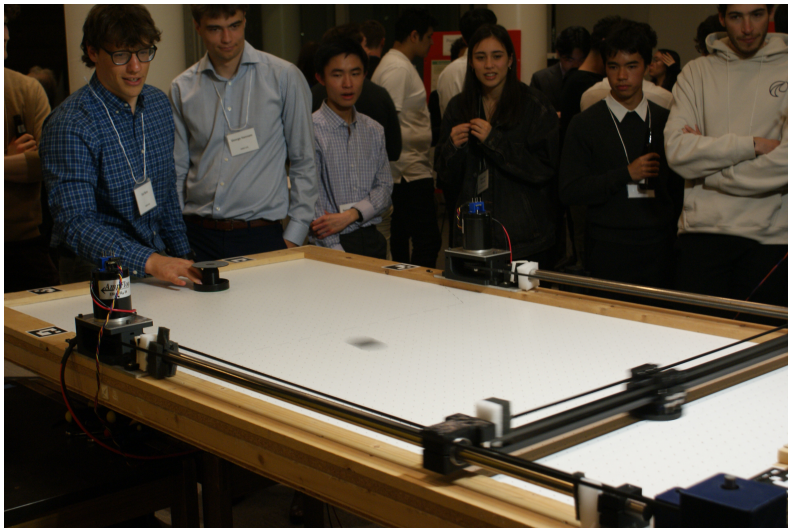
Puck Pilot

Robotic Air Hockey Table



Hudson Nock
Hunter Ma
Ian Hartley
Mauro Ferraz
Peter Voznyuk

Project Sponsor
Engineering Physics Project Lab

Project 2509
ENPH 459
The University of British Columbia

April 8, 2025

Executive Summary

The PuckPilot Robotic Air Hockey Table is a multiyear project, sponsored by the Engineering Physics Project Lab, with the end goal of creating an AI powered air hockey table capable of beating the best humans in air hockey. We inherited the mechanical and electrical foundation for the table from teams who took on the project in previous years. Our team's major goals for the two-year project are to overhaul the computer vision tracking system, create a new simulated environment for our AI agents to train in, and ultimately integrate the agent with our physical environment to run on the real table.

In the first year of the project, we have successfully implemented a computer vision system with our new high-speed camera, created a simulated environment based on experimentally determined parameters, developed a new reinforcement learning (RL) model and control algorithm, and made a few mechanical improvements to the mallet-gantry system. Our new computer vision system has allowed for very accurate position and velocity measurements, due to the improved calibration methodology and our high frame rate camera. The simulated environment and new RL model we developed have resulted in high-level play within the simulation, however, this did not translate all that well into our real system. Finally, the mechanical improvements we made to the table have increased the system's robustness and decreased the noise produced during operation.

Our recommendations for the future of the project include improving the model of the real system through a new system identification procedure and better determination of puck dynamics. Furthermore, with this model, another RL model should be trained to achieve a successful sim-to-real transfer. Finally, to bring the system to a complete state, future iterations of this project should seek to acquire and mount the PuckPilot system to a professional air hockey table.

Table of Contents

Table of Figures

1 Introduction

The AI Air Hockey Table is sponsored by the UBC Engineering Physics Project Lab with the objective of developing a robot that can beat human opponents at air hockey. This is the fifth year of development, with multiple previous capstone teams having made significant progress towards achieving the project's goal. Our team has taken on the AI Air Hockey Table—which will be referred to "PuckPilot" in this report to distinguish it from previous iterations—as a two-year capstone project. This year, we focused on understanding the fundamentals of the project. We were not strictly limited to building on the frameworks that previous teams had chosen—one of our main achievements this year was designing our own environment for reinforcement learning (RL), including a custom physics simulation. We also made significant improvements to the computer vision system: we installed and integrated a new, high-speed camera; we designed an algorithm to track the puck without requiring substantial modifications; and we were able to track the human opponent's mallet, allowing us to incorporate it into the RL model's state space.

At a higher-level, the purpose of this project is to develop a "full-stack" AI system that outperforms humans at a novel physical task—namely, beating them at air hockey. Our ultimate goal for the project is to challenge and beat the air hockey world champion. To achieve this, we must build a reliable, robust, and well-documented system to allow efficient development next year and beyond.

2 Discussion

The air hockey system can be broken down into three main subsystems - the electromechanical system, the computer vision system, and the reinforcement learning system.

