Reinforcement-Learning Wheel Final 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

Executive Summary

The sim-to-real gap describes the discrepancy between a robot's simulation and real-world performance, and is a significant challenge in machine-learning research. Sponsored by the UBC Engineering Physics Project Lab, we designed and constructed a reduced-complexity experimental platform for investigating the sim-to-real gap. The reinforcement-learning wheel is a robotic system for balancing a ball on top of a wheel (Figure 1), constraining the control to a single degree of freedom. This reduced complexity allows for the performance comparison of varying control models in a single task, across both simulation and reality.

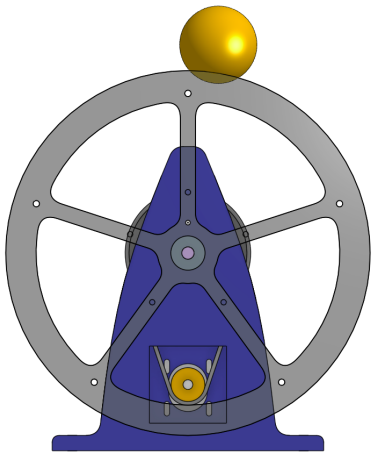


Figure 1: Ball-on-wheel robotic system.

Our system successfully balanced a ball using a classical controller in both simulation and the real environment (video in Appendix .2). However, the parameters varied greatly, with the real system requiring correction for the inertia of the wheel. This indicates a large gap between how the ball and wheel interact in simulation versus in reality. Thus, we recommend increasing the quality of the simulation to model measured values of the real system.

In the simulation, an imitation-learning model (machine learning by mimicking classical control) perpetually balanced the ball. The reinforcement-learning model (machine learning by random trials) can balance the ball, but often drops it. Moving to the real system, neither of these models successfully balanced the ball. We recommend continuing this project, focusing on implementing machine-learning control models into the real system, especially for use of non-circular track designs.

Contents

1 Introduction

The sim-to-real gap refers to the differences between a real system and its simulation. The purpose of this project is to explore the sim-to-real gap by defining a system which can be built and used to train machine learning (ML) models both in simulation and reality. Overcoming the sim-to-real gap implies training a model in simulation that is able to be deployed on the system in reality. While one could train a model on a real system, this has a greater cost associated. Therefore, overcoming the sim-to-real gap means increasing the feasibility of using ML models to control physical systems.

To explore the sim-to-real gap we have constrained ourselves to a ball-on-wheel problem, where the goal is to keep an unconstrained ball on top of a rotating wheel track. In this project, we have mostly experimented with a circular wheel geometry, but have designed the system such that other wheel geometries can be easily tested.

In both simulation and reality, we demonstrated the controllability of the circular ball-wheel system using classical control. In simulation, we demonstrated that a reinforcement learning (RL) model can be trained to balance the ball on the circular track. Initial training with the real system implies the same can be done in real. The design and initial construction for a reset mechanism has been completed to facilitate the training of models on the real system. We also achieved initial classical control development with a peanut-shaped track. The peanut and circle shapes were chosen based on controllability. The proof of concept controls were provided by Surov [2] and Vasquez [3]. The cited projects gave us an approximate performance goal for our physical system.

An extended ambition for the project is to generalize our model training to control more complex shapes that would be too difficult to characterize with Lagrangian mechanics. We hope to explore this system further, and ultimately train a ML controller in simulation that works on the real system.

This project was completed as a senior capstone project in engineering physics at the University of British Columbia (UBC). The sponsor of this project is the UBC Engineering Physics Project Lab, located in Vancouver, BC.

2 Discussion

2.1 Theory

2.1.1 Classical Control

To characterize the ball-on-wheel balance system, we derived a transfer function from θ_w to θ_b (see Fig. 2, derivation in Appendix .1). We assume rolling without slipping and linearize about $\theta_w = 0$. Control of this system was achieved in real by Vasquez [3]. Therefore, our motivation for this analysis was not to prove controllability but to determine how the mass and radii of the

ball and wheel affect controllability. We used MATLAB to plot the transfer function's response to an instantaneous perturbation of the ball for possible combinations of physical parameters, with an example shown in Figure 3.

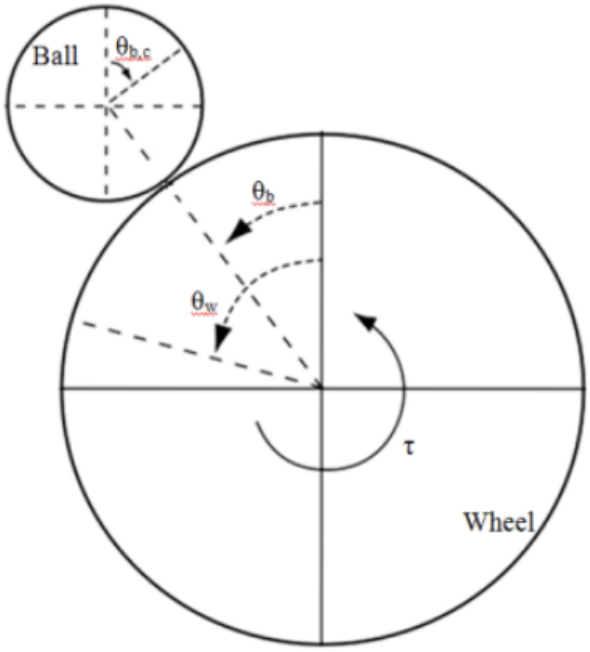


Figure 2: Diagram for Ball on Wheel setup. Source [1]

