Stewart Platform for Unsupervised Machine Learning

Thomas Bichel
Thomas Deckers
Nathan Maguire
Dante Prins

Sponsored by Engineering Physics Project Lab:
Dylan Gunn
Miti Isbasescu
Bernard Zender

Project 2211
Engineering Physics 459
Engineering Physics Project Lab
The University of British Columbia
April 8th, 2022

Executive Summary

It is well understood that agents trained in purely simulated environments will reliably be outperformed by those trained even partially in the real world when applied to real-world machine learning (ML) challenges. This discrepancy is called the "reality gap".[1]

Knowing this, the incentive in developing methods to train neural networks in real-world environments is abundantly clear. The challenge is that repeatedly and consistently staging a real-world environment for a neural net to train on is an expensive proposal, typically requiring at minimum human supervised resetting of the training stage.
In this project, we have demonstrated that automating such a training platform is far less expensive than anticipated for sufficiently constrained problems. Using only a low-cost implementation of a 3-degree of freedom platform (Stewart platform), consumer grade optical tools (Raspberry Pi, Raspberry Pi Camera), and a mechanism made up of readily available mechanical components, we were able to construct a training platform for a neural net specifically designed to solve a classic marble maze game.

Long-term stability and consistency tests are required before we can make any concrete conclusions about the system's efficacy. Specifically, test cycles using classical control need to be run on the system with intermediate supervision for several days at a time to identify potential failure cases from mechanical position shifting and edge cases in which the marble fails to reset correctly.
With any mechanical failings determined in these tests rectified, the control loop can then be outfitted with ML architecture where the results will reveal the capability of machine learning to solve mazes attached to our system.

The relative simplicity of our implementation supports the viability of a low-cost paradigm for the real-world training of neural nets, and provides a foundation for multiple research avenues, such as the extrapolation of training on simplified real-world platforms into progressively more complex environments, or a simplified regime for optimizing the ratio of simulated to real data for performance against expenses.

[1] https://ai.googleblog.com/2017/10/closing-simulation-to-reality-gap-for.html#:~:text=The%20difficulty%20of%20transferring%20simulated,enabling%20effective%20real%2Dworld%20performance.

Table of Figures:

Appendices:

1) Introduction

In the training of hardware-controlling neural networks (NNs), the primary obstacle in bridging the gap from simulation to reality lies in the construction of a training environment that a NN can manipulate and explore over many iterations without supervision. The UBC Engineering Physics project lab is interested in exploring this gap in the study of machine learning by constructing simple versions of these environments and analyzing the data obtained from such apparatuses.
The goal of this project was to develop a platform for unsupervised real-world NN training on a problem with an adequate degree of complexity to utilize the greater generality that neural nets have over traditional methods.
The problem selected was a traditional platform maze game, in which a marble is placed on a maze platform that may be tilted with two knobs on the side of the box, controlling two of the angles in which the maze may tilt.

[2]

Figure 1: Inspiration for our chosen environment: a marble maze table game

The dynamics and optimal solutions of this system are sufficiently complex that analysis by an NN may yield interesting behaviors and flexible enough that unorthodox solutions could feasibly emerge given sufficient training time.

Incorporating this system into an autonomously repeatable neural net training environment essentially breaks down into designing 3 functional subsystems: a means of deterministically controlling the dynamics of the ball, a means of gathering consistent data on the environment for learning and eventually training purposes, and a means of resetting the game back to a starting point for repeated trials.

---

[2] https://www.amazon.ca/Hey-Play-Labyrinth-Wooden-Marbles/dp/B072QNFNN3 (TEMPORARY)

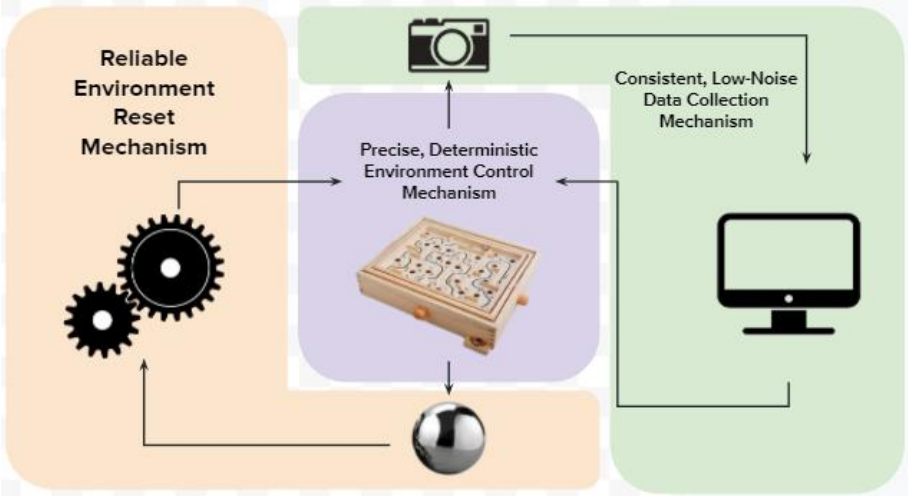


Figure 2: Function-level system diagram

The scope of this project was constricted by a limited time scale, so the feasibility of a fully developed product was relegated to future iterations in favor of a minimal viable product (MVP) to serve as a proof of concept. The electro-mechanical design and construction was specifically emphasized with control software being implemented last to demonstrate capabilities, and not to provide fully general control.

2) Theory

2.1) Subsystem Breakdown

2.1.1) Control Subsystem

We quickly found that a Stewart platform[3]-based design for our environment control mechanism was appropriate for our chosen application. A three-actuator system achieves the two planes of tilt in a classical marble maze with an additional vertical degree of freedom (DOF).

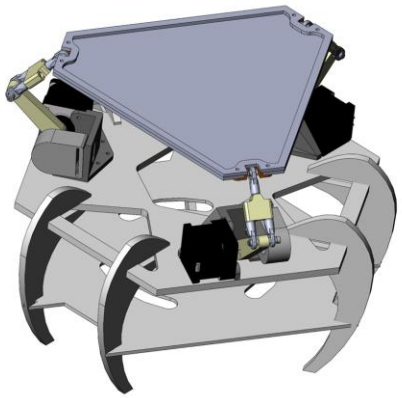


Figure 3: Stewart Platform with 3 degrees of freedom

[3] https://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/f2017/psl58_aw698_eb645/psl58_aw698_eb645/index.html

