Reinforcement Learning Wheel Intermediate Report

Exploring Simulation To Real Transfer For Robotics

Sponsor: Engineering Physics Project Lab

Authors:

Kevin Barr	95689006
Ashli Forbes	55440929
Sean Ghaeli	78590676
Mackenzie Mar	87610697

1 Executive Summary

The Reality Gap describes the discrepancy between a robot's simulation and real-world performance, and is a significant challenge in machine learning research. The Engineering Physics Project Lab, our sponsor, aims to create experimental platforms for investigating the reality gap, specifically in the area of reinforcement learning. The Reinforcement Learning Wheel has been designed to fulfill these objectives by enabling reinforcement learning research. This system employs reinforcement learning controllers to rotate wheels of different geometries to achieve various control objectives, while also featuring self-resetting capabilities. For example, one application is to balance a ball on top of a circular wheel.

The Version 1 hardware successfully demonstrates the controllability of a ball-on-wheel system using a classical controller, achieving the goal of balancing the ball for 10 seconds. However, testing the V1 prototype revealed the need for design changes to improve performance, such as replacing worn set screws. Moving forward, the aim is to implement a reinforcement learning controller that can generalize to various wheel geometries. This will involve training reinforcement learning models in an existing simulated environment.

Contents

List of Figures

List of Tables

2 Introduction

The sponsor of the Reinforcement Learning wheel is the Engineering Physics Project Lab. The lab is a place of innovation for students of the program. Its independent funding and expert directors allow students to explore complex and difficult design problems. The lap is proposing the Reinforcement Learning Wheel to further investigate the performance gap between Machine Learning or Reinforcement Learning models trained in simulations and their performance on physical systems. This discrepancy is known as the Reality Gap. The goal of the sponsor is to gain expertise in simulation to real systems and investigate the Reality Gap. Specifically, their interest is in creating experimental platforms that enable reinforcement learning research.

2.1 Background and Significance

Machine Learning and AI have been a hot topic in media and scientific literature. With the advancements made in the sector, especially by companies like OpenAI, it begs the question: what is Machine Learning at a basic level? Machine Learning or ML is a method of teaching computers to learn and improve from experience without being explicitly programmed.

Reinforcement Learning (RL) is a subset of machine learning, again where a model learns from trial and error. In specific with Reinforcement Learning, the learning comes through interactions with a set environment. Like a child learning to play a game or learning how to grasp different objects, it takes failures and tries different actions to become better at the task. The basic structure is that of rewards and penalties for the action or result of actions taken, then adjusting behaviour. Models learn and adjust their behaviour over many trials. Developing models by training on physical systems could take years to achieve what computers can achieve in a matter of hours. Thus, the use of simulations is necessary to train ML models efficiently.

When it is difficult to model a system with classical physics or apply classical control algorithms, there is motivation to use RL. However, models developed purely in simulation often do not perform as well in real life. The building, training and testing of the model is done in a controlled and perfect environment. Yet, in the real world, the model cannot keep operating in an unpredictable and complex one. Signals have lag, motor actuators are not perfect and the model is not robust. This means the model does not generalize well and is not scalable.

Reducing this gap is a significant challenge in robotics. Currently, many researchers are looking at ways to achieve this. Leading theories include but are not limited to domain randomization, varying parameters within the system, feeding back physical results and hardware loop testing. All these techniques have resulted in major strides in the closing of the gap [2]. The benefits of increasing simulation fidelity include cost reduction and support for rapid prototyping. It will result in more general solutions and allow for the application of true Reinforcement or Machine Learning to physical problems whereas they are currently constrained to cloud computations. The motivation for this Capstone is to investigate this issue further.

2.2 Problem Statement

The capstone project is a reinforcement learning wheel. The goal for this year is to design a control system that balances a ball on a rotating wheel. This system enables a high-level investigation of the Reality Gap. This system enables the study of different approaches to minimize the impact of the Reality Gap.

