each time step in the simulation was listed as 20 milliseconds, while the hardware ranged from 300-500 milliseconds depending on latency. Tests were performed to see incorporation of delay within the sensor readings to tackle this.

In addition, sensor readings for the hardware platform was relatively noisy compared to the simulation. Additional randomized noise error experiments were also conducted to tackle this and is outlined in the results section.

Finally, the simulation platform was made to reflect more accurately the scale of the actual student hardware, and different x limits were tested to see if the cart could still be balanced without having access to a larger actuation space.

3 Results Discussion

Before conducting the Sim2Real tests needed for the Cart Pole platform, first a hyperparameter study was completed to determine the most suitable learning configurations of the simulation.

Using those selected parameters, an iterative test and repeat process was then completed to understand the main differences between the simulation and hardware setups, to ultimately reach a successful Sim2Real transfer.

3.1 Simulation - Hyperparameter Studies

As mentioned in section 2.2, four hyperparameters were explored for testing: γ the discount, ϵ_{start} the starting rate of decay for level of explorative actions we take, ϵ_{stop} the stoping rate of decay for level of explorative actions we take, and the learning rate.

Within the hyperparameter studies run for the OpenAI Cart Pole environment, it was determined that the most stable and consistently rising learning rate in Figure 3a was 0.0001. This value was then used for all following simulations. The 0.001 learning rate also had a high performance, but the step count for balancing the pole seemed too sporadic to be reliable, with many outliers remaining even after what had seemed to look like it was reaching convergence. This led to the use of 0.0001 as the learning rate even though it had a slower growth trajectory. All other learning rates did not have sufficient growth to be justified for use. For the smaller learning rates of 0.1 and 0.01, both seem to have unable to converge from too rapid of a learning rate. On the other hand, the other extreme of 0.00001 had been too resistant to changes in the policy which led to lack of convergence.

For the discount factor, the best value was found to be straight in the middle at 0.5, 3d which then led to $\gamma = 0.5$ being used for all future runs. When testing the extremes of γ, with values such as 0.99 and 0.05, both sides tended to be much less successful for training longer sessions.

Several tests for the ϵ_{start} and ϵ_{end} hyperparameters were also conducted to gain an understanding of the exploitation vs. exploration balance needed for the simulation. After the parameter sweeps, it was clear that a larger ϵ_{start} was more beneficial to the overall learning pipeline, providing faster convergences at higher rewards. A larger ϵ_{start} indicates a more explorative start to the simulation, which correlates with the finding in Figure 3d. This led to all future tests run with 0.9 for this parameter. For ϵ_{end}, having a much smaller value ensured that convergence would be reached within the system. A smaller ϵ_{end} corresponds to having more exploitative behavior, which would be preferred once the policy agent has a better grasp of the best actions to run within the simulation. ϵ_{end} was set to 0.05. This test can be found in Figure 3b.

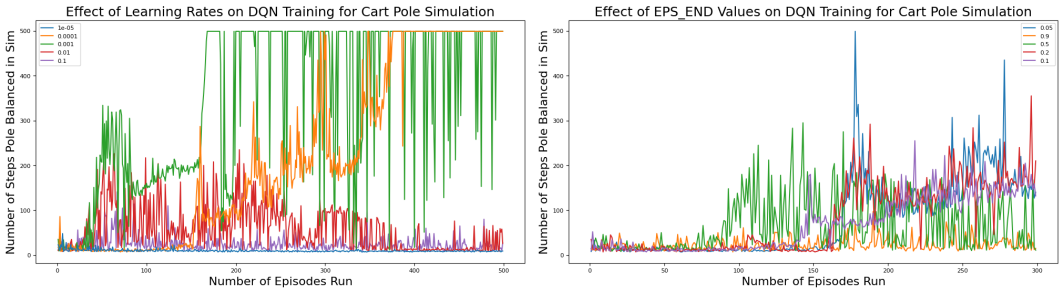
3.2 Building Simulations Closer to Hardware

In a perfect world, with the trained policy neural networks from the hyperparameter studies in section 3.1, the results of the simulation should be able to perfectly match and predict the results seen within the real world. Unfortunately, this is not generally the case when moving trained simulation results from a model onto hardware.

For the current setup with the hyperparameter testing, the cart pole simulation is emulating a near perfect environment. However, as referenced in section 2.3, there are many disconnects between the model and phenomena seen in real life, which will be tested for in this section.