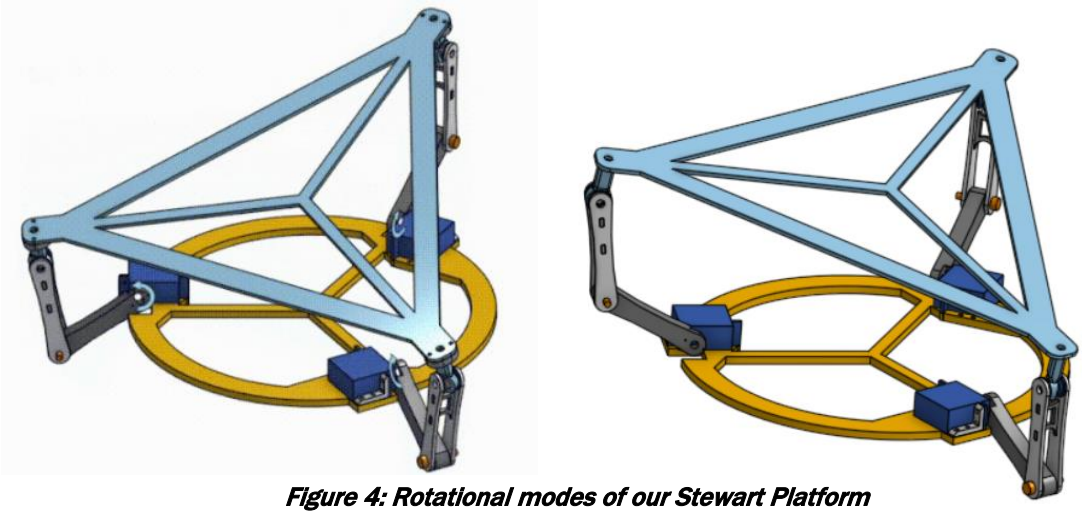


Figure 4: Rotational modes of our Stewart Platform

2.1.2) Data Capture Subsystem

The data capture subsystem similarly came naturally from the problem itself: optimizing the solution of a maze essentially reduces to finding optimal trajectories for a rolling marble on a 2D surface, so a camera positioned with a bird's eye view of the entire maze and a sufficiently fast response time to accurately capture the marble's dynamics was a natural choice.

We additionally included a bright LED light shining on the platform, so that the image captured by the camera is more consistent in different external lighting conditions. To this end, the lighting module should ideally be sufficiently bright to make up the majority of light cast on the platform, even in the brightest operating environment of the system.

2.1.3) Environment Reset Subsystem

The goal of this system is to retrieve a marble that has somehow fallen out of the play area and return it to a consistent location in the maze to resume normal functionality. Great emphasis is placed on reliability for thousands of cycles so that the system is fully autonomous.

2.2) Environment Control Algorithm

A well-defined control scheme for the environment is a vital prerequisite for an NN to be capable of developing more complex behaviors. In robotics, techniques for obtaining the actuator states require positioning an end effector in a specified state. Typical inverse kinematics problems have no closed form solutions and instead

require numerical approximations to solve. However, for Stewart platform-type systems in which the number of actuators is equal to the degrees of freedom, the inverse kinematics have an analytical solution.[4]

The problem reduces to first defining the position and orientation of the upper platform with respect to the rigid base, then computing the lengths of the actuating legs corresponding to this platform state, and finally obtaining the servo angles required to produce said leg lengths. A more detailed discussion is outlined in appendix A.

Since the inverse kinematics have a closed-form analytical solution, no iterations whatsoever are necessary to execute it and as such its computation time is on the order any other simple arithmetic scripts. As such, the response time of the system's control script is negligible relative to the image processing time and presents essentially no delay in its response time.

[4] https://www.researchgate.net/publication/326513089_Inverse_Kinematics_of_a_Stewart_Platform

3) Design

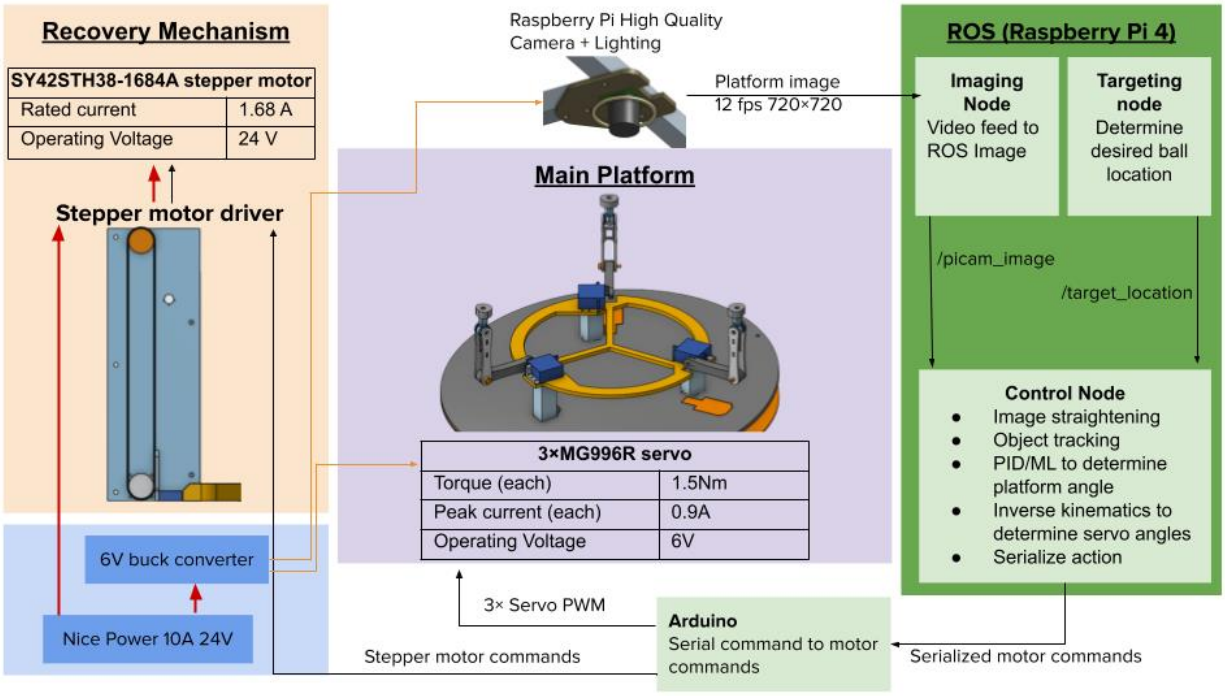


Figure 5: System diagram. Information flow given by black arrows, 24V power given by red arrows, and 6V power given in orange

3.1) Stewart Platform Implementation

A generic Stewart platform consists of three principal components: a mobile upper platform plate, a fixed base plate, and several supporting “legs” whose lengths may be varied using some type of motor, linking the orientation of the platform to the base.
Typically, the motors controlling these legs are some type of linear actuator, however electrical linear actuators are prohibitively expensive and slow for a proof of concept, especially in packages small enough for our system. As such, we adapted the design to use hinge-type arms controlled by rotary servos.

Furthermore, the most general design for a Stewart platform gives the upper platform 6 degrees of freedom via 6 actuators. The loosening of rigid system requirements allowed us to focus primarily on the types of motion that most directly impacted the trajectory of the marble.