There are two main components to the system: the physical system and the virtual simulation/RL pipeline built in parallel. The physical system consists of a wheel driven by a motor which is controlled, and a ball that moves freely along the wheel. Depending on the geometry of the wheel, the goal of the control system may vary.

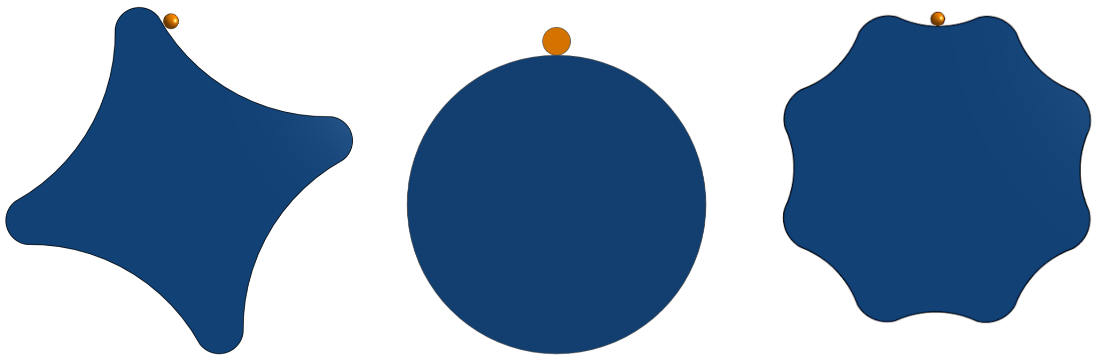


Figure 1: Wheel Geometries

In the case of a circular wheel, the goal is to keep the ball balanced on top of the wheel. This control problem can be approached with classical control such as PID [5]. This problem closely resembles the Inverted Pendulum problem, where a model must learn to balance an object which moves left or right. When adding points of equilibrium, where the ball comes to rest, the dynamics change. Instead of keeping the ball balanced on top, which would happen naturally at points of equilibrium, the goal is to rotate the wheel without letting the ball fall off. At the end of this θ-degree rotation, the ball would be resting a different point of equilibrium. This problem is less trivial and difficult to solve with classical control. This is where Machine Learning comes in.

This mechanical system has four state variables, the positions and velocities of the ball and wheel. Training a machine learning model requires input data. One option is to use computer vision to get values for these four state variables, as discussed in Section 3.2. Another option is to minimize image processing and use the image as direct input to the ML pipeline. These options will be explored when implementing machine learning.

2.3 Scope and Limitations

The scope of this project includes researching the Reality Gap and how it affects the performance of RL models in the context of the system. Ideally, the RL model would generalize well to wheels of different geometries, meaning the control system can balance a ball while rotating with differently shaped wheels. The RL model would also ideally perform the same on different physical and virtually simulated systems. However, in practice, this is often not the case and will be looking at ways to improve the model's ability to generalize while maintaining performance. The timeline for this research project is two years. The objective for year one was to design the mechanical and electrical aspects of the system and implement classical control to balance the ball on the circular wheel in order to validate the electromechanical system and test the responsiveness of the control system. The second year will focus on developing the simulated environment and the reinforcement learning pipeline.

This intermediate report will cover the progress made in the first year. This includes the electromechanical design of the system, image processing, implementation of classical control (PID) to balance the ball on a circular wheel, performance results of the classical control, and recommendations to improve the current iteration.

3 Discussion

This section will review design challenges and the approach to them. After this review, there is a discussion of the validation of the system and improvements made to future iterations. As part of the design process, a list of design challenges the system would need to address:

- Ball detection
- Control the wheel's position (and ball position)
- Reset the system
- Collect data for control system
- Develop a Machine Learning pipeline