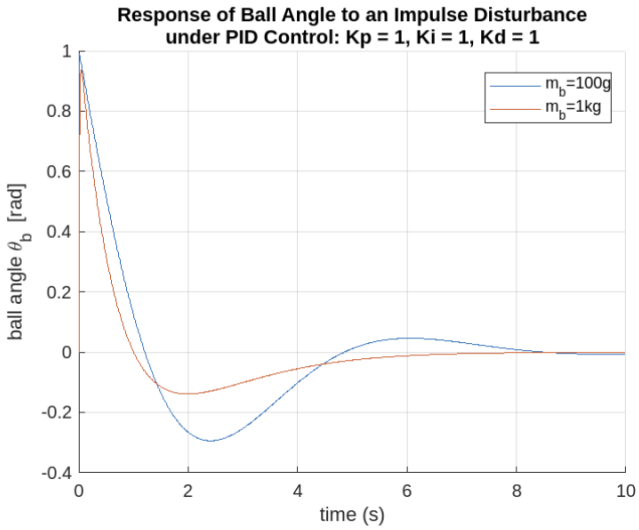


Figure 3: Impulse response of ball on wheel transfer function for two different ball masses, indicating that a heavier ball reduces controllability. This follows physical intuition, as a heavier ball will require more torque from the wheel to accelerate.

Through this analysis, we predicted a 12 cm wheel radius and a ball of mass between 100 g and

1 kg would increase controllability with sufficient inertia to reduce bouncing.

A more complex geometry could contain multiple equilibrium points. An example is shown in Figure 4, a shape we call the peanut track. Characterizing and controlling this system has been done by Surov [2]. Knowing the system is controllable while more complex, we aimed to apply learning algorithms to control this system.

Figure 4: Peanut-shape wheel geometry with two possible points of stable equilibrium. The task is to flip the wheel while transferring the ball from one stable equilibrium to the other.

2.1.2 Reinforcement Learning and Imitation Learning

Reinforcement learning (RL) is a branch of machine learning that focuses on training agents to make sequential decisions in environments to maximize cumulative rewards. At its core, RL involves four key components: agent, observation, reward, and loss.

Agent

The agent is the learner or decision-maker in the RL framework. It interacts with an environment by taking actions based on observations, and receives feedback in the form of rewards. The agent's goal is to learn a policy, a mapping from states to actions, that maximizes the cumulative reward over time.

Observation

Observations, also known as states, represent the current configuration or situation of the environment perceived by the agent. These observations can be raw sensory data, such as images or sensor readings, or abstract representations derived from the environment. The agent uses observations to make decisions about which actions to take.

Reward

The reward signal is a scalar feedback signal provided by the environment to the agent after each action is taken. It indicates the immediate desirability of the agent's action. The objective of the agent is to maximize the cumulative reward over time. Rewards can be positive, negative, or

zero, depending on the desirability of the outcome of an action.

Loss

Loss, also known as the objective function or cost function, is a measure of how well the agent is performing its task. It quantifies the discrepancy between the predicted outcomes (actions) and the desired outcomes (optimal actions). The agent's goal is to minimize the loss function over the course of learning. Common loss functions used in RL include mean squared error (MSE) for value estimation and policy gradient loss for policy optimization.

In summary, reinforcement learning involves an agent that interacts with an environment by making sequential decisions based on observations, receiving feedback in the form of rewards, and optimizing its behavior to maximize cumulative rewards while minimizing loss.

Imitation Learning

Imitation Learning (IL) involves the agent learning by observing and imitating a supervisor's behavior, such as a human or classical controller, accelerating learning through expert guidance and reducing the need for exploration. This is in contrast to RL, which is unsupervised.

2.2 Design

2.2.1 Simulation

For to simulate the ball and wheel system, we chose Gazebo. While there are higher fidelity simulators, we thought that our system was simple enough that Gazebo was sufficient. For the control framework, we used the commonly-paired ROS. The first step in developing our simulation was making a basic CAD model of our system. The Gazebo simulation requires the robot model to be defined by a URDF file. We generated the initial URDF using onshape-to-robot.