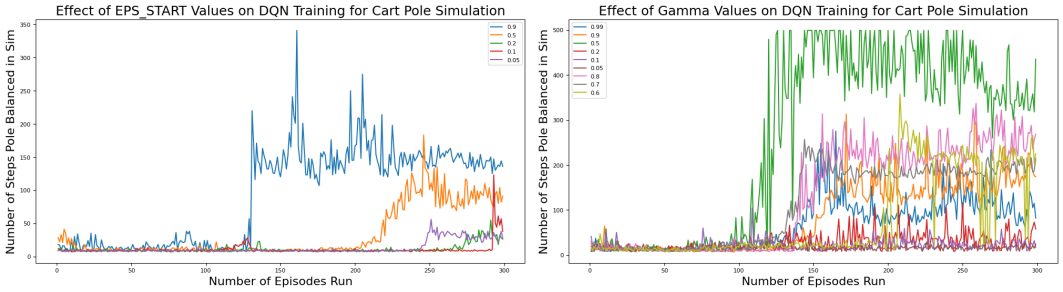
First, there is the issue of latency. In the hardware setup, there are delays in relaying the state information from the sensors to the Arduino Uno to the Raspberry Pi, and the delays in communication are not necessarily consistent. The hardware platform will always have a built in latency due to the design of the sensor and actuator connections to the on board computers. However, the current state of the simulation always has instant feedback of the positions of the cart and pole. Because of this, the first simulation to hardware test completed was incorporating several levels of delays (defined by waiting multiple steps ranging from 1-6) in the simulation itself. The results can be found in Figure 4a, where the only well converged simulation was the case with no delay in the simulation at all (the blue line). All other simulations with at least one or more step delays in receiving state inputs failed to make it past 100 steps in most test cases. To illustrate this further, Figure 4b has the state angles recorded from the last episode trained from each of these situations. As shown in the figure, the states with more than 1 or 2 delays all were under 50 steps in their simulation runs, corresponding to barely 1 second on the cart pole environment. It is clear even the simulator has a difficult time navigating and controlling the environment with this added delay, which provides much more insight into the difficulty of translating simulation to real hardware platforms.

In addition to issues regarding sensor delays, there are also difficulties with faulty sensor readings that must also be taken into account into the simulation. Due to voltage spikes and integration errors, the MPU5060 IMU sensor in particular tends to give results that are not very consistent for angular readings, especially for high accelerations that cause the cart to move incredibly quickly back and forth.



(a) Plot of number of steps the pole was able to balance for each episode run, testing the learning rate hyperparameter for the DQN reinforcement learning agent. Best result was 0.0001 learning rate.

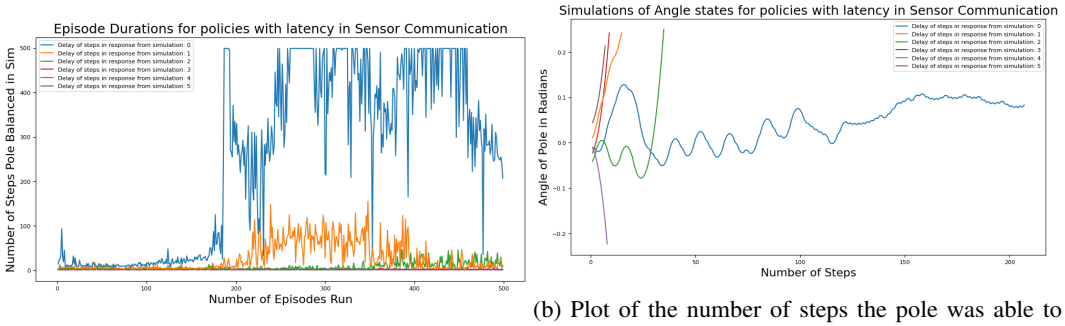
(b) Plot of the number of steps the pole was able to balance for each episode run, with changing the hyperparameter ϵ_{end} for the DQN reinforcement learning agent. Best was 0.05 ϵ_{end} for the result.



(c) Plot of the number of steps the pole was able to balance for each episode run, with changing the hyperparameter ϵ_{start} for the DQN reinforcement learning agent. Best value for ϵ_{start} was 0.9.

(d) Plot of the number of steps the pole was able to balance for each episode run, with changing the hyperparameter Gamma for the DQN reinforcement learning agent. Best value for gamma was 0.5 (green line).