The operating principle of our source material (the maze game) is to control the motion of the marble by adjusting the incline of the platform and using gravity to propel its motion. In essence, the critical degrees of freedom were rotation around

the two horizontal axes (roll and pitch). To simplify preliminary control and analysis, we reduced the complexity of the platform to 3 degrees of freedom: roll, pitch, and vertical height, and hence reduced the number of actuators needed to 3.
The vertical height was not introduced as a necessary dimension of the system, but to add an element of symmetry to the design, enabling quicker CAD and construction. The additional motor for the third leg also introduces an additional degree of freedom, which worked out to elevation control by virtue of our choice of rotary motors.

3.1.1) Dynamic System Performance

With the goal of having a responsive system, the dynamics of the platform are the primary measurement of performance of the platform; a faster platform can change the motion of the ball more rapidly, leading to a greater upper limit on the controllable speed of the ball, raising the maximum potential performance of an agent.

Calculations and simulations were conducted with the goal of qualitative characterization of the system, outlined in appendix B. The primary question was whether the system could be controlled with available 1.5Nm servos and what level of performance could be expected.

The simulations produce graphs such as figure 6, which relate starting ball speed to stopping distance.

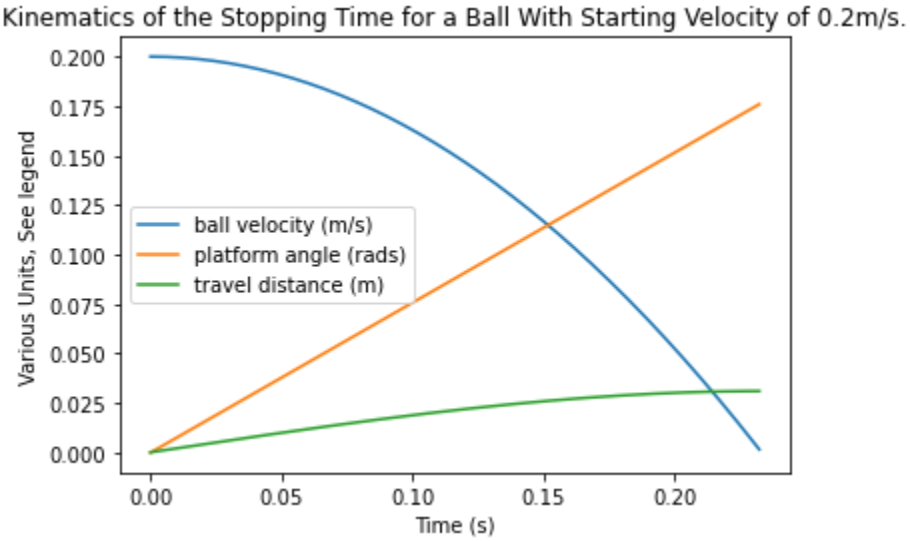


Figure 6: Kinematics of the stopping distance for a ball starting at speed 0.2m/s on a flat platform.

The data suggests we could stop the ball in ~2.5cm (seen in the figure above). The maximum marble velocity is reasonably constrained to below 0.2m/s, so this performance was deemed acceptable for an MVP, so we proceeded with the available 1.5Nm servos.

3.2) Mechanical design

The final implementation of our Stewart platform was designed around reducing motion in unintended degrees of freedom ("slop" or "play") as much as possible. Any backlash or sliding between parts has the potential to introduce dynamic error into our system; error that our computation process' cannot readily predict or account for.

For example: the hinged type "legs" (previously mentioned), used a tight tolerance bushing and shoulder bolt assembly with a wide center arm (Figure 7) to minimize slop when lateral torque is applied to the arm, while still being able to rotate freely.

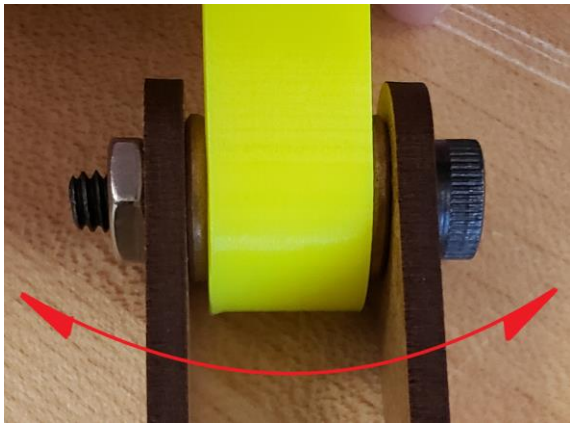


Figure 7: Arm hinge. Arrow denotes the direction of torque resistance

More examples of designing around degrees of freedom while minimizing slop can be found in appendix C.

3.3) Data Capture and Analysis

An ML training platform's performance hinges on its ability to collect reliable training data. For our case of image data, this means collecting data at a high enough rate to accurately manipulate the ball at its maximum velocity, and with the precision to accurately observe the system, to avoid providing false or inconsistent data to an ML agent.

For the purposes of this phase of the project, our goal was to design a hardware system that could meet these goals, and a software system that could demonstrate this.

A ROS architecture with images captured by a Raspberry Pi High Quality Camera (Picam) was employed.

3.2.1) Data capture hardware

With the goal of a robust and adaptable system, we sought out a framerate of 60 fps, and resolution of at least 720p. We avoided webcams designed for regular consumer use e.g. in video calls, as these provided limited, if any, control of operating parameters, and are more challenging to control from a script.
There are a limited number of cameras designed for microprocessors, and of those available, the only real-time camera capable of achieving the above specs is the Picam. This requires use of a Raspberry Pi board, we use a Raspberry Pi 4 with 4gb of RAM (the Pi). For a discussion of software and versions installed on the Pi, see appendix E.
Since the Pi only has two independent hardware PWM signals, we used an Arduino board to forward signals from the Pi to the servos.

Finally, as mentioned in the theory section, we included an LED ring light mounted around the camera to illuminate the platform, thereby improving consistency across trials. The particular lighting unit we used became hot to the touch, so we added a heatsink and small fan.

Figure 8: High-intensity LED to saturate camera feed beyond ambient light levels.

3.2.2) Data analysis and control software

For this phase of the project, the primary purpose of the software is to prove that the control system can accurately and consistently capture the position of the ball and respond in real time. To this end, classical (non-ML) control is easier to manipulate and provides a more transparent proof of functionality. In addition, this classical code provides a starting point for machine-learning applications, allowing sections of interest to be replaced with ML algorithms without rewriting basic functionality. Potential implementations of ML algorithms are discussed in appendix F.

A detailed description of algorithms used, and suggested improvements is included in appendix F.
Briefly, we straighten the image of the platform by identifying the four pink stickers of the platform using HSV identification. From this straightened image, we can accurately identify the location of the ball first using motion detection via image subtraction, and if that fails, by attempting to identify the pattern of light and dark spots that is unique to a reflective marble. Knowing the ball location, we obtain a target platform angle using 2 dimensional PID control. We can then use the inverse kinematic discussed in section 2.2 and appendix A to convert this platform angle into servo angles, and finally forward this to the Arduino using serial communications.