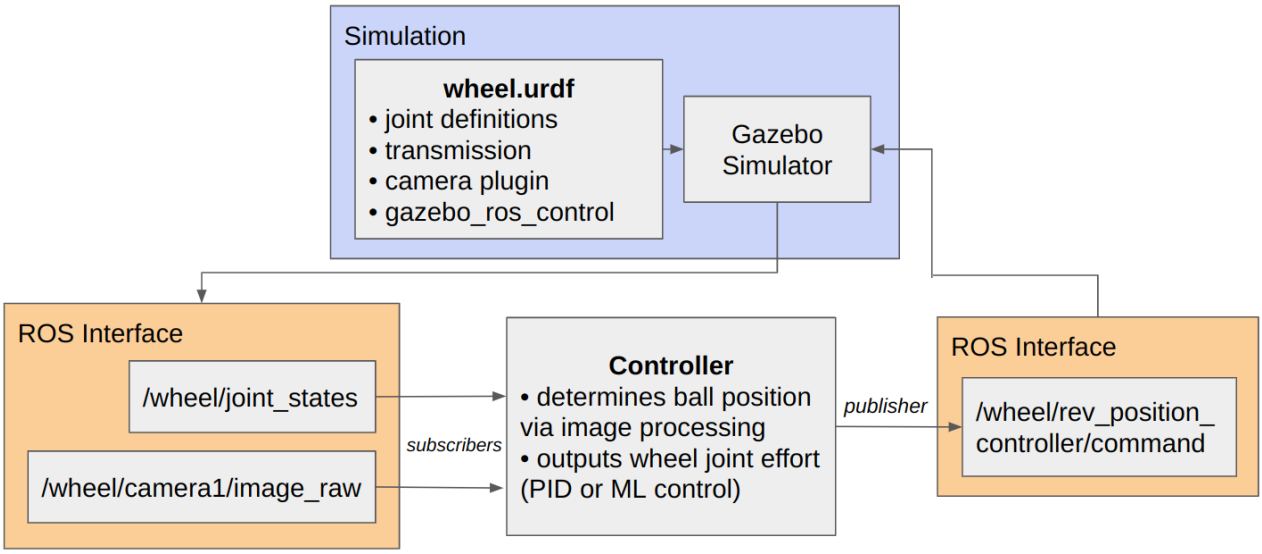


Figure 5: Gazebo-ROS software diagram for simulation

Our system has one continuous joint, allowing the wheel to rotate about its axis freely. Using ROS, we are able to get data from the simulation on the angular position and velocity of the wheel. While there was the option to get the ball position directly from the ROS, we used a Gazebo camera plugin since this is how we found the ball position in the real system. We also implemented a modifiable artificial delay in simulation to match the delay in real.

The main design challenge of the simulation was characterizing the motor accurately since the motor was not modelled directly. In place of a motor, we have the ROS joint controllers in simulation. The effort controller was chosen over the position or velocity controllers based on its PID performance and analogue to pulse width modulation (PWM) motor control. Furthermore, the ROS controller framework provides the option of PID parameter tuning, which could be used to characterize the motor to increase simulation fidelity.

2.2.2 Real System

Closing the sim-to-real gap requires aligning the real system with the simulation and ensuring compatibility with its outputs. This entails replicating the inputs and outputs of the simulation, such as ball detection, wheel movement, and reporting on wheel position or velocity. Achieving this involves critical software, hardware, and mechanical decisions, as illustrated in the system diagram shown in Figure 6.

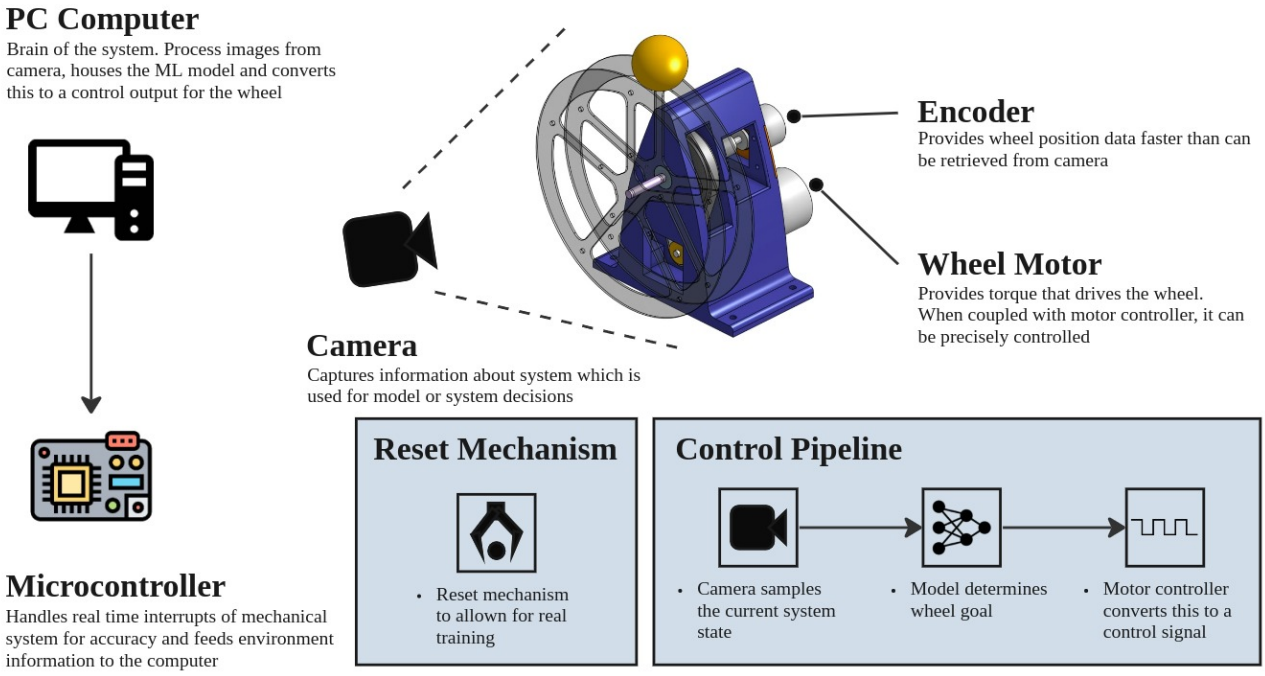


Figure 6: System Diagram with Brief descriptions of hardware components.

Computer and Software

The system depicted in Figure 6 aims to keep the ball balanced, utilizing specific components and the following workflow. Initially, the USB camera captures an image of the system. The image is transmitted via USB A to the PC computer, running on Xubuntu 20.04 and Python 3.10 for compatibility with ROS. As depicted in Figure 7, the PC control loop employs OpenCV to process the image, identifying the ball's location by analyzing contours and moment weights. The classical control or learned model utilizes the ball position and wheel state information to determine the desired action, which can include achieving a specific wheel position, wheel velocity, or just balancing the ball. The tuned controller or ML model then decides on a PWM request scaled by 100 for user readability.