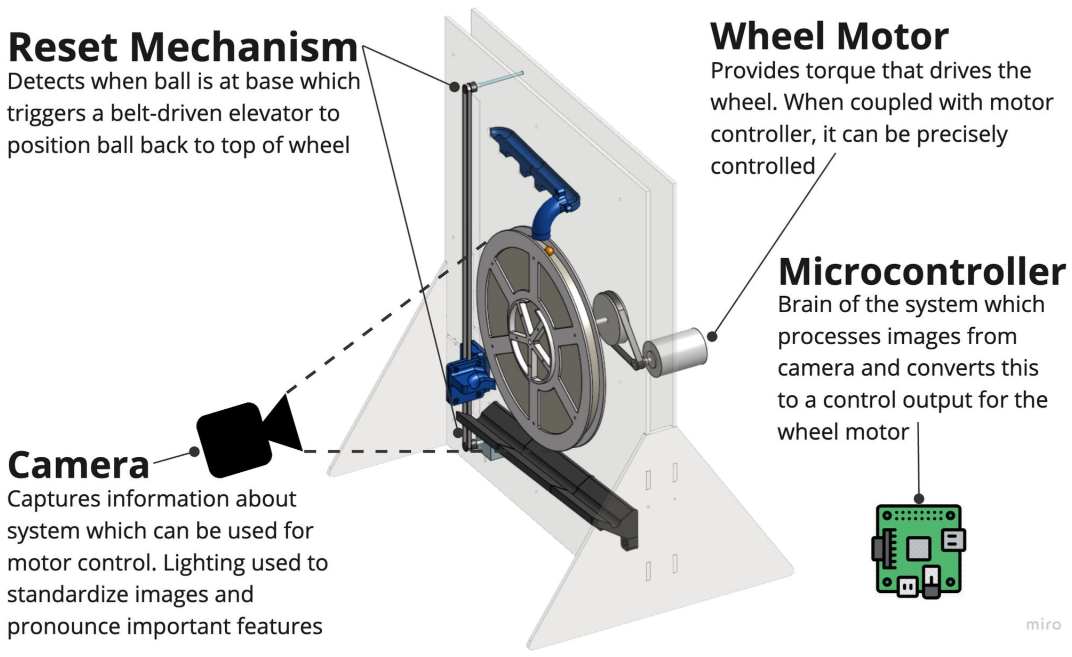

Reset Mechanism
Detects when ball is at base which triggers a belt-driven elevator to position ball back to top of wheel
Wheel Motor
Provides torque that drives the wheel. When coupled with motor controller, it can be precisely controlled
Microcontroller
Brain of the system which processes images from camera and converts this to a control output for the wheel motor
Camera
Captures information about system which can be used for motor control. Lighting used to standardize images and pronounce important features
miro

Figure 2: System Diagram

3.1 Resetting The System

A Reinforcement Learning (RL) model learns from its successes and failures. Outcomes link to rewards and penalties. The objective of an RL model is to determine the actions that maximize rewards. In the context of the RL problem, give a reward for maintaining the ball on the wheel, and penalties for letting the ball fall off the wheel. While actions refer to torque applied to the wheel in either direction. The model learns from its actions and outcomes over thousands of trials, resetting to an initial state each time. When training virtually, resetting a system is easy. However, resetting a physical is a design challenge of its own. In the context of the RL problem, the model fails when the ball is no longer on the wheel. The implementation of a reset mechanism to self-reset the system remedied this.