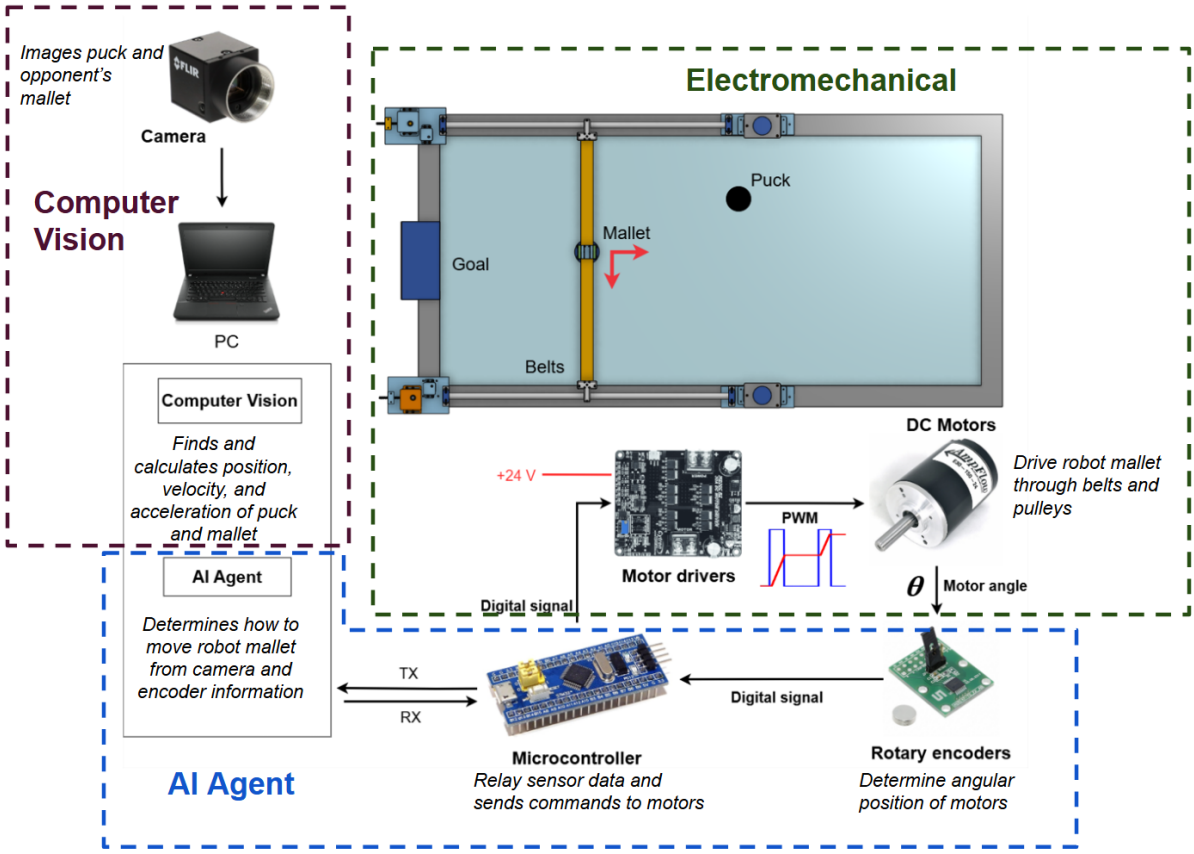


Figure 1: System level diagram showing the system loop and approximate boundaries between subsystems.

This report will walk through each system involved in the system loop. In addition, the simulation framework used as a training environment for the RL agent will be discussed.

2.1 Electromechanical

The electromechanical system consists of all physical hardware and electronics, excluding the camera. The electrical system is shown in Fig.2. This schematic is representative of the actual physical layout of the electronics, with the capacitor installed inside the electrical cabinet and all other electronic components mounted on top of it to minimize the risk of accidentally shorting the capacitor. See Appendix A for a full

description of the hardware.

These components and the overall system architecture were inherited from previous teams. A custom PCB was also designed to house the electronics, and featured an automatic safety interlock consisting of a series of relays. These relays would open and close only when certain checks were passed in firmware—for example, the high-side power relay would remain open until the capacitor was fully charged, and thus safe to connect in parallel with the motors[1]. An additional BluePill was used for this safety-critical firmware. Such a system is attractive because it reduces the risk of human error in operating the table, which is especially important for project hand-off. However, we decided to revert to manual control of system power this year due to difficulties debugging and modifying a PCB. This allowed us to study the electromechanical system in-depth and ensured that we could have the table up-and-running with less risk of malfunction.

Throughout development, we kept safety at the forefront of our design process. The greatest safety hazard is the capacitor, which can store approximately 47kJ when fully charged. An accidental short would be catastrophic and could cause serious injury. We were not confident that the few safety measures in place would be sufficient to mitigate this risk. For instance, a strip fuse with a 1200A breaking capacity was previously installed in-series with the capacitor; however, the capacitor's manufacturer lists a peak current of over 2000A, and an estimate of dead-short current ($I = V/R_{\text{ESR}} = 24\text{V}/5\text{m}\Omega = 4800\text{A}$) exceeds that. Although we would not expect to see sustained arcing (the primary concern of exceeding breaking capacity), we would still be taking on an unacceptable risk by operating outside the specified limits. This was one factor in our decision to remove the capacitor in the short-term and design our own electrical circuit from the ground-up.

We also took measures to mitigate mechanical hazards. Crowd control barriers were placed around the cabinet and the side of the table where the gantry is mounted to prevent pinching. Another mechanical hazard is the mallet carriage colliding with the table walls, or the crossbeam with the ends of the gantry rails, which could damage the gantry and injure bystanders. We placed conservative limits on the mallet's travel area in firmware and lowered the voltage (*i.e.* the PWM duty cycle) of the motors from the power supply voltage of 24V to 15V. Finally, we placed foam on the ends of the gantry rails to cushion impact.