This implementation was slower than necessary for precise control, taking approximately 0.08s to process each frame. However, it did function reliably, and served its purpose in testing the function of the hardware.

3.4) Reset Mechanism

The central design requirement for the reset mechanism was consistency since the core concept of our system relies on its reliable functionality over thousands of cycles. Mechanical components are the most likely to break down over many cycles, so the general approach to the design was to use as few moving parts as possible at the expense of expandability or elegance to achieve minimum functionality.

The principal function of the reset mechanism is to take a marble that has fallen from the platform, regardless of how it fell, and elevate it in order to return it to the platform. This allowed us to segment it into two primary components: a collection ramp and an elevator.

We elevated the Stewart platform and removed the inner part of the base plate to provide clearance for a 10-degree collection ramp beneath the maze, with an expanded surface area to catch falling marbles from the vast majority of their possible trajectories. The ramp terminates in a funnel to a single track, thus keeping the marble path to the elevator consistent regardless of failure case.

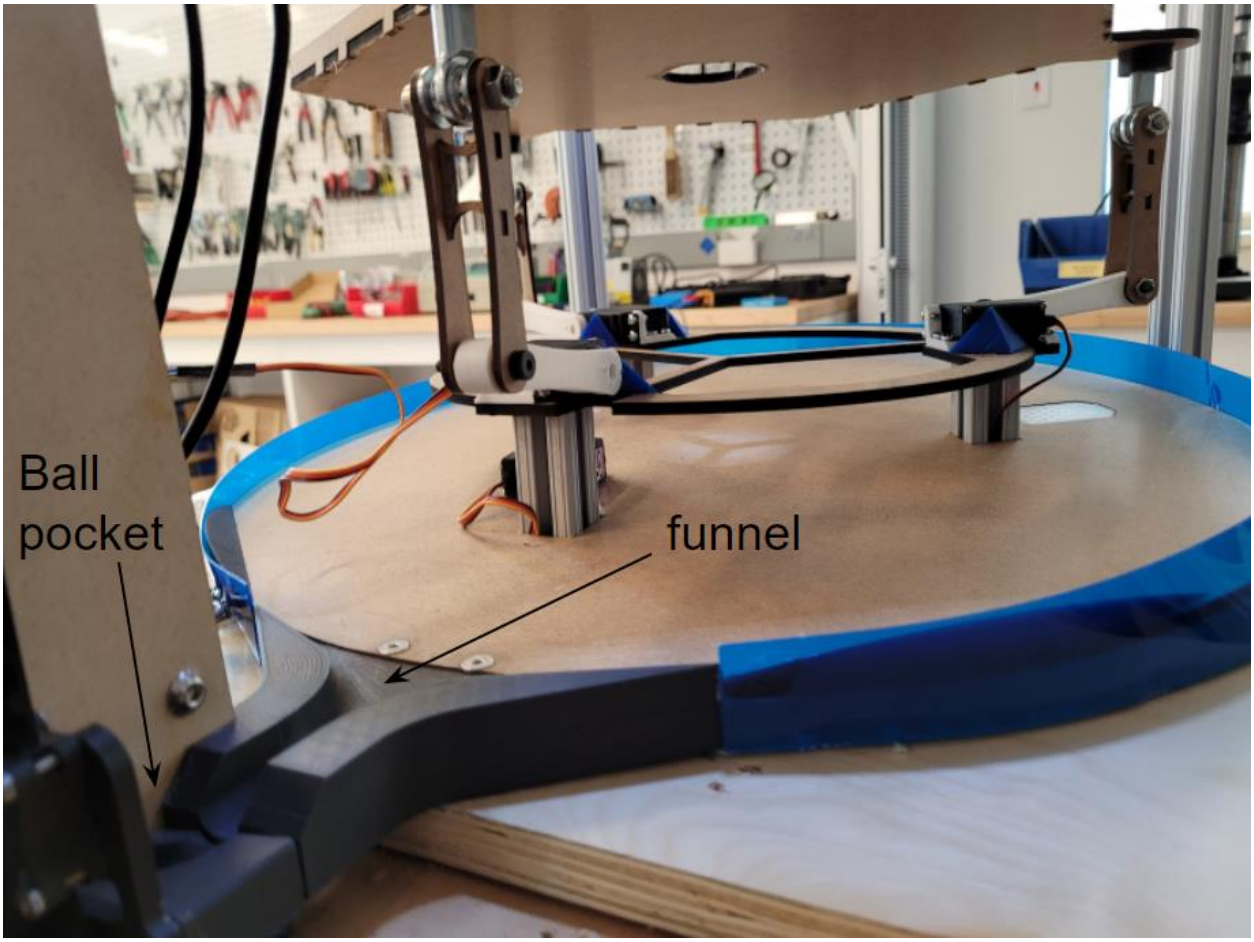


Figure 9: Reset mechanism: collection ramp

After rolling in, the ball pocket is elevated by a belt drive (figure 10a) until reaching a hole in the top and rolling onto the slide (figure 10b) that directs the marble back to the maze.

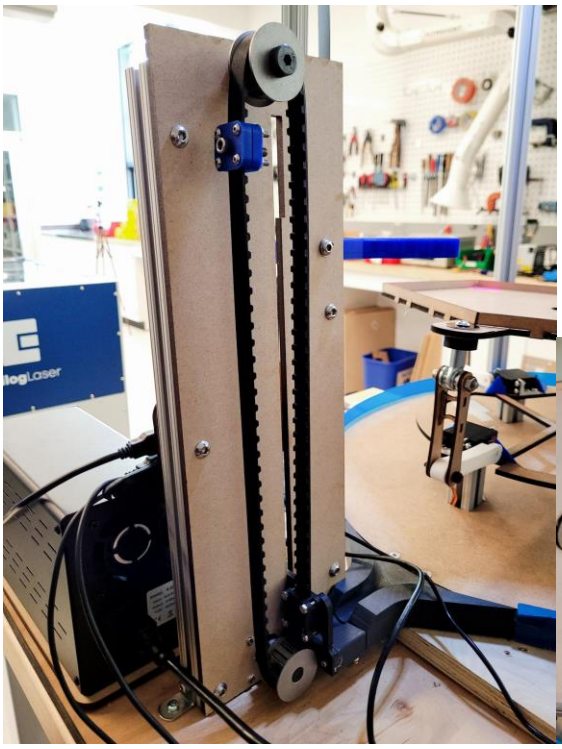


Figure 10a: Marble elevator.