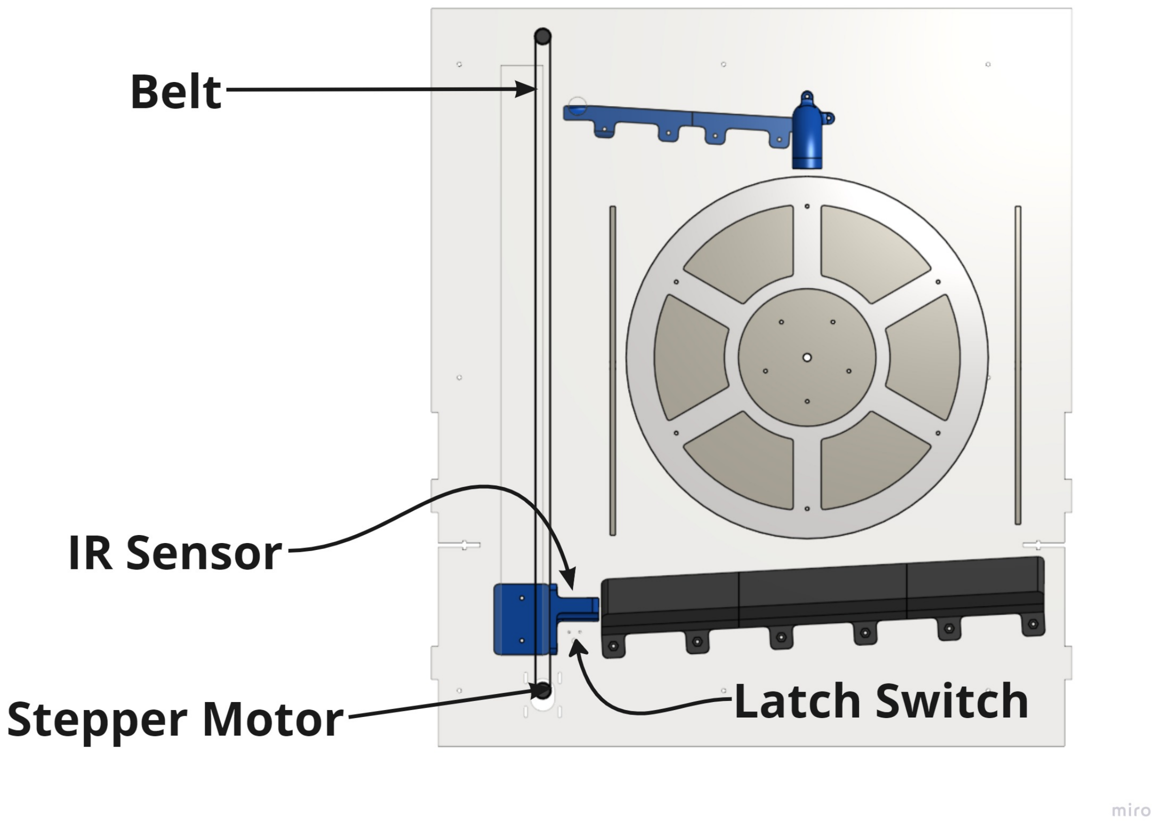


Figure 3: Reset Mechanism

The Reset Mechanism consists of ramps to catch/deposit the ball and a belt pulley system to raise the ball. Barriers constrain the ball to the same plane as the wheel. The motor that drives the belt is a stepper motor. This allows for fixed steps and the ability to know where it is when bringing the ball up and down. When the ball falls off the wheel, an IR sensor detects it, which detects objects 1 to 2 cm in front of it. Using a black ball activates the sensor as it absorbs light, whereas the blue elevator reflects light. The ball senses light is no longer reflected, initiating the reset routine. The reset routine involves driving the stepper motor counter-clockwise, depositing the ball on the ramp, and then back down to catch the ball again.

3.2 Image Processing

For any control problem, the sensors measure the state of the system and then the controller operates. While there are other sensor options, using a camera sensor is a natural choice to make observations about arbitrary ball positions and track geometries.

Regarding camera specifications, the resolution and frames per second (FPS) requirements are not aggressive. Specifying the optical parameters of the image sensor such as focal length is not needed since there is the freedom to place the camera at any distance away from the wheel. A camera designed for real-time video streaming is also optimal. Given the choice of microcontroller (Section 3.3), the Raspberry Pi Camera is an astute choice. The camera configuration is as follows: distance of 40cm away from the front of the wheel, an imaging resolution of 340×240 pixels, and 30fps. One asset of the Raspberry Pi Camera is its versatility, which leaves the option open to tune these parameters should the need arise.