[1] See Shular et al. (2024) for further information about this interlock system.

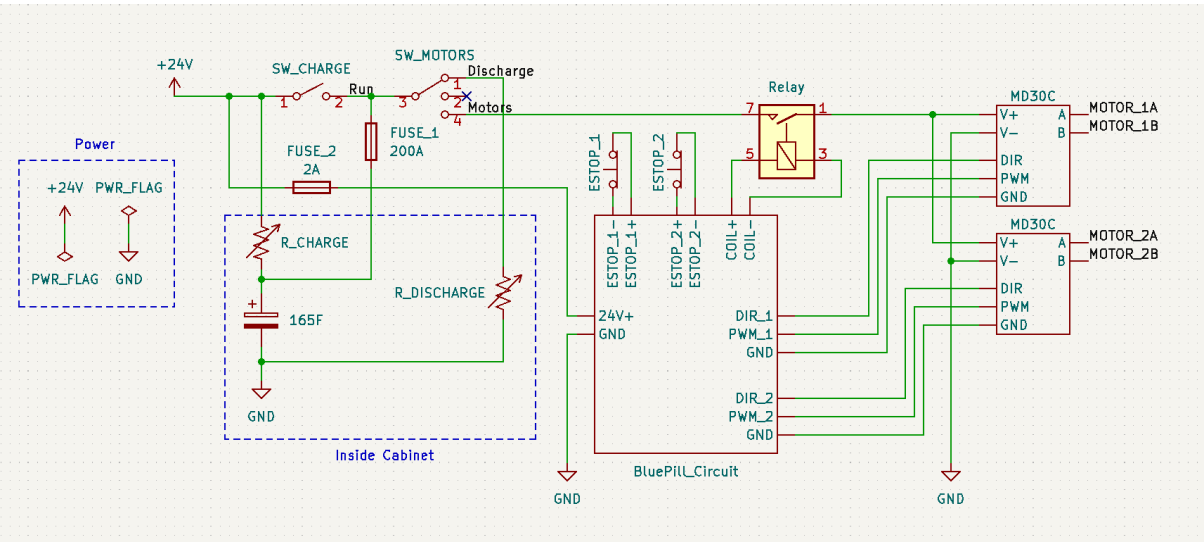


Figure 2: Schematic of power electronics. See Fig. 10 for schematic of microcontroller circuit.

2.2 Computer Vision

2.2.1 Lighting

Previous teams could not reliably track the puck without modifications that altered the kinematics of the puck—for instance, placing an LED on top of a 3D-printed puck. We resolved this by using retroreflective material, containing microscopic corner reflectors that reflect light directly back to its source regardless of incident angle, and a high-intensity LED lamp directly adjacent to the camera (Figure 4). By adhering a thin layer of retroreflective fabric to the puck, we achieved a high contrast ratio under illumination without significantly altering the puck's mass. High contrast is desirable because it allows us to set the camera's exposure as low as 100µs, which reduces motion blur and makes precise position estimates possible during high-speed gameplay (Figure 5).

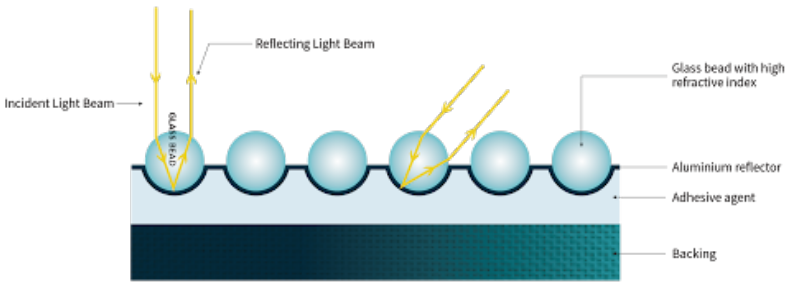


Figure 3: Diagram illustrating the principle of retroreflection, where incident light is returned directly to its source regardless of angle.

Figure 4: Camera and illumination mounting configuration showing the coaxial arrangement of the camera and LED array.