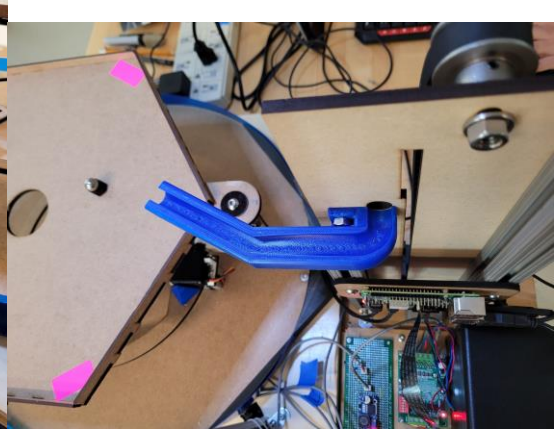


Figure 10b: Marble slide.

3.5) Validation Testing

The functionality of the inverse kinematic algorithm was verified on the platform by obtaining a sequence of servo angles corresponding to pre-defined platform orientations and physically measuring the incline of the platform.

The functionality of the control pipeline was verified by placing the ball on a plastic nut so that it didn't roll, sending targets in perpendicular directions and ensuring that the platform tilted the ball towards the target. When the ball was free to roll, the current implementation was unable to bring the ball to a stop at an arbitrary location, and was only able to bring the ball to specific corners, depending on the calibration. A video of the control algorithm in action is linked in appendix G. The two biggest factors that we were able to identify in this low degree of control were first the low framerate and high latency of the image processing pipeline, and second the high degree of slope and variability among the servos.

The system was also given a light practical test while it was on display at the 2022 Engineering Physics Project Fair. Over the course of the evening, it was able to continually attempt simple tasks unsupervised for up to an hour at a time. This demonstrates performance of the mechanical system, but long-period reliability

testing is still required before the system is proven to be suitable for ML training, this test is outlined in 5.1.

4. Conclusion

The most valuable result obtained from this project is demonstrated in its consistent, if minimal performance on the night of the Project Fair. The system was assembled with the most minimal components available, introducing a large degree of instability in both the mechanical system from the axial instability of the motors, and the software system from Raspberry Pi's hard upper limit on the system's processing time.

Yet despite these limitations, it succeeded in its primary task; without supervision our control script was able to use the electro-mechanical system to manipulate, analyze, and make decisions, consistently and without supervision over hour-long time scales.

Though the application of simply trying to stop the marble in a prescribed location is exceptionally simple relative to the types of complex behaviors desired from the successor of this system, the system's consistent functionality demonstrates its minimum viability, and holds open an avenue for future exploration with greater computing power and more robust actuators.

5) Next Steps

5.1) Reliability Tests

As of this writing, the system has been fully assembled and the reset routine, camera image and servo control have been verified. Validating its performance as a proof of concept will require testing its ability to maintain consistent performance over progressively longer time scales.

The first round of tests should have the system checked with fair regularity, as this early stage will primarily be concerned with identifying edge cases in the system's reset mechanism that prevent further function such as locations in the recovery platform wherein the marble tends to get stuck. Once these failure points are addressed with light modifications such as in figure 11, the system should then be allowed to run for days at a time to uncover remaining traps for the ball to reach the desired reliability of thousands of cycles.

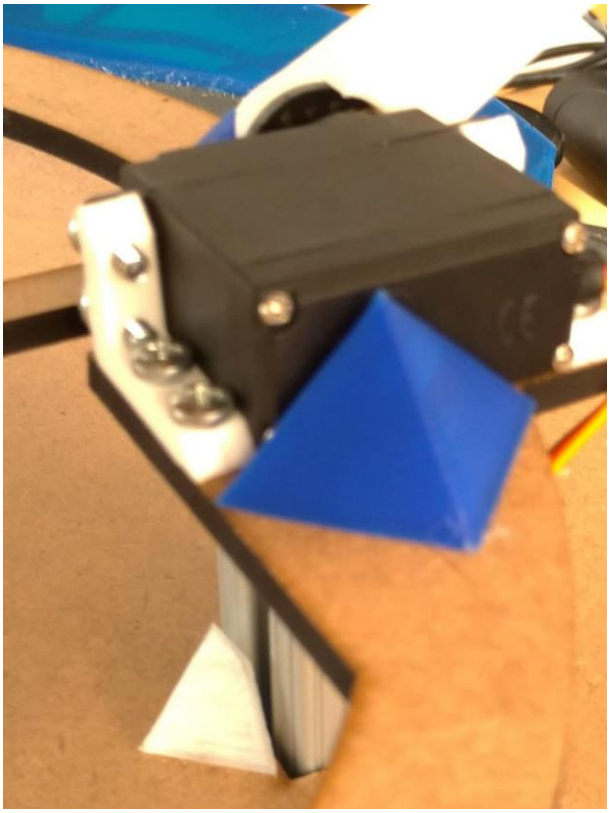


Figure 11: Early "sticking spots" iteratively modified with triangles after observing a stuck ball.

5.2) Transition from Classical Control to Machine Learning

In the “data collection and analysis” section, we described several improvements and alternative algorithms that could be used in classical components of a control pipeline.
The ultimate intended application of this platform is for ML control. This is an open-ended problem, with a diverse range of approaches, each with their own strengths, weaknesses, and insights on ML techniques. We describe a few possibilities for this in appendix H.

5.3) Component Improvements to Increase Performance

The performance of the system was impeded by intrinsic limitations of the minimal parts used in its construction.
The servo motors employed as actuators were highly inexpensive hobby-tier models, and hence the system was limited by their low-grade mechanical design: their motor axes had no internal supports, and so their plane of actuation was prone to horizontal oscillations that interfered with the intended trajectory of the marble. Adjusting the system to use servo motors fixed with rigid axles would reduce mechanical tremors and make the theoretical actions sent by the code more precisely correspond to real-world trajectories.
The limited processing power of the main computing chip also hindered real-time behavior in the system. Our system partially relied on cycling through image feedback with a high frequency, ideally on the order of a human eye’s perception of around 60 Hz. The picam was able to provide data at this rate, but the raspberry pi was unable to process image data at this speed, limiting our processing rate to around 10 Hz in runtime, and in essence slowing the reflexes of any system implemented with it. Running the image processing instead through a NVDIA Jetson unit or a desktop computer would substantially improve reaction time and smoothness.

5.4) Summary and implementation

The proposed tests can be performed on the existing system to get a sense of the required areas for improvement, and the areas that have already received attention in our design.
However, the discoveries that we expect to be made in this testing process, in addition to the hardware suggestions above, are significant enough that we recommend a full redesign and reconstruction of the platform.