RL has demonstrated some capabilities in learning to map images to control outputs [3]. However, the high-dimensional nature of image data proves to be challenging and often leads to unreliable controller performance. This motivates the need to explicitly program software that discerns the position and speeds of both the wheel and ball to feed these into the RL controller. There is also motivation to use ambient lighting so that the images captured by the camera have less variability to external lighting influences. Furthermore, the lighting also helps reduce glare on the clear plastic guard rails which helps with determining the ball position. A tripod mounted the lighting from below, which is the optimal place for avoiding the casting of additional shadows.

The usage of the OpenCV library enabled image processing implementation B. The controlled design of the imaging environment licenses the ability to make strong assumptions about the input images: constant background and ball colours and constant areas of interest. The first assumption allows the use colour thresholding to filter all pixels that have a colour that does not match the ball, which leaves us with an image that only contains the ball itself with noise. From here, contouring is used to find the circle in the image and use the centroid of the detected circle is used as the position of the ball. This routine proves to be satisfactory both in reliability and execution time (Table 1).

3.3 Controlling The Wheel

The main challenge considered in controlling the wheel was the responsiveness and stability of the control system. Forcing the consideration of the mechanical, electrical and software systems of the design.

Minimizing the wobble from the wheel's rotation was crucial so the camera would have a stable image of the system. Furthermore, the housing for the mechanical system is quite a bit larger than the wheel itself, so the aspect ratio of the base is larger, giving more stability. To leverage greater torque from the motor, utilization of the following formula is helpful:

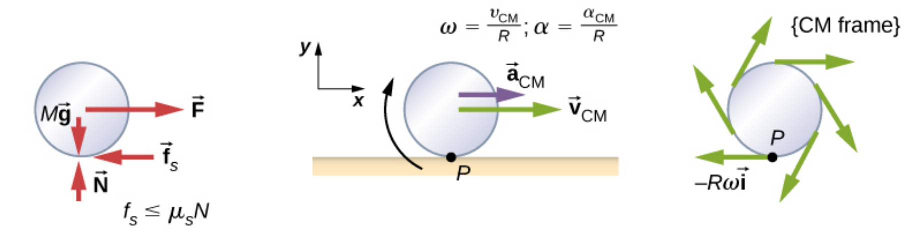
$$\frac{d_{out}}{d_{in}}\tau_{in} = \tau_{out}$$

The pulley attached to the wheel shaft has a larger diameter than the pulley attached to the motor, yielding greater torque. Small rotations of the motor lead to larger movements of the wheel, contributing to the responsiveness of the control system.

Closed-loop control time is a key metric to quantify the performance of the system. The closed-loop control time is the time required to measure the state of the system and convert it into a Pulse Width Modulation (PWM) output for the motor. This metric is also known as the time delay of the control system. Classical control theory indicates that increasing time delay results in reduced controllability of the system. Therefore, the goal was to reduce the time delay as much as possible. Although there was no strict requirement for this metric, having it below 33ms would allow the camera to operate at the maximum FPS allowable for the chosen resolution (see Section 3.2).

Initially, the time delay was upwards of 700ms, indicating an underlying problem. The measurement of code execution time, determined to be 150ms, demonstrated that the rest of the delay could only be from one of two sources. Either the camera was operating at a lower FPS than expected or the operating system of the microcontroller processed incoming frames in a manner that introduced too much delay. Analyzing the FPS of the video stream determined that the latter was the issue.

The implementation of code that shortened the camera buffer queue and reduced the resolution of the camera resolved this problem. These changes brought the time delay under 33ms, as needed. Seeing as the ball position remained easily discernible both by software and the human eye, the reduction in resolution is a negligible threat to controller performance



(a) Forces on the wheel

(b) Wheel rolls without slipping

(c) Point *P* has velocity vector in the negative direction with respect to the center of mass of the wheel

Figure 4: Rolling without Slipping Mechanics, Image from [1]

Minimum (ms)	Average (ms)	Maximum (ms)
15	30	80

Table 1: Closed-loop control time metrics

The whole goal of rotating the wheel is to counteract the ball's inertia and use the force of friction to move it back closer to the center. The goal of this system is to exploit rolling without slipping. As seen in Figure 4, the 'restoring' force that would bring the ball back to the center relates to frictional force, which is reliant on the normal force which is reliant on the force of weight and angle of contact with the wheel. Another way to increase this force is to increase the angular speed of the wheel. This was the method which Section B further details. Due to the light weight of the ball, past an angle of contact of around $\theta = 30°$ to $45°$, regardless of the speed of the wheel, the ball would fall. This motivates investigating different ball sizes, weights, and colours next year, not only to make the RL model robust but to yield greater controller performance.