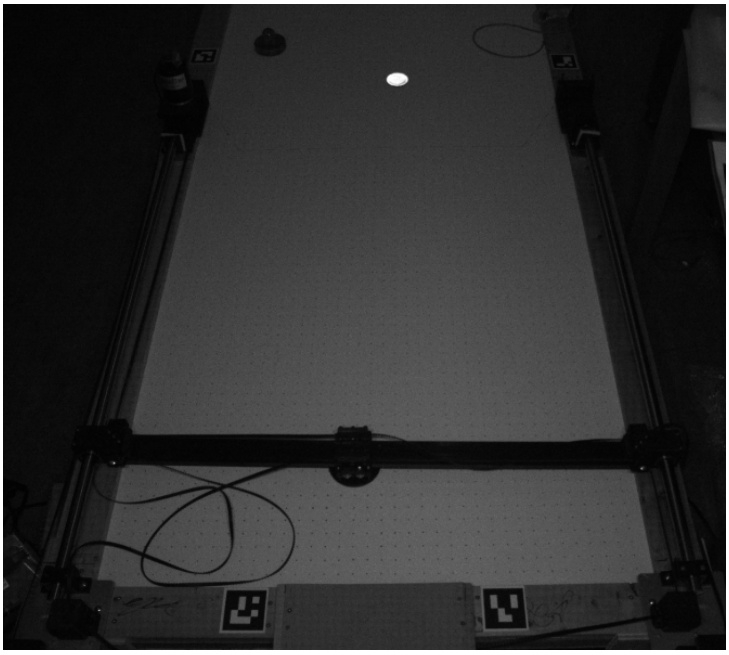


Figure 5: Camera view showing the high contrast between the retroreflective puck and playing surface, captured with 100 μs exposure.

2.2.2 Determining Puck Location

The goal for this section is to understand how the camera relates to the 3D world, and find a relationship between the pixels on the image and the points on the table.

Camera Intrinsic Calibration We performed standard intrinsic calibration using OpenCV, capturing multiple views of a checkerboard pattern displayed on a television screen. This process yielded the camera's intrinsic matrix and distortion coefficients, establishing the relationship between 3D points and their 2D projections in the image plane.

For more details, see Appendix B.1.

Extrinsic Calibration Challenges Traditional extrinsic calibration relies on precisely measuring the 3D positions of ArUco markers placed in the environment. However, because the air hockey table surface exhibited non-planar deviations, we:

- Could not measure the 3D positions of the ArUco markers
- Could not use a Z-coordinate constraint to project from the pixel to the point on the table (Figure 6)

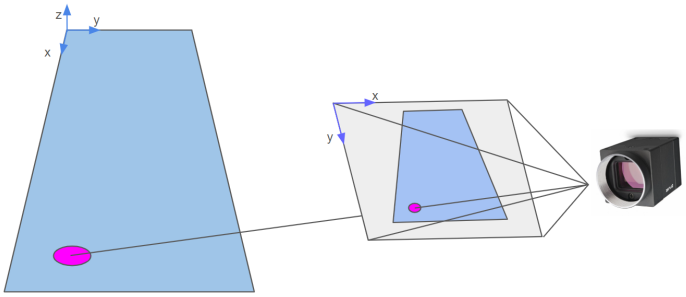


Figure 6: There are multiple solutions when projecting a pixel to 3D space, so we need another constraint to get a unique solution, in this case the constraint would be the height of the table at that point.

Multi-View Optimization Approach To overcome these challenges, we developed a novel calibration procedure to solve for the 3D positions of the ArUco markers and the table surface height, modeled as a second-order 2D polynomial. Refer to Appendix B.4 for details.

Inverse Projection Using the calibrated ArUco marker positions and camera intrinsics, we determine the camera's pose (i.e., rotation and translation) using Perspective-n-Point algorithm. With these parameters established, we can inverse-project pixel coordinates to physical table coordinates, accounting for the puck height above the table surface. This calibration achieves high precision, with a mean positional error of approximately 1 mm and maximum deviations of 2.5 mm (Figure 7). For more details, see Appendix B.2.2.

Puck Occlusion A significant challenge in our system is puck occlusion by the mallet gantry. When partially occluded, the visible puck contour no longer provides a reliable centroid measurement. We address this challenge through a robust contour analysis approach.