6) Deliverables

Enclosed Stewart Platform System delivered to project lab April 9th

- Stewart Platform
- Recovery System
- Arduino
- Raspberry Pi
- Raspberry Pi High Quality Camera and 12mm c mount lens
- Nice Power supply returned to storage

System Control Script shared with fizzym via Github under the package
https://github.com/thomasbd14/ENPH-459-2211-RL-Marble-Maze

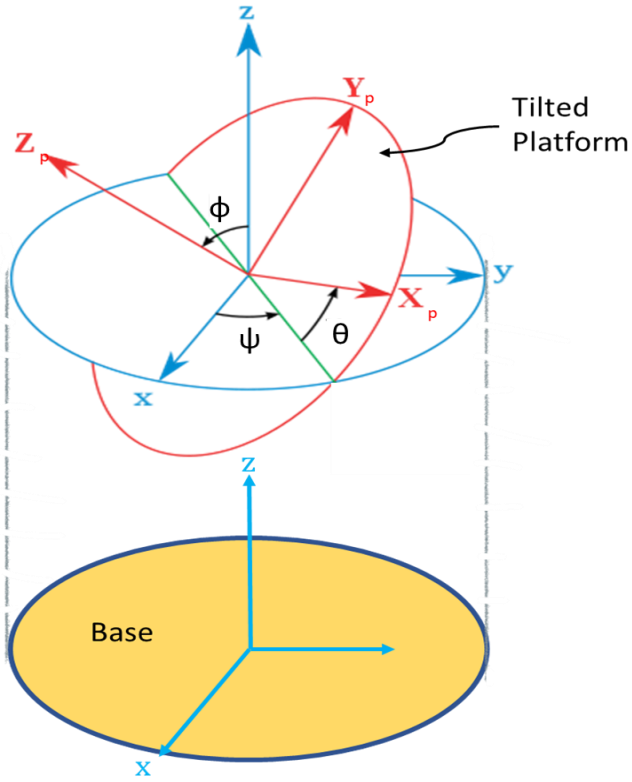
- Classical control pipeline
- Arduino motor control scripts
- Inverse kinematics script

CAD shared with dylangunn@gmail.com & miti.fizz@gmail.com through Onshape

- Main Stewart Platform Assembly
- Belt clamp
- Recovery plate fences
- Electronic mount sketches

Appendix A: Inverse Kinematics

This algorithm relied heavily on one tutorial paper cited multiple times across multiple papers in the literature.[5]
The problem is presented as such: a Stewart platform consists of 2 planar surfaces called the "platform" (***P***) and the "base" (***B***) such that the orientation of ***P*** is entirely dictated the states of each of the joints between ***P*** and ***B***, which are generally formed by a collection of actuators that dictate the lengths of the legs.
The orientation of the platform with respect to the base is described by the series of 3 angles required to rotate the base coordinate axes to that of the platform coordinate axes. These are typically referred to as "Euler Angles", ***Figure 1A: Diagram of Euler Angles*** and can be used to construct rotation matrices that convert vectors in the platform reference frame to the base reference frame.



It is a matrix multiplication of a rotating matrices of all 3 individual axes

$$
{}^{P}\mathbf{R}_{B} = \mathbf{R}_z(\psi).\mathbf{R}_y(\theta).\mathbf{R}_x(\varphi)
$$

$$
= \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} . \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} . \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \sin\varphi & \cos\varphi \end{pmatrix}
$$

$$
= \begin{pmatrix} \cos\psi\cos\theta & -\sin\psi & \cos\psi\sin\theta \\ \sin\psi\cos\theta & \cos\psi & \sin\psi\sin\theta \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} . \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \sin\varphi & \cos\varphi \end{pmatrix}
$$

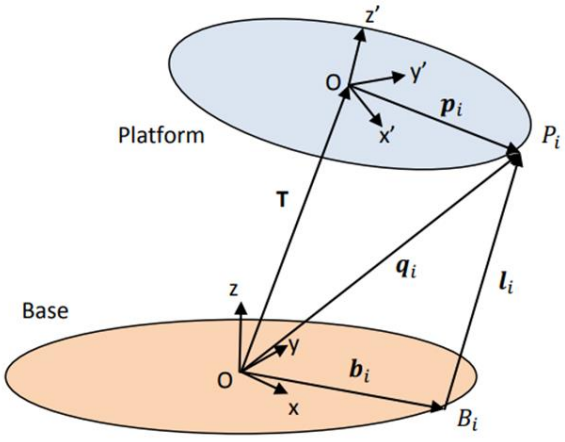
$$
= \begin{pmatrix} \cos\psi\cos\theta & -\sin\psi\cos\varphi + \cos\psi\sin\theta\sin\varphi & \sin\psi\sin\varphi + \cos\psi\sin\theta\cos\varphi \\ \sin\psi\cos\theta & \cos\psi\cos\varphi + \sin\psi\sin\theta\sin\varphi & -\cos\psi\sin\varphi + \sin\psi\sin\theta\cos\varphi \\ -\sin\theta & \cos\theta\sin\varphi & \cos\theta\cos\varphi \end{pmatrix}
$$

:

Equation 1A: Platform-to-base rotation matrix with angles listed in figure 1A.

[5] *The Mathematics of a Stewart Platform*, Unknown, Wokingham U3A Math Group

These rotation matrices can be used to rotate any vector in the platform reference frame to the stationary base reference frame. In particular, these can be applied to the platform vectors that point from its centroid to the connecting points of the actuating legs $\overrightarrow{p_i}$. From there, simple vector addition using the input translation vector between the two centroids $\vec{T}$ and the constant base-leg attachment point $\overrightarrow{b_i}$provides the vectors describing the displacements between each actuating attachment point $\overrightarrow{l_i}$.



$$l_i = \mathbf{T} + {}^{P}\mathbf{R}_{B} \cdot p_i - b_i$$

Figure 2A: Model of Stewart Platform

The magnitudes of these vectors are in turn directly controlled by their respective actuators. In the case of our rotary application, this corresponds to the necessary angle for a servo to take. For our inverse kinematics, the displacements resulting from a given servo angle were physically verified, and such practice is encouraged in all applications, but on average the following algorithm may be followed:
Let $\overrightarrow{a_i}$ be the coordinates of the arm/leg joint of the i^{th} servo in the base frame, s denote the length of the length of the segment hinged to the arm of the servo, α denote the servo's angle from the horizontal, and β indicate the servo's actuation plane's angle from the x-axis.
Then:

$$\alpha = \sin^{-1}\frac{L}{\sqrt{M^2+N^2}} - \tan^{-1}\frac{N}{M}$$

$$\text{where} \quad L = l^2 - (s^2 - a^2)$$

$$M = 2a(z_p - z_b)$$

$$N = 2a\left[\cos\beta\,(x_p - x_b) + \sin\beta(y_p - y_b)\right]$$

Equation 2A: Translation from actuating leg lengths to actuator states

Where the subscripts "p" and "b" refer to being individual components of the aforementioned vectors $\overrightarrow{p_i}, \overrightarrow{b_i}$. With these vectors being found through the aforementioned rotation matrix or held as constant in this system, respectively, one

then has a closed-form expression to translate a desired platform orientation to the actuator states required to induce it.

Appendix B: Performance Dynamics

This section describes the calculations used to go from servo specs to stopping distance of a ball on the platform.

Using kinematics we can tie the platform rotation speed and position to controllable ball speed. Thus the quality of our implementation of the Stewart platform for ML control can be assessed based on platform speed. This becomes the metric used to determine the performance level of servo motors needed for the prototype.