The closed-loop control time metrics and Table 1 details it further. While most control loop times are around the average, there is variability in both directions. This is one of the reasons that the usage of a C++ based microcontroller may be desirable, for more reliable fulfillment of processing times.

Due to the volatility of the system, a motor needed the ability to move back and forth quickly and precisely. For an overestimated need of $3g$ acceleration at the wheel edge, a torque of approximately $\tau = 0.2N \cdot m$ is needed on the wheel axle. This must take place at low speed, so a DC motor is the best choice. Because of many changes in direction, it is also desirable to operate within the continuous range to avoid overheating the motor. Thus, the choice of a brushed DC motor allows for starting and stopping within the continuous operating range.

3.4 Software Control System

In order to control the system and have a source of direction, the system required a microcontroller. The chosen microcontroller is a Raspberry Pi4. With an operating system of Debian and a GPU VideoCore VI at 500MHz, this was suitable for the first iteration of hardware [4] This allows for easy integration of the Raspberry Pi camera as well as the reading the physical state from the GPIO inputs and outputs. The chosen Operating System or OS was the standard Debian, which is compatible with the Pi. The Raspberry Pi 4 runs Python code, which while slower than C/C++, provided great support for OpenCV as mentioned in Section 3.2, allowed for the location of the ball.

There were four functions made, first `init_board()` which initialized and properly allocated all GPIO Pins. Then `homing_sequence()`, which performed the homing routine specified in Section 3.1. Then flowing to a `while(True)` loop that then would either look for input from the sensor or look for the ball. The full `main.py` code looks like:

```
"""Main Script
This runs the main control loop for the wheel.
Calls the homing, elevator, initialize and controll functions/files.
"""
import RPi.GPIO as gpio
from modules.initialize import init_board
from modules.elevator_routine import homing_sequence, elevator_routine

IR = 16
init_board()
homing_sequence()
controller = Controller()
try:
    while(True):
        if (gpio.input(IR) == False):
            elevator_routine()
        controller.control_routine()

except KeyboardInterrupt: # If there is a KeyboardInterrupt (when you press ctrl+c),
    print("Cleaning up!")
    gpio.cleanup()
```

Links to GitHub and the complete code is found in the Appendix in Section B. In addition to this, the electrical pinout with appropriate sensors and drivers can be found in the Appendix in Section A.5 at Figure 8.

The implementation of a proportional-derivative (PD) controller enabled hardware validation and system testing. Despite the simplicity of the controller, the system has demonstrated the ability to do a good job of balancing (A.6).

Data logging and visualization aided with the task of finding optimal PD gains. For example, the use of Figure 5 determined that ball positions more than 10 pixels away from the equilibrium point were in need of additional error amplification. This change improved controller performance.