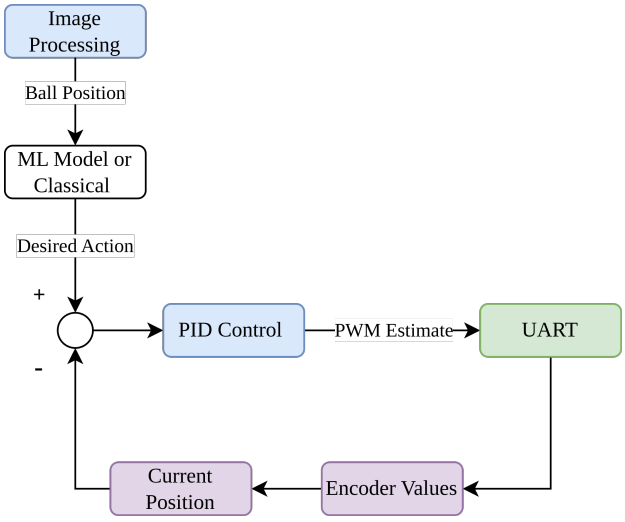


Figure 7: Computer Information Flow

Firmware and Controlling Hardware

The PWM request from the PC to the microcontroller is transmitted via UART (universal asynchronous receiver/transmitter) protocol, chosen for its speed, full duplex capability, and reliability over long wired connections, as opposed to SPI or I2C. The PWM value is serialized within the package and transmitted through the PC's USB A comm port, then passing through the USB to serial converter, ultimately reaching the STM32 F702RB Nucleo Board. This microcontroller controls the motor and reads values from the encoder to assist the PC in decision-making. It operates in C to utilize real-time interrupts for encoder reading. Additionally, the system leverages the board's DMA to manage UART receiving and store queues to prevent message loss. The Nucleo board processes the PWM request, sending it along with the motor direction to the motor driver rated for 30 Amps. This external hardware receives 24 volts of power and connects to the A and B ports of the DC brushed motor. This moves the wheel and thus the ball as intended. Refer to the information flow diagram in Figure 8.

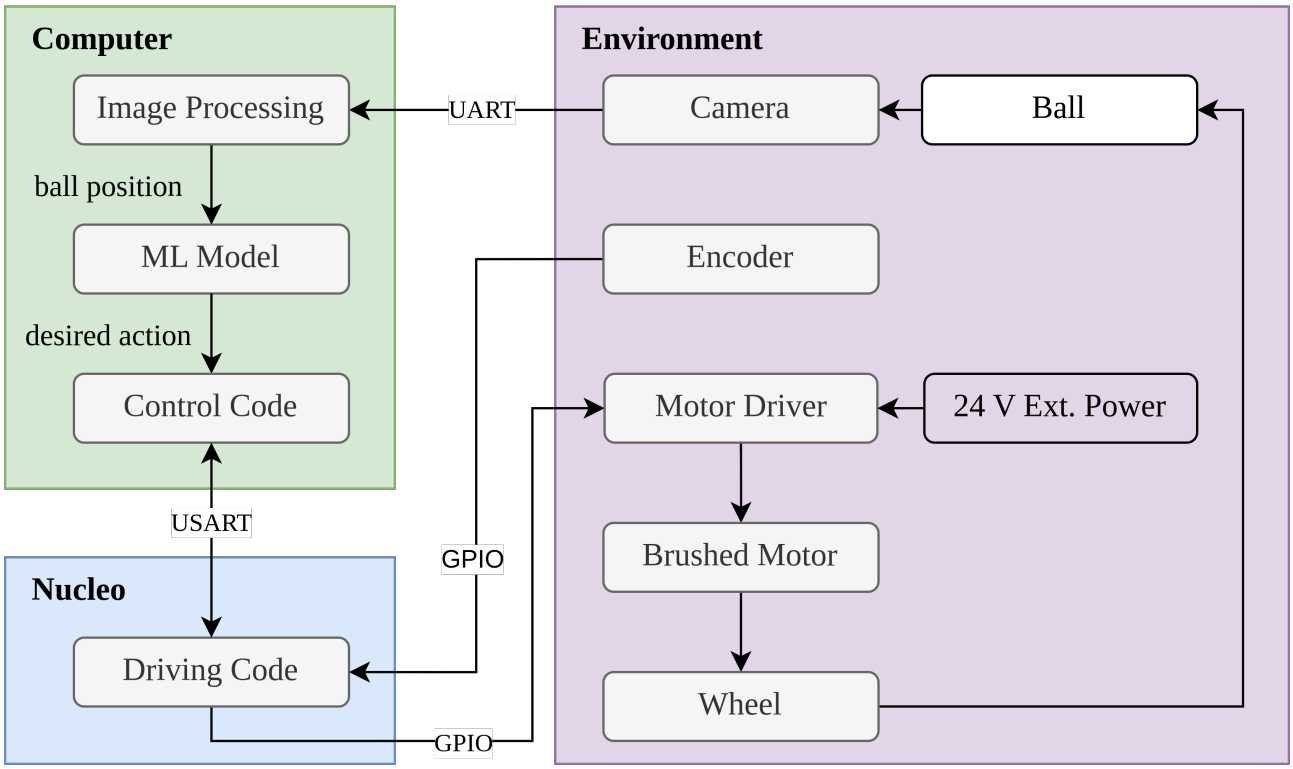


Figure 8: Information flow of the whole system

Mechanical Description/Design

The main mount for the wheel is depicted in the system diagram of Figure 6, and has been designed to be sturdy so as to minimize vibrations being carried through the system. In addition, all parts are readily accessed to be mounted or dismounted. The motor torque is geared down through a belt threefold to increase the torque at the shaft connected to the wheel. The holes that mount the motor itself are slotted so that the belt can be manually tightened. The wheel shaft is also connected by a shaft coupler to the encoder. The whole mount is then bolted down to an optical breadboard, and placed in front of the camera also mounted onto the breadboard.

The wheel itself is connected to the shaft via a water jet aluminum hub connected by a set screw. Using a weaker material such as plastic for the hub cannot handle the torque transfer and cracks. The wheel is manufactured by laser cutting two sheets of material and separated by standoffs.