Calculations

Servo motor speed is a function of applied load, As seen in figure 1B. This drop in applied torque is caused by back EMF from the switching of magnetic fields in the armature while turning rapidly. The applied load from the platform is determined by platform weight.

Torque load caused by angular acceleration of the platform was found to be negligible, thus the load was approximated as simply the weight of the platform. Maximum torque load appears in a configuration where the servo arms were bent to a maximum of 85°, with the platform at 20°, causing torque derating. We obtain the equation:

$$T_{load} = m_{platform} * g * l_{servo\ arm} * cos(85° - 20°)$$

We made some additional worst-case assumptions for this model: the entirety of the platform mass was approximated as a load on a single joint being supported by one servo, with linkage mass is neglected. These assumptions are inaccurate but provide an upper bound that intrinsically imposes a safety factor onto motor loads. Using the torque equation above we find a load torque of 1 Nm assuming a 1kg platform, determining our operating point on figure 1B.

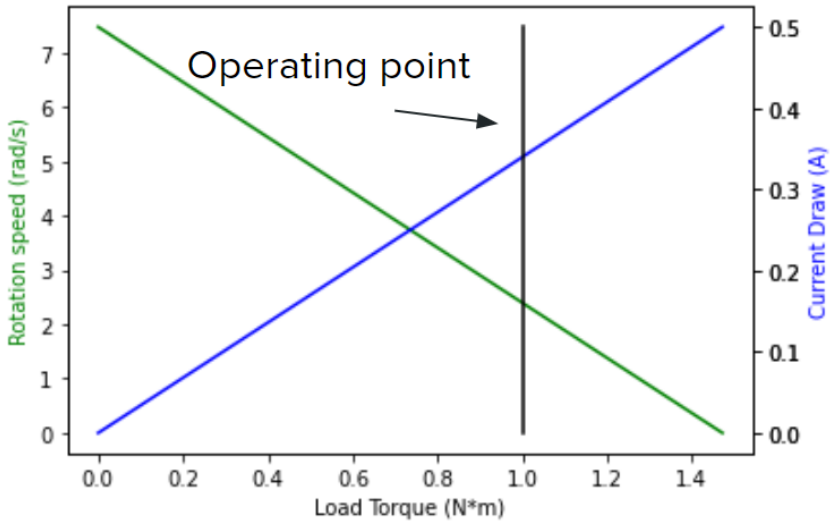


Figure 1B: The Load torque vs. Rotational speed tradeoff curve for the specified 1.5N*m MG996R servos. The operating point is determined by the torque required by the stewart platform and specifies the servo speed for our system.

Using linkage lengths, servo speed is translated back to platform rotation speed, which is used in a kinematic simulation of ball speed to determine stopping distance.

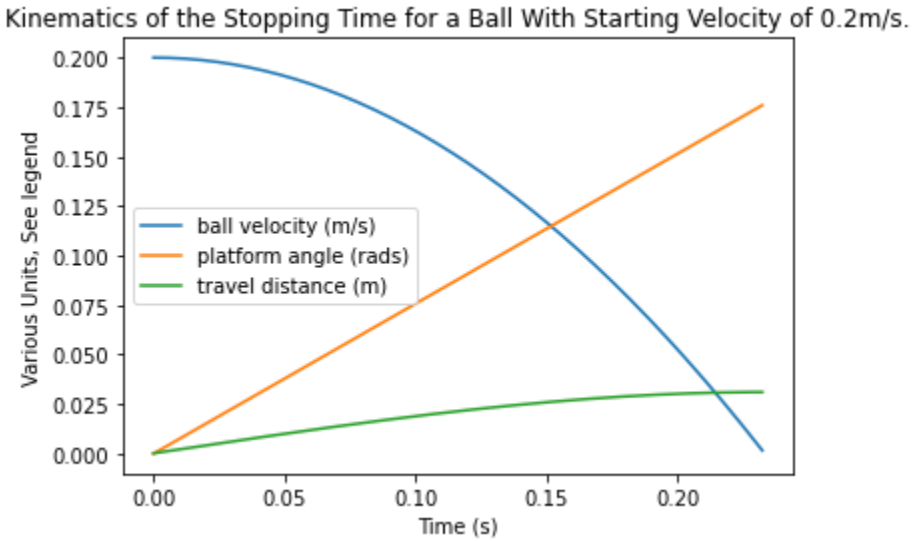


Figure 2B: Kinematics of the stopping time for a ball starting with velocity 0.2m/s on a flat platform.

In figure 2B we see a sample simulation for a proposed starting velocity, derived from the instantaneous horizontal acceleration of the ball due to platform incline:

$$a_x = g * sin(\theta) * cos(\theta), \qquad \theta(t) = \omega * t$$

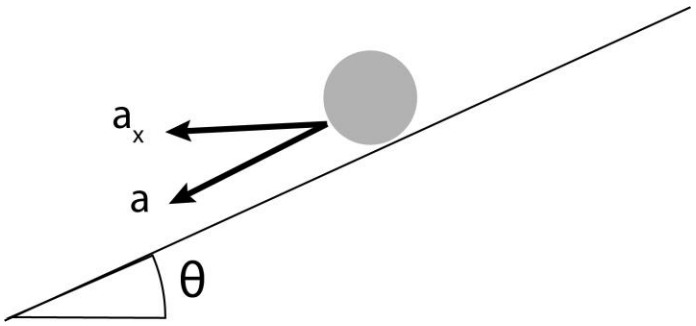


Figure 3B: The instantaneous acceleration on the rolling ball used to determine stopping distance.

It must be emphasized these calculations were not performed with an emphasis on accuracy, with numerous assumptions in the dynamics stated above, and no consideration of ball slipping or sliding. Still, we see satisfactory performance of the simulation with conservative measurements of platform mass and torque loads, giving us confidence in the use of the provided servos for this prototype.

Appendix C: Mechanical Design

Onshape CAD - ***This link is 'view' only. Email Dylan for access to edit.***

The "arms" that enabled the control of the stewart platform via servos were designed to introduce as little slope as possible into our system.

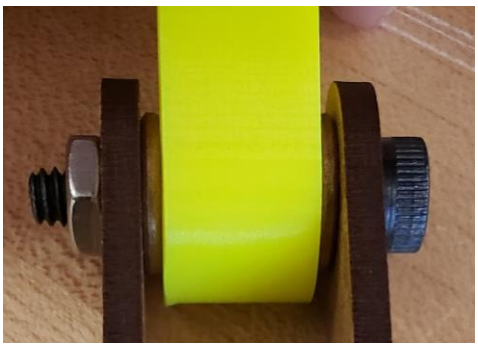


Figure 1C: hinge assembly

The hinge uses a shoulder bolt and bushing assembly to enable rotation while resisting any other motion. The large central pivot provides increased stability against external torques, and tight tolerances on both the bushing's radial dimension and length ensure that there is only looseness in the degree of freedom desired.