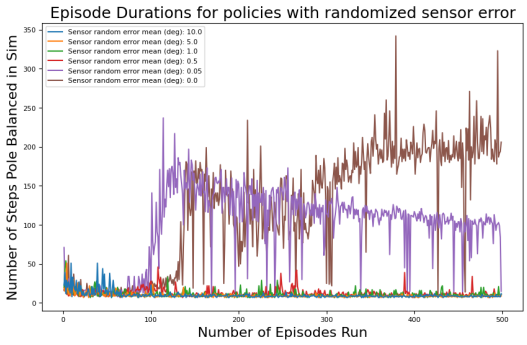
Figure 3: DQN Reinforcement Learning agent hyperparameter study



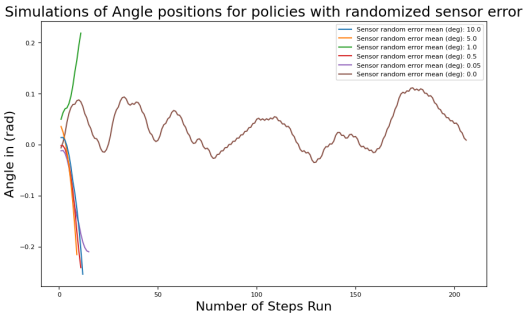
(a) Plot of number of steps the pole was able to balance for each episode run, testing effects of delay within the simulation for the DQN reinforcement learning agent. Light blue is no delay, while everything else has several levels of delays incorporated within the steps.

(b) Plot of the number of steps the pole was able to balance and the angle of the pole in radians to show general pole movement when using one of the trained policies. Other than the initial policy with no delay in the situation, all other tests seemed to struggle to keep the pole upright past 50 steps, corresponding to 2 seconds.

Figure 4: Introducing Latency into RL cartpole simulation environment, incorporate multi step delays in sensor communication

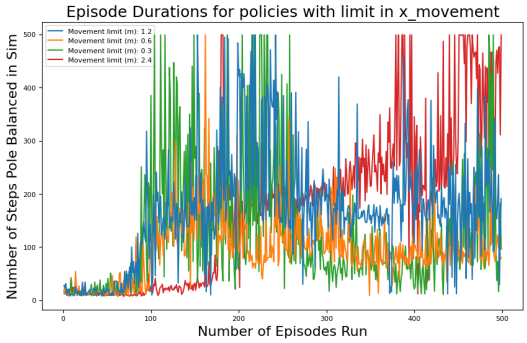


(a) Plot of number of steps the pole was able to balance for each episode run, testing effects of randomized sensor error incorporated within the simulation for the DQN reinforcement learning agent.

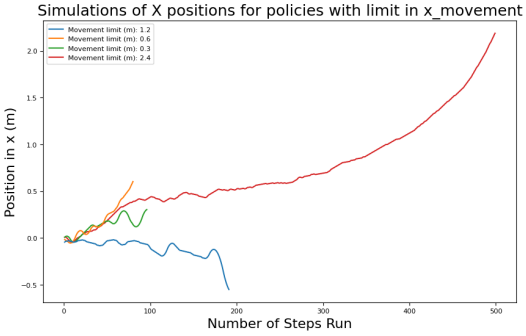


(b) Plot of the number of steps the pole was able to balance and the angle of the pole in radians to show general pole movement when using one of the trained policies. Other than the initial policy with no delay in the situation, all other tests seemed to struggle to keep the pole upright past 25 steps, corresponding to 1 second.

Figure 5: Introducing randomized sensor error into RL cart pole simulation environment, incorporate multi step delays in sensor communication



(a) Plot of number of steps the pole was able to balance for each episode run, testing effects of shortening track from 2.4m on each side, to 1.2m, to 0.6m, to 0.3m to match the student track.



(b) Plot of the number of steps the pole was able to balance and the position of the of the pole in meters to show general pole location when using one of the trained policies. For this case, most tests were able to reach 100 steps, which is close to 2 seconds.