The rotary encoder attached at the shaft's end on the back side detects changes in angular position. With a resolution of 2400 ticks per revolution, it accurately relays system information to the PC. The phase is carried back to the Nucleo board, using pull-up configurations and an encoder handling data structure in order to handle the absoluteness of the device, the change per request is sent back over UART to the PC.

Telemetry and Logging

Upon receiving the changed ticks via UART, the PC utilizes this information to assist in position

and velocity control decisions, logging it using a background class. While certain logic components could reside on a C-based microcontroller in other systems, it all resides on the PC in this setup, granting future models access to all information. The logging class provides telemetry and insights for the entire system.

Once the ball has been moved according to the desired action, the camera captures a second image, and the cycle repeats. In the event the ball falls off, the reset mechanism places it back on top.

Reset Mechanism

The reset mechanism is a system to collect and replace the ball back to top of the wheel when it falls off. This system is necessary to allow for automated training of machine-learning models in reality (i.e. leaving the wheel overnight to train). This design should drop the ball with minimal initial kinetic energy, which would influence the success of the ball balancing.

The initial plan for the reset mechanism was to simply drop the ball out of a spout onto the track. However, this design was pivoted from because of changes in the ball size and track design. Using a larger ball meant that the ball would have to fall from a greater distance, increasing its starting energy. Additionally, the track design changed from the ball rolling in a channel to the ball rolling on two rails. This also prevented a dropping-type design because the ball was likely to hit one rail before the other, bouncing away from the wheel.

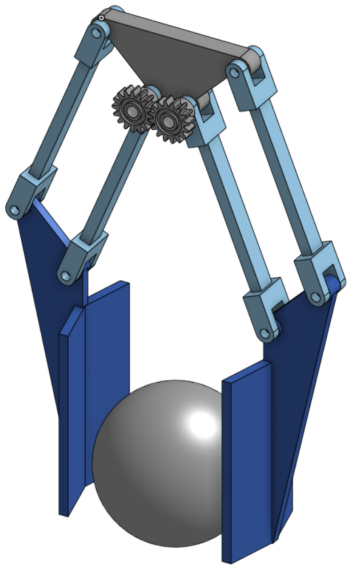
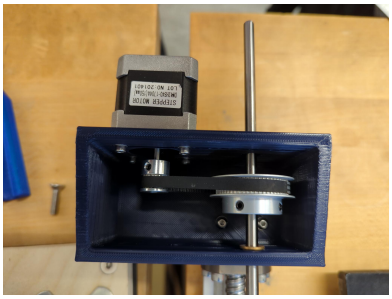


Figure 9: Claw design capable of grabbing variable ball sizes.

In January, we chose to develop a claw to place the ball on top of the wheel, avoiding a dropping motion. This allowed for sufficient reduction in ball starting energy, without the added complexity of needing compressed air for a vacuum cup. The claw is mounted on an arm which raises/lowers on a lead screw, then rotates into place overtop the wheel with a stepper motor.

(a)
Reset mechanism walls.



(b) Mounted arm pivot assembly.

(c) Gears and servomotor for the claw.

Figure 10: Unfinished reset-mechanism parts.

Late into the project, assembly of the reset mechanism was delayed 3 weeks because of repeated missing packages with McMaster-Carr parts. While available sections of the reset mechanism were completed, final 3D printing and assembly slipped just past the end date of the project.

2.3 Tests and Results

2.3.1 Simulation

An **episode** describes a training run started in some initial position (i.e. ball on top of the wheel) and ending once some condition is reached (i.e. ball off the wheel). **Batches** are made of several episodes. **Loss** represents the difference between the optimal model and the current model. After each batch, the model is updated to decrease loss.

We started model training in simulation with unsupervised reinforcement learning. As expected, we observed that the loss decreased between batches for most models we trained, reaching a plateau. However, the performance of the model with the greatest best reward mean was still poor when compared to PID.

At this point, we followed several steps to debug our RL Training scheme (Jones). This included modifying the reward function, batch size, number of layers in the net, and the action space. The most impactful was normalizing the inputs/observations to the model. We found that wheel velocity was consistently much greater, so its impact on model output was disproportionate. However, even with this improvement, the model performance was still worse than PID as it was unable to balance the ball for greater than 10 seconds.

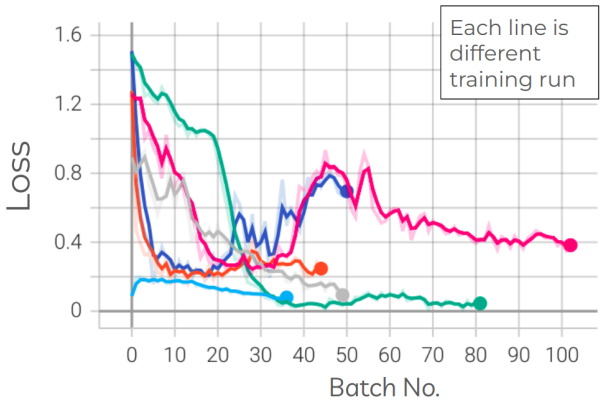


Figure 11: Loss function with unsupervised reinforcement learning

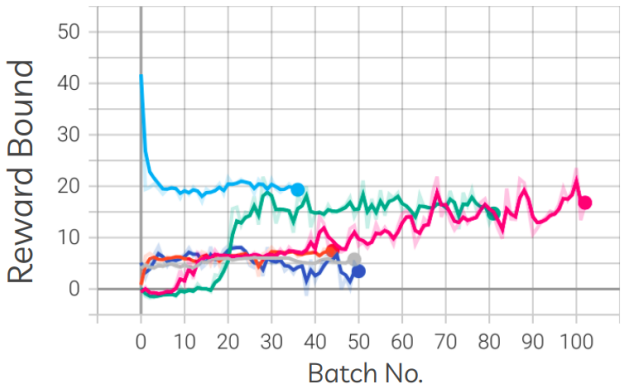


Figure 12: Reward mean with unsupervised reinforcement learning

To determine whether or not a model was capable of learning with the PID control, we began training with supervised imitation learning (IL). With IL, instead of updating the model by comparing the best episodes to the current model, the comparison is between PID and the current model. The figure below depicts the loss after training with IL. Comparing the scale of IL loss and RL loss, we see IL loss is much smaller. This indicates the model has learned PID. We also observed the model now balanced the ball perpetually!