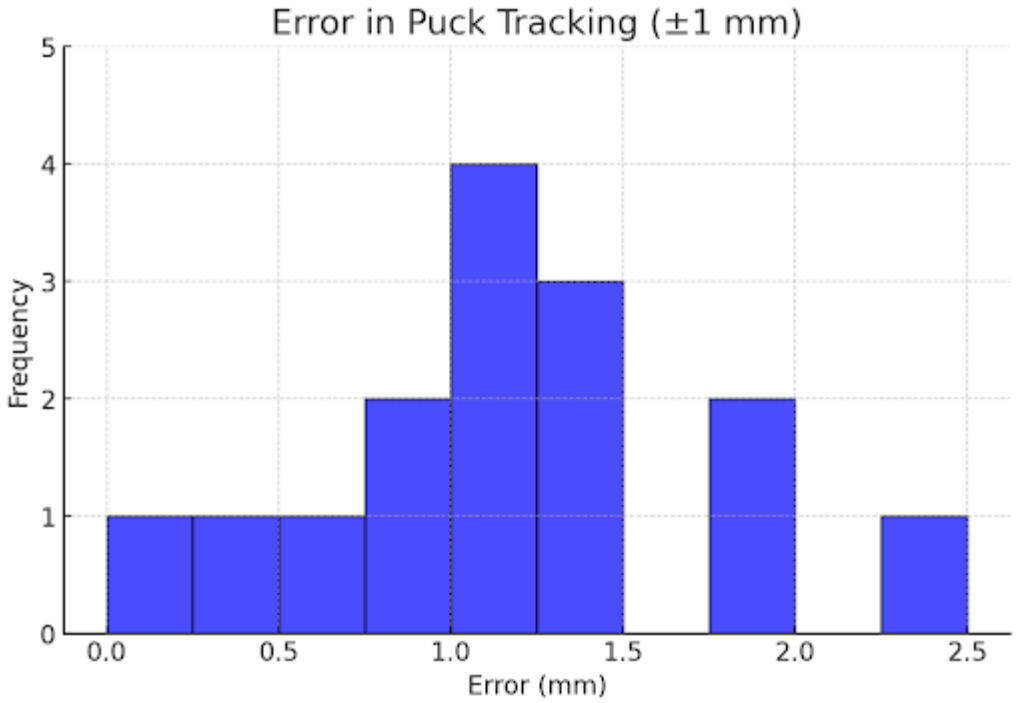


Figure 7: Spatial distribution of inverse projection errors across the table surface. Note the ± 1 mm error due to measurement uncertainty.

First, we project the entire puck contour to world coordinates, analyzing its shape in the (x, y) plane. In unoccluded states, this projection approximates a circle regardless of perspective distortion in the image. When occlusion occurs, either the top or bottom portion of the puck remains visible.

Using this contour, we can generate two candidate puck centers (procedure is outlined in Appendix C).

Next, we determine the accuracy of each candidate solution and select the candidate with the highest conformity, or default to the candidate nearest to the previous position when confidence is low.

This approach maintains tracking accuracy as long as the puck is partially visible. Figure 8 shows the circle of best fit and its moment using a noisy and bottom-occluded contour. For more details on puck occlusion, see Appendix C.

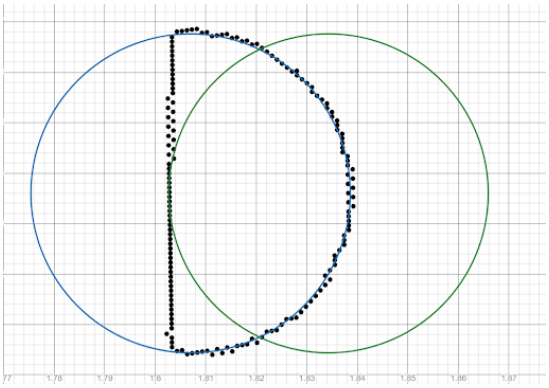


Figure 8: Finding puck centroid from a half covered puck. Black dots: contour of puck. blue and green circle: Puck candidates, in this case the blue candidate matches the data better

2.2.3 Opponent Mallet Tracking

To enable comprehensive gameplay analysis, our system must also track the opponent's mallet. We developed a specialized attachment for opponent mallets that maintains normal gameplay while enabling robust visual tracking. Refer to Appendix D for more detail.

2.3 Simulation

2.3.1 Mallet Movement

System Dynamics and Voltage Transformation We are simulating the mallet movement using the feedforward model given to us by previous teams. This model relates the voltages applied to the two motors (V_1, V_2) to the resulting angular motion (T_1, T_2) and involves up to third-order time derivatives of the angles:

$$\begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 & b_1 & b_2 & b_3 \\ b_1 & b_2 & b_3 & a_1 & a_2 & a_3 \end{bmatrix} \begin{bmatrix} \dddot{T}_1 \\ \ddot{T}_1 \\ \dot{T}_1 \\ \dddot{T}_2 \\ \ddot{T}_2 \\ \dot{T}_2 \end{bmatrix} \tag{1}$$

where a_i, b_i are constants characterizing the system dynamics. The mallet's Cartesian coordinates (x, y) are related to the motor angles via $y = R(-T_1 - T_2)/2$ and $x = R(T_1 - T_2)/2$, where R is the pulley radius. From this, we can transform the control problem into Cartesian space. By defining effective voltage commands V_x and V_y as linear combinations of the motor voltages ($V_x = V_1 - V_2$, $V_y = -V_1 - V_2$), we decouple

the dynamics along the x and y axes (see Appendix E.1 for derivation):

$$V_y = \frac{2}{R}\left[(a_1 + b_1)\dddot{y} + (a_2 + b_2)\ddot{y} + (a_3 + b_3)\dot{y}\right] \tag{2}$$

$$V_x = \frac{2}{R}\left[(a_1 - b_1)\dddot{x} + (a_2 - b_2)\ddot{x} + (a_3 - b_3)\dot{x}\right] \tag{3}$$

Voltage Profile Because it would be computationally inefficient to solve (2)-(3) for general V_y and V_x, we constrain the simulation to only allow voltages of a specific form using the Heaviside step function $u(t)$:

$$V_x(t) = \hat{V}_x\left[u(t) - 2u(t - t_{x1}) + u(t - t_{x2})\right] \tag{4}$$

$$V_y(t) = \hat{V}_y\left[u(t) - 2u(t - t_{y1}) + u(t - t_{y2})\right] \tag{5}$$

Here, $\hat{V}_x$ and $\hat{V}_y$ represent the constant voltage applied during the acceleration/deceleration phases. Given arbitrary initial conditions, $x(0), \dot{x}(0), \ddot{x}(0)$ and $y(0), \dot{y}(0), \ddot{y}(0)$, we can construct a closed form expression for the resulting path of the mallet given $\hat{V}_x, \hat{V}_y$, t_{x1}, t_{x2}, t_{y1}, and t_{y2} whilst still allowing a large variety of possible movements of the mallet (see Appendix E.2).