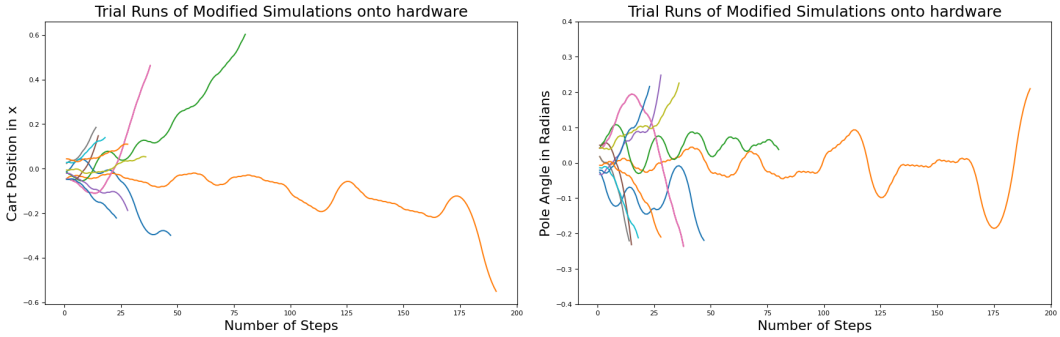
Figure 6: Introducing shorter track into RL cart pole simulation environment, making as similar to student environment as possible.

This was incorporated into the simulation by adding in a random noise element to the state angle readings, ranging from a normal distribution with a mean value between 0 and 10. The final results can be seen in Figures 5a and 5b. Similar to the previous test, the simulation does much worse with the incorporation of these sensor errors, with only the policy with no sensor random errors incorporated into it succeeding to get past 100 steps.

One final test completed to bridge the simulation to hardware gap was the limitations on the size of the track. While the original track has over 4.8 meters to move the cart in between, the hardware setup developed by the student was limited to at most movement in 0.6 meters on both sides. Multiple simulation runs were completed with varying lengths of track, and results are shown in Figure 6. For this case, most simulations were able to reach successful completion to 100 steps within the simulation, even the track that was 0.3m long. Finding the right place to stop the simulation will be important to ensure the best policy is shared onto the hardware setup.

3.3 Final Implementation of Simulation into Hardware

With all of these iterations, the final simulation to hardware can be pipeline can be completed. A policy taking the smallest value of each of the latency, sensor error, and x movement limitations in



(a) Using new modified simulation policy, testing how well pole was able to balance over the various trial runs, plotting number of steps against the cart position in x.

(b) Using new modified simulation policy, testing how well pole was able to balance over the various trial runs, plotting number of steps against the cart angle y.

Figure 7: Final simulation to hardware studies of fully modified simulation policy that takes into account hardware restrictions.

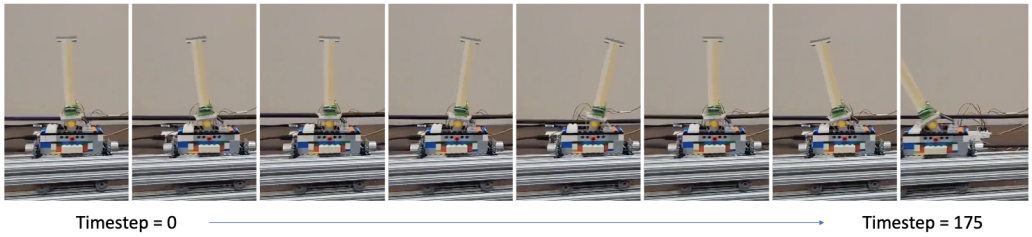


Figure 8: Frame by Frame still shot of the pendulum moving with the modified simulation policy.

the previous tests was trained. These became values of: 1 step delay, 0.05 sensor error noise, and a track size of 0.6 meters. The training episode run data is shown in the appendix 12.

This policy was run a multitude of times on the student platform, and most runs were under 50 steps or less (1 second). One simulation run reached close to 180 steps, almost making it to 4 seconds of balancing. The results of the state angle and position during these runs can be found in Figure 7a and Figure 7b. A frame by frame stillshot is also included for reference of the 180 step balancing trial in Figure 8.

4 Conclusion

As seen throughout this project, there are still many discrepancies and differences when transferring knowledge from hardware to software. In this project, a partial transfer of knowledge was successfully completed, taking into account the many differences of the software and hardware. It is hoped that further sim2real developments can be approached in the same manner to bridge the sim2real gap.

References

[1] Simsangcheol. (2023) Sim2real. [Online]. Available: https://medium.com/@sim30217/sim2real-fa835321342a#:~:text=Sim2Real%2C%20short%20for%20%E2%80%9Csimulation%20to,environment%20to%20real%2Dworld%20applications.