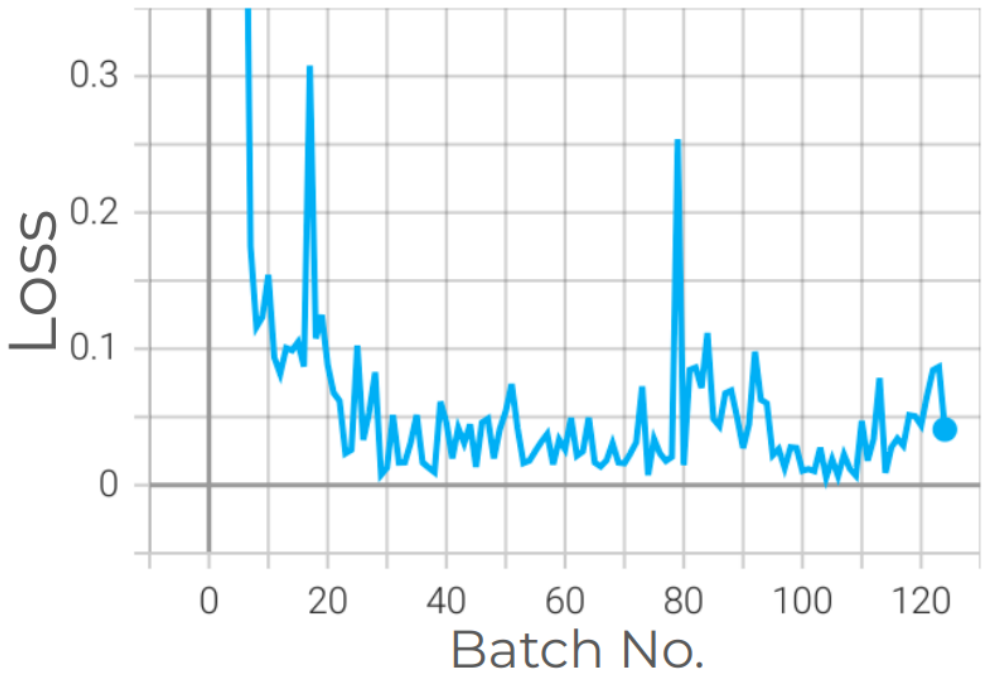


Figure 13: Loss with IL training

2.3.2 Real System

The telemetry and data logging class provide insight into the system, allowing tuning and debugging. First looking at the final closing times, these are broken into two components image processing closing times, and hardware control loop times. Analysis focuses on closing times, divided into image processing and hardware control loop times. Image processing encompasses the duration between frame reception and ball position transmission to the control code. The control code duration is measured from ball position reception to successful PWM signal transmission through the USB port. Results from a previous test day during the project fair are shown in Figure 14. With hardware closure loop times averaging approximately 0.088 seconds and image processing times averaging approximately 0.0788 seconds, both systems respond swiftly enough to achieve the desired outcome of balancing the ball.

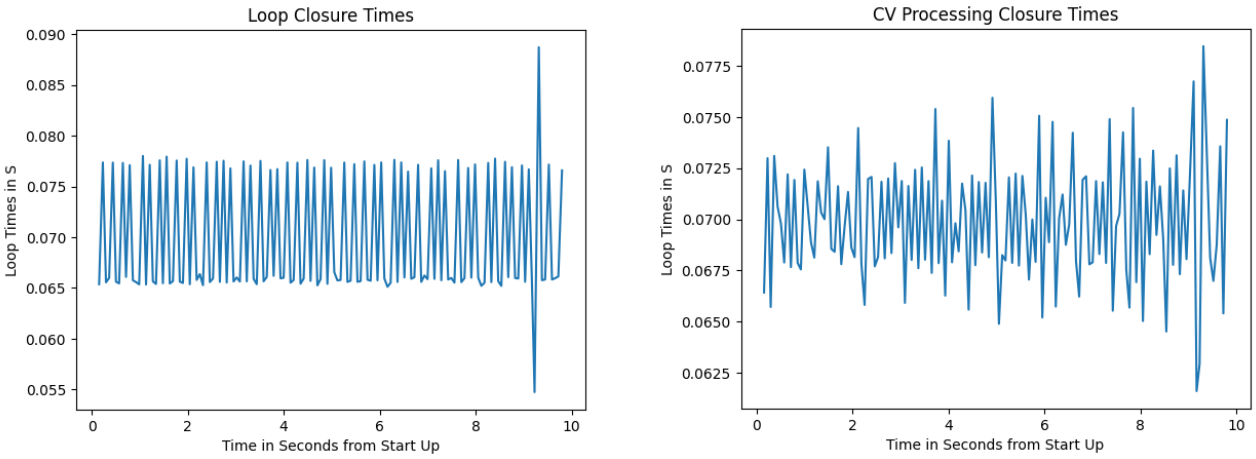


Figure 14: Hardware and image processing closure times respectively from code ran at the project fair.