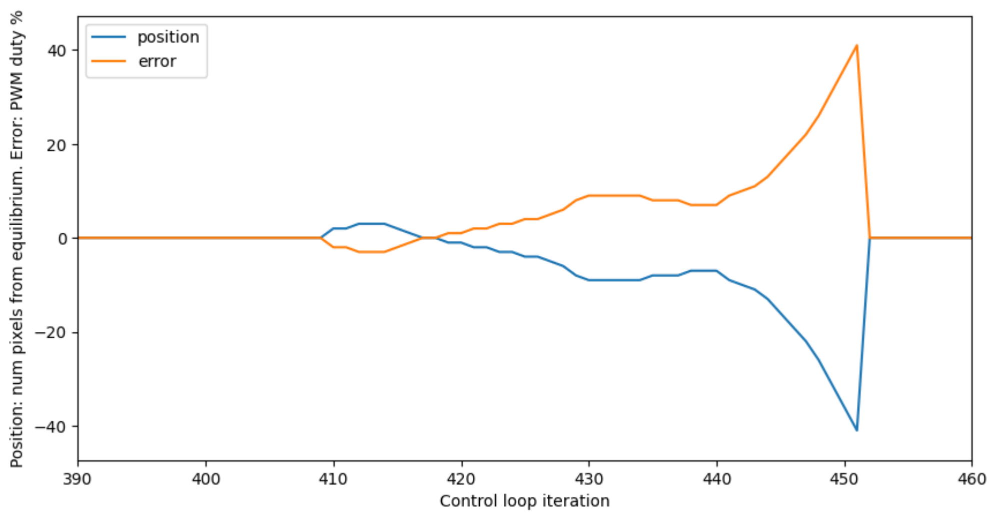


Figure 5: Controller Telemetry

3.5 Simulation

Although the advancement of the simulation was not far enough to train controllers and integrate them into the real system, there is still important progress on this front. Investigations began with using ROS as a robot controller and Gazebo as a simulator. While other options were and still are available, the team's personal experience with these platforms makes them a good starting point. Furthermore, following advice from mentors, making the switch to different platforms is a relatively simple task, and is thus associated with little risk.

Figure 6 demonstrates the successful integration of a simplified CAD model of the system with a single joint into Gazebo.

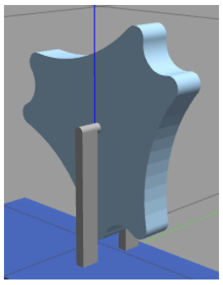


Figure 6: Wheel in Gazebo simulated environment

Appendix C demonstrates the implementation of a ROS package that can launch an instance of the simulation environment in Gazebo. The next step on this front is to use ROS to control the wheel, and this will conclude the bulk of the work in setting up the simulation environment to enable reinforcement learning to take place. From here, iteration of software tools will be possible.

4 Conclusion

The first prototype of the system setup meets the objective for the first year. It is able to balance the wheel for about 10 seconds semi-consistently. The reset mechanism is functional, and the feasibility to prototype and expand this system has been established. The physical system is displayed in Figure 7. With some changes and recommendations to better suit the project's high-level objectives and improve upon faults found when prototyping, the next year will be focusing on the investigation of minimizing the Reality Gap.

There were issues with the compile-time and speed of the controller, which is critical as the reaction time of the wheel in order to achieve balance and precise movement. There are also minor changes to be made to the physical design. A crucial one that did result in limited demos during the Project Fair was that the motor's output was not being directly translated to the wheel. This had to do with the wear and tear on the set screws found both on the pulley connected to the motor and the hub connecting the wheel to the shaft. After repeated tightening, the acrylic plastic eventually gave way and no amount of tightening would result in fixing the slipping problem. The set screws were found to be worn down and thus regardless of the controller output, the wheel could not balance the ball. Next year the hub cap will be re-designed to fix these problems. The recommendations for the hardware updates for next year including the set screw issue are as follows:

- experiment with different ball sizes
- re-design of hub-cap that would lead to less set-screw and gripping issues
- re-design of housing, to allow for easier iterations and fixes
- switch out IR Sensor to a different sensor to a possible physical latch sensor to make the reset routine ball colour agnostic
- add an encoder to ensure accurate and wheel-agnostic tracking of wheel position

A general note moving into phase two of the project will be furthering the understanding of the Simulation to Real Pipeline and addressing issues surrounding the Reality Gap. This will allow the project to focus more on the higher level objectives and produce meaningful results. A key hardware change is the choice of the microcontroller. While the Raspberry Pi 4 served well for V1, moving forward, the compile time requires an upgrade, so the exploration of other options remains on the table.