Physical and Safety Constraints The motors operate within a maximum voltage $|V_1|, |V_2| \leq V_{max} = 24V$. From the definitions $V_1 = (V_x - V_y)/2$ and $V_2 = (-V_x - V_y)/2$, a sufficient condition to respect these limits is $|\hat{V}_x| + |\hat{V}_y| \leq 2V_{max}$. Therefore, we can guarantee the path is possible (assuming the feedforward is accurate).

2.3.2 Puck Movement

Accurately predicting the puck's trajectory is essential for the RL agent's planning. This section details the physics model used for the puck and the methods for efficiently calculating collision events.

Puck Kinematics Model The puck's motion between collisions is primarily influenced by friction with the table surface and air resistance. We model these forces as opposing the direction of velocity $\mathbf{v}$. Assuming a constant friction force magnitude f and air resistance proportional to speed squared ($|\mathbf{v}|^2 = \dot{s}^2$, where s is the distance travelled along the path), the equation of motion for the puck's speed $\dot{s}$ is:

$$m\ddot{s} = -f - B(\dot{s})^2 \tag{6}$$

Here, m is the puck mass, B is the air resistance coefficient, $\dot{s}$ is the speed, and $\ddot{s}$ is the tangential acceleration. The parameters m, f, B, along with the coefficient of restitution e_p for bounces, were determined experimentally (as described in Appendix F).

Equation (6) can be solved analytically. Given an initial speed $v_0 = \dot{s}(0)$ at $t = 0$, the distance s travelled at time t is:

$$s(t) = \frac{m}{B} \ln \left(\cos \left(\frac{\sqrt{Bf}}{m} (Cm + t) \right) \right) + D \tag{7}$$

where the integration constants C and D depends on the initial speed v_0:

$$C = -\frac{1}{\sqrt{Bf}} \arctan \left(v_0 \sqrt{\frac{B}{f}} \right) \tag{8}$$

$$D = -\frac{m}{B} \ln \left(\cos \left(C \sqrt{Bf} \right) \right) \tag{9}$$

This solution is valid for $t \in [0, t_{stop}]$, where $t_{stop} = -Cm$ is the time at which the puck stops due to friction ($\dot{s}(t_{stop}) = 0$). This analytical form allows direct calculation of the distance travelled without iterative time-stepping.

Collision Prediction and Classification To simulate efficiently, we predict the time t_{coll} and type of the next collision event. This involves determining whether the puck will first hit a side wall, a goal corner, or enter the goal.

To do this, we cast rays on either side of the puck in its direction of travel. We use the rays to classify the type of collision as seen in Figure 9. More detail is provided in Appendix G.1

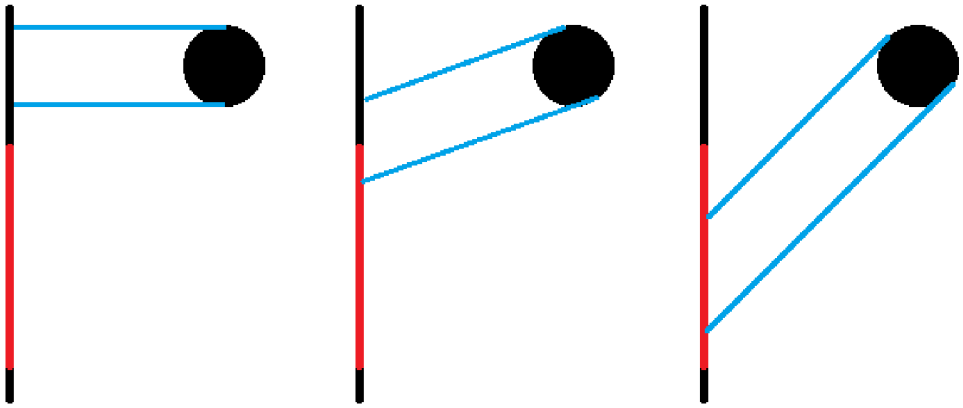


Figure 9: Determining Puck Collisions with ray casts
Black circle: puck. Black line: Wall. Red line: Goal. Blue line: projected ray.
Left: both rays intersect a wall therefore this is going to be a wall bounce
Middle: only way ray enters the goal, so this is a corner bounce
Right: Both rays enter the goal, so the puck does not experience a collision

Collision Resolution At the moment of collision t_{coll}, the puck's state is updated. Its position is advanced to the collision point. The velocity vector $\mathbf{v}$ is reflected based on

the surface normal $\mathbf{n}$ at the point of contact, incorporating the coefficient of restitution e_p:

$$\mathbf{v}_{new} = -e_p(\mathbf{v}_{old} \cdot \mathbf{n})\mathbf{n} + e_p(\mathbf{v}_{old} \cdot \mathbf{p})\mathbf{p} \tag{10}$$

The normal $\mathbf{n}$ is perpendicular to the wall and parallel $\mathbf{p}$ is parallel to the wall. For corner collisions, it represents the effective normal at the point of contact. With the updated position and velocity, the simulation proceeds recursively until it steps forward the desired amount.