[2] S. Höfer, K. Bekris, A. Handa, J. C. Gamboa, M. Mozifian, F. Golemo, C. Atkeson, D. Fox, K. Goldberg, J. Leonard, C. Karen Liu, J. Peters, S. Song, P. Welinder, and M. White, "Sim2real in robotics and automation: Applications and challenges," *IEEE Transactions on Automation Science and Engineering*, vol. 18, no. 2, pp. 398–400, 2021.

[3] T. Blüher, H. Billiet, and R. Stark, "Model building for better transfer of ai systems using reinforcement learning from simulation to the physical world," *Procedia CIRP*, vol. 109, pp. 113–118, 2022.

[4] F. Mrad, N. El-Hassan, S.-H. Mahmoud, B. Alawieh, and F. Adlouni, "Real-time control of free-standing cart-mounted inverted pendulum using labview rt," in *Conference Record of the 2000 IEEE Industry Applications Conference. Thirty-Fifth IAS Annual Meeting and World Conference on Industrial Applications of Electrical Energy (Cat. No.00CH37129)*, vol. 2, 2000, pp. 1291–1298 vol.2.

[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[6] Webbylab. (2023) Arduino vs raspberry pi: Key differences comparison table. [Online]. Available: https://webbylab.com/blog/arduino-vs-raspberry-pi-key-differences-comparison-table/#:~:text=Arduino%20is%20better%20suited%20for,cheaper%20than%20Raspberry%20Pi%20devices.

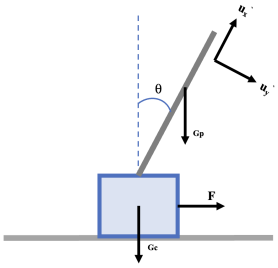
[7] M. T. Adam Paszke. (2023) Reinforcement learning (dqn) tutorial. [Online]. Available: https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html

[8] M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, S. Gelly, and O. Bachem, "What matters in on-policy reinforcement learning? a large-scale empirical study," 2020.

[9] R. V. Florian, "Correct equations for the dynamics of the cart-pole system," 2005. [Online]. Available: https://api.semanticscholar.org/CorpusID:13144387

5 Appendix

5.1 Cart pole Framework



!

Figure 9: Simplified force diagram of cart pole problem, illustrating the mechanics of the various factors playing into the movement of the pendulum.

With these parameters and force diagram, the following equations of motions can be determined:

$$\ddot{\theta} = \frac{g \sin\theta + \cos\theta \left(\frac{-F - m_p l \dot{\theta}^2 \sin\theta}{m_c + m_p} \right)}{l \left(\frac{4}{3} - \frac{m_p \cos^2\theta}{m_c + m_p} \right)} \tag{1}$$

$$\ddot{x} = \frac{F + m_p l (\dot{\theta}^2 \sin\theta - \ddot{\theta} \cos\theta)}{m_c + m_p} \tag{2}$$

The full derivation can be found in the system breakdown and mathematical formulation by Florian [9]. These equations of motion are used to determine the physics of the simulator environment referenced above, built by OpenAI, as well as provide context for comparison between the hardware and the simulation.

Parameter List		
Name	Unit	Description
θ	radians	Angle of inverted pole from center axis
$\dot{\theta}$	radians/sec	Angular velocity of inverted pole from center axis
x	m	Distance of cart from center
$\dot{x}$	m/sec	Linear velocity of cart
F	N	Horizontal force applied onto the cart to move forwards or backwards
m_p	kg	Mass of the pole
m_c	kg	Mass of the cart
l	m	Length of the pole
g	m/s^2	Gravitational acceleration

Table 1: Table of important parameters for the cart pole problem.

5.2 Cart pole Hardware Photos

!