- Switch from ROS and Gazebo to a stronger physics-based simulation environment, possibly PyBullet or MuJoCo

- Switch to a faster and stronger micro-controller, the Raspberry Pi only has a GPU of 500 MHZ. If switched to a possible NVIDIA Jetson which has a 128-core at 921 Mhz or a possible fast C++ based controller
- Implement Reinforcement Learning based controller
- Create different and interchangeable wheels to demonstrate the generality of the solution
- Reach out to relevant experts in the field to begin an iterative process for the simulation to real pipeline

5 Deliverables

1. CAD document of mechanical system
2. Ohysically built system (electromechanical system)
3. URDF files compatible with Gazebo
4. Software (PID Control Code, Image Processing, sensor processing, motor control)

References

[1] Section 11.1: Rolling from University Physics Volume 1.

[2] Konstantinos Bousmalis and Sergey Levine. Closing the Simulation-to-Reality Gap for Deep Robotic Learning. 2023.

[3] Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine. How to Train Your Robot with Deep Reinforcement Learning – Lessons We've Learned. 2020.

[4] Adafruit Industries. Raspberry PI Model B Specifications PDF.

[5] Joshua Vasquez. BALL BALANCING WHEEL PUTS A SPIN ON INVERTED PENDULUMS. 2021.

Appendices

A Mechanical Design

A.1 Parts List

Link to Parts List

A.2 CAD

CAD File

A.3 Hardware

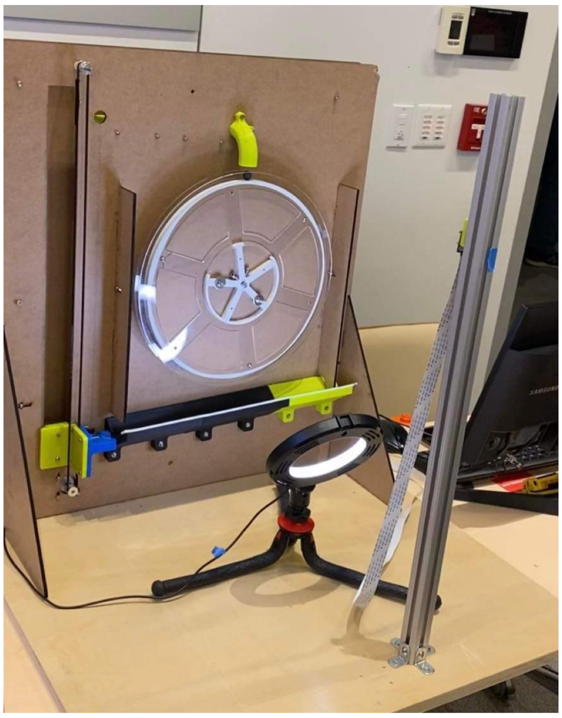


Figure 7: Wheel hardware

A.4 Modifications

List of Modifications

A.5 Electrical Diagram

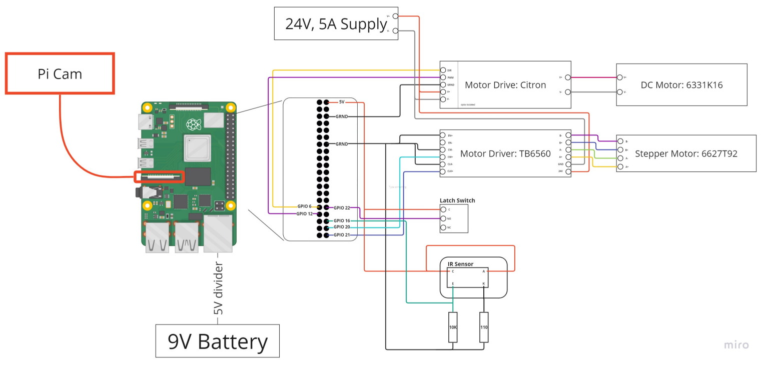


Figure 8: Electrical Pin-Out

A.6 Website Page

Link to project page

B Control Software

GitHub Repository

C Simulation Software

GitHub Repository