2.3.3 Puck-Mallet Collisions

Accurately determining the moment of collision between the puck and mallet is crucial. Standard simulation with a small fixed Δt can be computationally prohibitive. This section describes an efficient algorithm that uses an adaptive time step based on a lower bound on the time until the next potential collision.

Iterative Lower Bound Approach The core strategy is to avoid small, fixed time steps. Instead, for a given simulation interval (e.g., until the next agent decision), we calculate a safe lower bound t_{lb} on the time until a puck-mallet collision could occur. The simulation state (puck position, velocity; mallet position, velocity) is then advanced by time t_{lb}. This process is repeated until the desired simulation interval is covered. Advancing the simulation by t_{lb} (ignoring puck-mallet interactions during this interval) takes the same computation as advancing by a small Δt, yet t_{lb} is typically much larger. This significantly speeds up the simulation while guaranteeing no collision is missed. Refer to Appendix H for calculation details.

Simulation Update The overall simulation loop advances as follows:

1. Determine the maximum time T_{rem} remaining in the current simulation interval.
2. Calculate the collision lower bound time t_{lb}.
3. If t_{lb} is invalid (e.g., imaginary), set $t_{lb} = \infty$.
4. Choose the adaptive time step $T_{step} = \min(t_{lb}, T_{rem})$.
5. Advance the puck state (position, velocity) by T_{step} using Eq. (7).
6. Advance the mallet state (position, velocity) by T_{step} using its own dynamics model (from Appendix E.2).
7. Subtract T_{step} from T_{rem}.
8. If $T_{rem} > 0$, repeat from step 2 with the updated states.

This iterative process steps through time efficiently. When t_{lb} becomes very small, it signals an imminent potential collision, which is handled by assuming collision occurs at that point.

2.3.4 Vectorization and Performance

The simulation was optimized through vectorization using NumPy. Vectorization refers to the process of restructuring code to replace explicit iterative operations with operations that apply simultaneously to multiple data elements. In the context of our simulation, this technique allowed the system to process and update multiple environment instances concurrently rather than sequentially.

Our benchmarks demonstrated that the system could simulate approximately one hour of agent experience in 8 seconds of computational time, which is sufficient for training a reinforcement learning agent.

There are some unavoidable bugs in the simulation. For example, if the agent hits the puck into a corner and tries to move towards the corner this would cause the puck to glitch through the wall. Since we don't want this behavior during deployment, we train the agent to avoid this in the reinforcement learning phase.

2.4 Reinforcement Learning

Reinforcement Learning (RL) is a machine learning method in which an agent learns optimal behavior through interactions with its environment. In RL, the agent observes the current state, selects an action, and receives feedback in the form of a numerical signal called a reward, which measures the desirability of its chosen action. The agent's goal is to maximize the cumulative rewards over time, effectively learning through trial-and-error experiences. By repeatedly updating its strategy based on these interactions, the agent progressively improves its decisions and performance.

2.4.1 State Space

The state representation was carefully designed to provide the agent with comprehensive situational awareness within the air hockey environment. The state space encompasses the following key components:

- Position and velocity vectors of the puck, providing essential information about its current trajectory.

- Position and velocity vectors of the agent's mallet, enabling awareness of its own effector state.
- Position and velocity vectors of the opponent's mallet, allowing the agent to aim better and anticipate countermoves.
- Calculated future expected position of the puck based on physics simulation, which we expect to reduce the complexity of the learning task by providing predictive information that would otherwise need to be inferred by the neural network.

Additionally, we incorporated a binary defense/attack flag as part of the state representation. This flag serves as a high-level strategic directive, indicating whether the agent should prioritize defensive positioning or offensive maneuvers.

2.4.2 Action Space

As outlined in the previous section, the simulation environment requires voltage components (V_x, V_y) and timing parameters ($t_{x1}, t_{x2}, t_{y1}, t_{y2}$) to calculate the mallet's trajectory between agent steps. However, designing the action space such that the Reinforcement Learning (RL) agent directly outputs these six parameters presents significant challenges. Firstly, learning the complex relationship between the timing parameters and the resulting movement dynamics would be difficult for the agent. Secondly, direct output of these parameters could lead to unsafe actions, allowing the agent to command movements that result in high-speed collisions with the table boundaries.

To address these issues and create a more learnable and inherently safe action space, we adopt an alternative approach. The RL agent instead outputs four values: the desired final position $\mathbf{x_f} = (x_f, y_f)$ and the magnitudes of the voltage components, $|V_x|$ and $|V_y|$. This formulation is advantageous because the target position $\mathbf{x_f}$ directly relates to the task goal, and the voltage magnitudes $|V_x|, |V_y|$ correspond to the speed of the intended movement, making the learning task potentially easier.