Each control approach in simulation has a counterpart in reality to facilitate future simulation integration. This includes wheel position control and wheel velocity control. Test results, illustrated in Figures 15 and 16, demonstrate the effectiveness of these approaches. Both wheel position-based control and wheel velocity control are successfully implemented.

Classical Control of Ball on Wheel

The ball-on-wheel setup (Figure 2) was subjected to classical control theory in Appendix .1, and it was found to be controllable using a PID error control, where error is defined as θ_b (as per Figure 2).

We also had a term K_w to penalize the wheel's velocity, which we viewed as a regularizer term. In the end, we determined that this was behaving the same way as K_i, so we ended up setting it to zero. To tune this controller, we started by increasing K_p until the system entered an oscillatory state, but the oscillations were diverging. We then increased K_d, which decreases overshoot. When

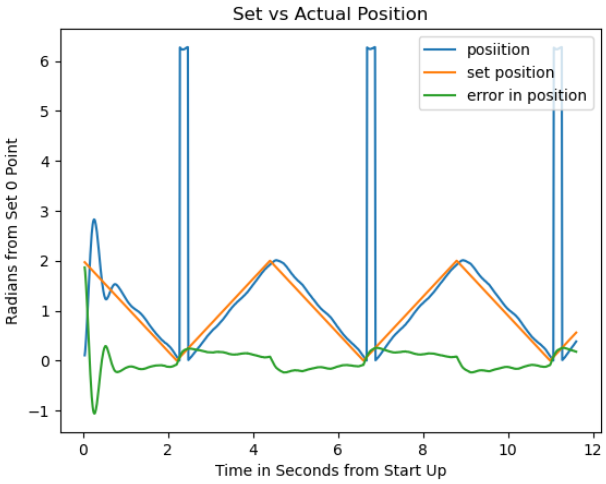


Figure 15: Position based control results from a scaling test. Note the roll over from 0 to 2π.

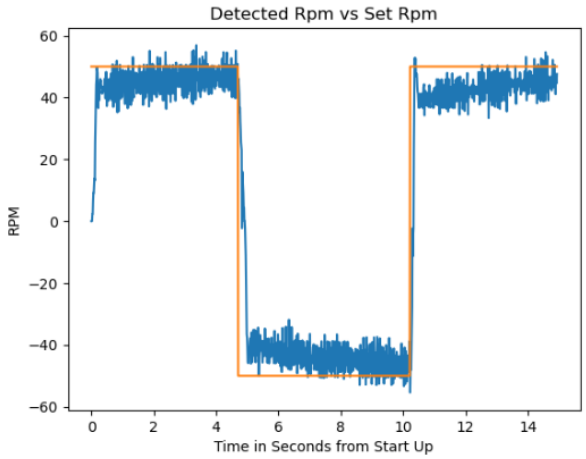
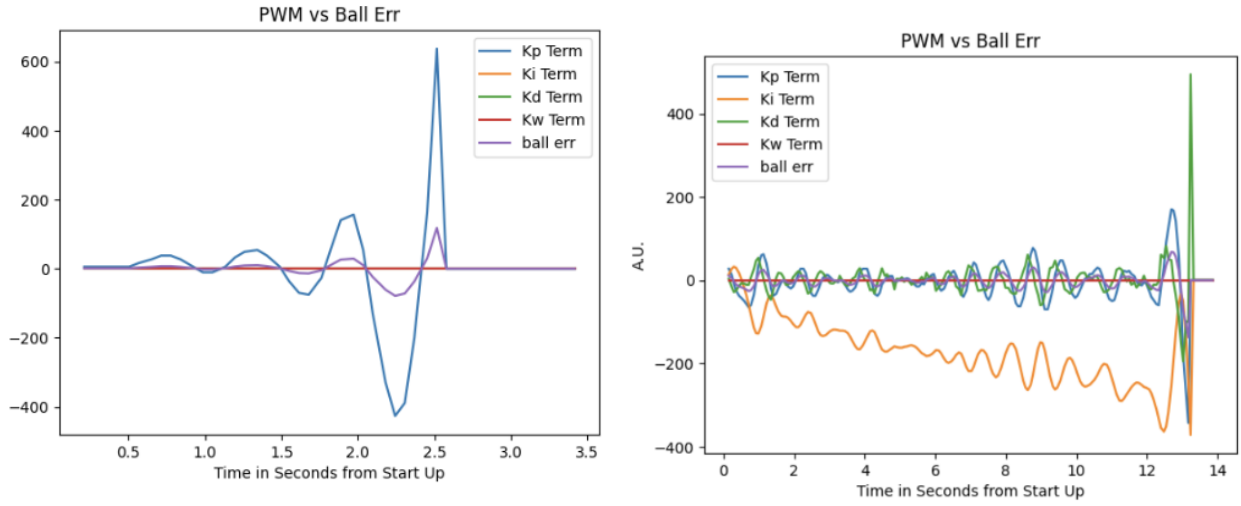


Figure 16: Stress test of wheel velocity PID response.

we found the rise time was too slow, we increased K_i term. We repeated this repeatedly until we found a stable set of PID parameters in Figure 17. Notice that K_i is significantly higher than the other parameters. One way to reason about this is that the integral term builds over time following the inertia that builds up in the wheel's rotational motion, and so K_i serves to account for that build-up.

3 Conclusion

Our exploration reveals significant differences between PID parameters in the real system and simulation. The real system employs a PID controller with $K_p = 2.5$, $K_i = 15$ and $K_d = 0.225$, while the simulation uses a PD controller. Friction between the wheel and the ball proves crucial in the real system's success, a factor not accounted for in the current physics engine. Without control



(a) $K_p = 5$, $K_i = 0$, $K_d = 0$,$K_w = 0$.
These parameters lead to an unstable system

(b) $K_p = 2.5$, $K_i = 15$, $K_d = 0.225$, $K_w = 0$.
These parameters lead to a stable system

Figure 17: PID control of Ball on Wheel system with two parameters, one leading to stable system, the other leading to unstable system.

over friction, the simulation fails to model collision dynamics accurately, widening the simulation-to-reality gap as track geometries become more complex.