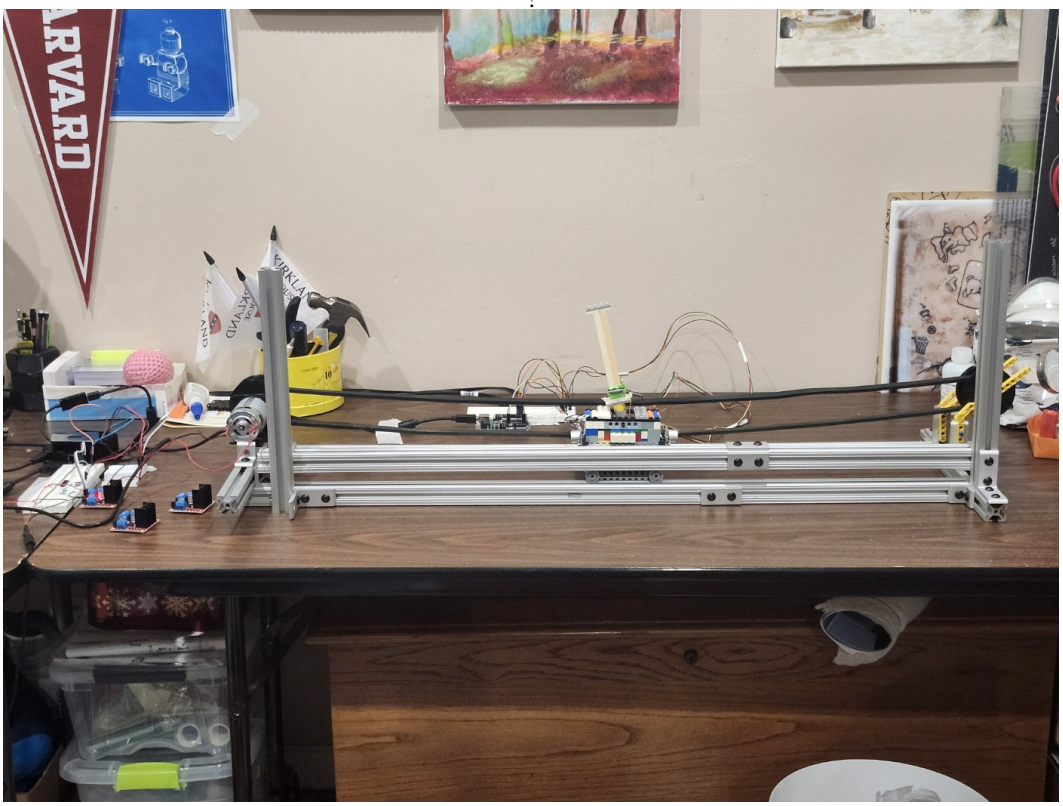


Figure 10: Full hardware setup, with pendulum in the center moving along the rails of the cart pole platform.

5.3 Figures for Hardware Tests

!

Figure 11: Full hardware setup, with pendulum in the center moving along the rails of the cart pole platform, different view. Can see more clearly the raspberry pi, motor, and arduino setups.

!

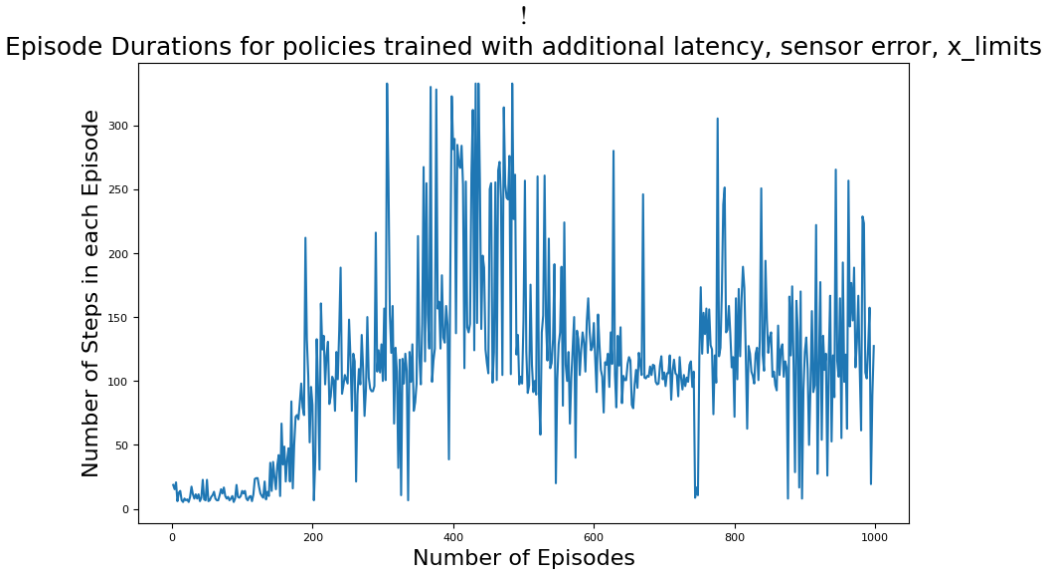


Figure 12: Plot of number of steps the pole was able to balance for each episode run, testing effects of shortening track, adding latency, and randomized sensor errors for final hardware testing