On the other end of the servo arm is a 3DOF ball joint.
The ball is rigidly bolted to the arm assembly and the ball joint was specified to have high positional accuracy. There is very little play in this portion of the system except in the degrees of freedom desired.

Figure 2C: Ball joint assembly

As previously noted, time constraints played a large factor in the design and construction of the device. The CAD may be missing parts that, while inconsequential to the overall design, contribute a sort of “mechanical tuning” to the system. While the purpose of most of these can largely be deduced while exploring the system, for propriety, a few will be listed below.

Figure 3C: Recovery fence height addendum: to prevent ball from ramping out of the recovery ramp

Figure 4C: Pyramids: To prevent the ball from resting on any flat surfaces after leaving the maze

Appendix D: Electrical Design

Electrical power was provided by a DC power supply set to 24V and a voltage reducing Buck converter for the servos and ring light. Important information is listed below.

Electrical Component	Voltage	Approximate Current Draw	Power Source	Control connection	Model information
Stepper Motor	24V	0.05A	24V DC Supply.	Control Arduino with isolated ground.	SY42STH38-1684A
Stepper Motor Driver					TB6560
Platform servos	6V	0.25A regular use. (1A draw indicates servo instruction error).	Buck converter 24V to 6V isolated ground.	Control Arduino with shared ground.	MG996R (metal geared from Bernhard)
Ring Light for Camera	6V	0.07A			https://www.adafruit.com/product/5138.

Table 1D: Electrical setup information.

Appendix E: Software installation on the Raspberry Pi

The Project Lab recommends against using ROS 2, as ROS is better documented, and has broader compatibility. ROS and many of its libraries are optimized for Python 2.7, which was used for this project, although this version is no longer officially supported.
ROS is also not compatible with the latest version of Raspbian OS, Bullseye, so we used the previous version, Buster.
Installation of ROS was achieved by following this guide:
http://wiki.ros.org/ROSberryPi/Installing%20ROS%20Melodic%20on%20the%20Raspberry%20Pi
Rosdep, a standard tool for finding and installing ROS package dependencies could not run successfully, so ROS packages had to be individually downloaded from github and built using catkin-make.
OpenCV also did not install via pip, and had to be built from source following this guide: https://docs.opencv.org/3.4/d2/de6/tutorial_py_setup_in_ubuntu.html.
However this process did take about three hours of compilation.

As well, open CV-contrib, which includes many helpful libraries including support for ArUco markers and several tracking algorithms, could neither be installed via pip or from source (although it did take several hours before failing in each of these cases).
This was possibly caused by a conflict with the existing install of open-cv on the pi at this time, but the error message was unclear.

For the reference of potential future users of the system, the password to the user account on the system's Raspberry Pi is "enph".

Appendix F: Detailed description of image capture and analysis pipeline

Image Capture

Separating the video feed from the Picam into individual frames to be processed was a surprisingly non-trivial issue. For training an NN it is critical that frames are captured at a constant rate so that the agent's inputs always have consistent impact. If insufficient computational resources are available for capturing images, frames will be dropped. The Picam python library has sample code for using multithreading for rapid image capture.[6]

Even using this code with 8 threads, we were only able to capture 60 frames per second (fps) at a resolution of 400×400 in color or 600×600 in grayscale before losing frames. Having additional code running on the Pi further lowers these numbers. This relatively low resolution reduces the accuracy with which the ball location and velocity can be known, which is detrimental to precise control.

One way to avoid this issue is to use the "raspivid" command to forward the video onto a port, and transfer it to a device with more computational resources available.

Image straightening

Given that the platform tilts underneath the camera, the camera's perspective must be transformed to accurately know the position of objects on the platform. In openCV, the image processing library we used, this can be achieved either by knowing the angle and dimensions of the platform, or by knowing the coordinates in the image of four known points on the platform.

Our implementation uses HSV recognition to identify the centroids of four pink stickers at known locations on the platform. Knowing the bounding boxes of the sticker's movement in the camera image allowed us to distinguish the stickers. After straightening, we applied a manually-created mask to black out the parts of the image outside of the maze.

This algorithm was robust to both dim artificial light as well as bright daylight without recalibration. However, the ball occasionally got stuck behind the stickers, and the usage of HSV forced us to use colour images which are slower to process than grayscale.

One way to address both of these issues is to use ArUco tags on the platform, but outside of the area accessible by the ball.

Alternatively, to reduce the computational load further, a hardware accelerometer could provide the platform angle.

---

[6] https://picamera.readthedocs.io/en/release-1.13/recipes2.html#rapid-capture-and-streaming

Ball detection

First, to identify the ball, we applied image subtraction between consecutive frames of the straightened image. Since our image straightening can vary by a few pixels, I applied an eroded version of the mask used in image straightening to remove data from the edges of the platform. The accuracy of this technique is limited, and so it is not recommended for any ML applications. However, it reliably provided an approximate ball location which was adequate for demonstrating the functionality of the hardware.

If image subtraction fails, it is still necessary to check if the ball is stationary. To do this, we exploit the fact that the reflection of the steel marble is very bright, but unlike other bright objects such as screws, the ball casts a dark shadow. Specifically, in a grayscale image, we apply a mask for brightness and a separate mask for darkness, dilate both and apply a bitwise-and operation.
Nominal object tracking algorithms exist such as MOSSE and GOTURN which may be more accurate or quicker but are currently untested on this platform.

If the ball is not found in the image for several consecutive frames, we conclude that it has fallen off the platform.
Having a known ball location, a control algorithm can be applied to determine a desired platform angle. We used simple two-dimensional PID control.

Control

Once the camera feed has been processed and PID control has provided a target platform orientation, the orientation angles are input to the inverse kinematics script located on the Pi, which then outputs servo angles corresponding to this orientation. These are then sent to our Arduino control board via a Serial collection, alongside a “T/F” character flag to indicate whether the marble is still present in the maze.
If the flag is “F” then the angles are sent to their respective motors to effect the desired change in orientation, if it is “T” then the recovery subroutine is initiated and no further platform motion occurs until the marble has been successfully returned by the recovery mechanism and found by the ball detection algorithm.

Due to the compromises made in the algorithms used, the control loop takes about 0.08 seconds to run, so we have limited the camera framerate to 12fps.

Appendix G: Control and recovery demo
https://youtu.be/WI97a7RUNUk

In this short video, the controller is attempting to send the ball along a circular path, but drops it in the hole, demonstrating the recovery mechanism.
This also shows an overview of the platform hardware.

Appendix H: description of possible ML pipelines

- Replace PID control with a reinforcement learning (RL) agent taking ball position and velocity as state space.
- Use a convolutional neural network (CNN) to recognize holes and walls. Use this to find a path through the maze, either classically or with another NN. Finally, steer the ball to target locations either with PID or yet another NN.
- Use a CNN trained with RL to map directly from the straightened platform image to platform angles.
- Develop a manual controller and train via imitation learning.