The peanut-shaped wheel performs well in simulation, but is inconsistent in reality due to shaft slippage and the absence of a reset mechanism to autonomously train. Imitation learning shows promise in simulation, as depicted in Figure 13, but for the real system requires autonomous resetting to go further. Implementing a reset mechanism, and improving the physics engine are necessary for achieving the original goal of exploring sim-to-real transfer of reinforcement learning models on physical platforms.

4 Recommendations

To better execute bringing control models across the sim-to-real gap, we recommend bringing the simulated system closer to the current physical implementation. This includes:

- Switch to a physics engine with more advanced collision dynamics, which will better model the ball-and-wheel interaction.
- Build a motor model with measured values from the current motor, allowing the simulation to model the non-linear wheel response.
- Further characterize the real system control latency and model it in the simulation control loop.

For further areas of exploration, we recommend:

- Finish implementing peanut-track control in reality.
- Use current design and parts to finish the reset mechanism for training models in real.
- Take the current imitation learning models and use them as a basis for successful reinforcement-learning training.
- If successful, add additional track designs for different control problems, designed specifically for challenging reinforcement-learning training.

We recommend that the Engineering Physics Project Lab continue this project, focusing heavily on the implementation of reinforcement-learning models with the current ball-on-track system.

5 Deliverables

5.1 Paperwork

Logbooks, expense tracking, our presentation slides, and our poster is found in the *Deliverables* section of the team Google Drive.

https://drive.google.com/drive/folders/1b01DVaHljX4LgL9jtRVRrvh8hnJMX1qJ

5.2 Physical System

The physical setup has been left on our table in the project lab for collection, alongside the partially-assembled reset mechanism parts. The laptop is stored in our bin for safekeeping.

5.3 Software

Our software for the wheel system and simulation is found in our team's GitHub organization.

https://github.com/Capstone2302

5.4 CAD Documents

Onshape CAD documents for varying parts of the system are also included.

- Wheel system design
- Reset-mechanism conceptual design
- 2-finger claw design

- Reset-mechanism ball return
- Camera mount

References

[1] T. Kaushik, A. C. Jahagirdar, and S. Singhai. Sliding mode control of ball-on-wheel system. In *2019 IEEE 16th India Council International Conference (INDICON)*, pages 1–4, 2019.

[2] Maksim Surov, Anton Shiriaev, Leonid Freidovich, Sergei Gusev, and Leonid Paramonov. Case study in non-prehensile manipulation: Planning and orbital stabilization of one-directional rollings for the 'butterfly' robot. *Proceedings - IEEE International Conference on Robotics and Automation*, 2015, 05 2015.

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

List of Figures

Appendices

.1 Ball On Wheel Classical Controllability Analysis

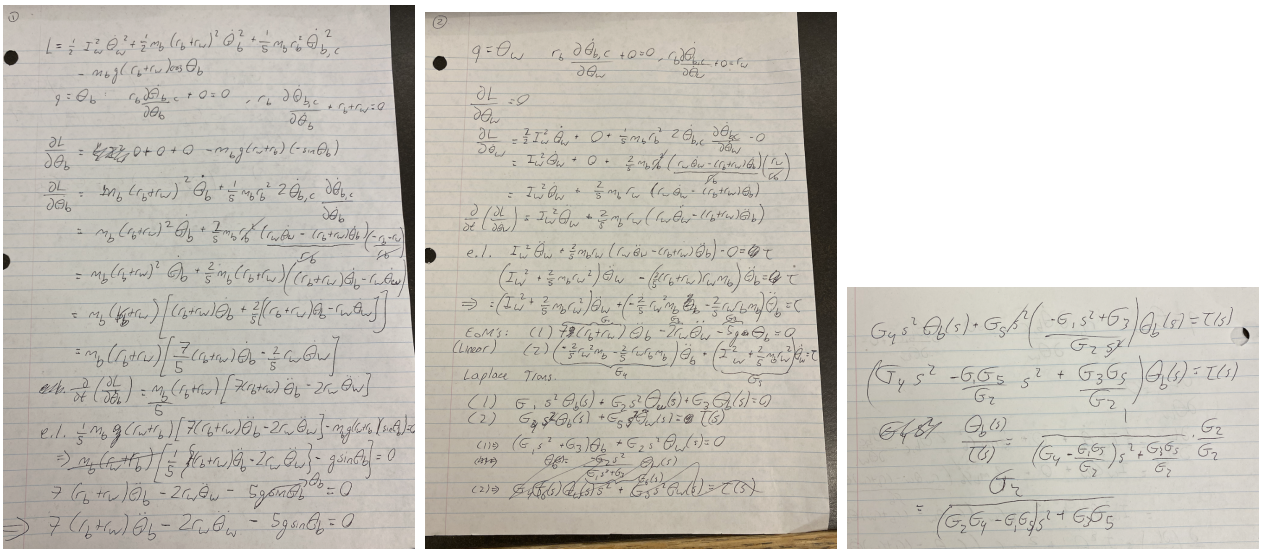


Figure 18: Deriving transfer function of input angle of wheel to output angle of ball with respect to wheel using Lagrangian mechanics.

.2 Video of Ball Balancing on Wheel

A video of a ball balancing on the wheel, taken at the Engineering Physics Project Fair:
https://www.youtube.com/watch?v=y72VbFgktZc