An algorithmic layer then processes the agent's output $(|V_x|, |V_y|, x_f, y_f)$ along with the current state (including the initial position, velocity, and acceleration) to determine the appropriate signs for V_x and V_y, and to calculate the necessary timing parameters $t_{x1}, t_{x2}, t_{y1}, t_{y2}$. This layer also enforces safety constraints. For more information read Appendix I.

This action space design offers significant advantages. By having the agent output target positions and velocity magnitudes – concepts closely related to the state space

and task goals – we facilitate learning. More importantly, the algorithmic layer ensures that, regardless of the agent's raw output, the resulting commands sent to the simulation (and to the real robot) are dynamically feasible and prevent collisions with the table boundaries, guaranteeing safety during operation and training.

2.4.3 Reward Structure

The reward structure is designed to guide the agent towards effective air hockey play by breaking down the complex task into shorter, distinct episodes corresponding to defensive and offensive phases. This episodic approach simplifies the learning problem compared to optimizing over a single, long trajectory covering multiple back-and-forth plays. Notably, the termination of an RL episode (signaling the end of a specific task phase for the agent) does not necessarily reset the physical simulation state, allowing for continuous play dynamics while providing focused learning signals.

Additionally, we reward the model for scoring based on the projected path of the puck, independent of if the shot is saved. This happens because eventually the models get so good at defending that no new goals are scored, causing an absence of goal-based reward signals to train off of.

In our setup, the agent operates in one of two modes, determined by a flag: Defence or Attack.

Defence Mode An episode in Defence Mode begins when the opponent gains possession of the puck (e.g., after the agent loses it or after a reset). The agent's objective is not necessarily to regain immediate control, but to execute a safe clear, inspired by professional play where players often direct the puck towards the side walls away from the opponent.

- **Success:** The agent is rewarded (+2) and the episode terminates successfully if it manages to bounce the puck vertically to demonstrate control.
- **Failure (Goal Conceded):** If the opponent scores (puck enters the agent's goal), the agent receives a penalty (-1), the episode terminates, and the simulation is reset.
- **Failure (Loses the Puck):** If the agent hits the puck such that it crosses the midline towards the opponent's side, the episode terminates. A small reward or penalty is given based on whether the puck's projected trajectory threatens the opponent's goal.

Specific conditions and reward values for these outcomes are detailed in Appendix J.

Attack Mode An Attack Mode episode starts after a successful defensive clear. The agent's objective is now to score a goal.

- **Outcome (Shot Attempt):** If the puck crosses the midline and is likely to enter the opponent's goal (based on the puck's velocity and projected trajectory), the agent receives a positive reward proportional to the squared velocity, or zero reward otherwise. The episode terminates, and a new Defence Mode episode begins.
- **Failure (Own Goal):** If the puck enters the agent's own goal during an attack attempt, the agent is penalized (-1), the episode terminates, and the simulation resets before starting a new Defence episode.

Details are provided in Appendix J.

Continuous Penalties In addition to the terminal rewards/penalties associated with episode outcomes, the agent receives small negative rewards at each time step to encourage efficient behaviour:

- **Movement Cost:** A penalty encouraging it to stay still
- **Energy Cost:** A penalty encouraging it to move slowly.

These costs incentivize the agent to prefer shorter, lower-intensity movements unless a high-speed action (like a shot) is necessary to achieve a primary objective, promoting controlled play.

This overall structure, with its short episodes and targeted rewards/penalties, aims to create a tractable learning environment while guiding the agent towards strategically sound and efficient air hockey maneuvers.

2.4.4 Training Architecture

The reinforcement learning agent was trained using the Proximal Policy Optimization (PPO) algorithm, leveraging the `torchrl` framework for implementation (Schulman et al. (2017)). PPO is a policy gradient method known for its stability and sample efficiency, making it suitable for complex control tasks like air hockey.

The agent's core consists of two main neural networks: an actor (policy network) and a critic (value network). Both networks share a similar Multi-Layer Perceptron (MLP) architecture composed of several linear layers with ReLU activation functions.

The actor network outputs the parameters (mean and standard deviation) for the action distribution, while the critic network outputs a scalar estimate of the state value function.

Advantage estimation, crucial for PPO's policy updates, was performed using Generalized Advantage Estimation (GAE) (Schulman et al. (2015)). Key hyperparameters, including the PPO clipping parameter ϵ, GAE's λ, and the entropy bonus coefficient, were intentionally set to low values. While this potentially slowed down convergence, empirical results indicated that these settings led to more stable training and better performance for this task.

Training was executed in a highly parallelized manner, utilizing 2048 environments concurrently to gather experience efficiently. Each environment consisted of two agents playing against each other. Data was stored in a large replay buffer, and training updates were performed periodically by sampling from this buffer. To enhance sample efficiency, data augmentation through symmetric flipping along the table's central x-axis was employed, effectively doubling the number of usable transitions per collected sample. Additionally, small random impulses were applied to the puck during simulation to prevent it from getting stuck in static states and ensure richer interaction dynamics.

Further details on the network architectures, the specific hyperparameter values, and the training loop configuration are provided in Appendix K.

3 Conclusion

The primary goals of this project were to complete the system control loop and establish resilient, accurate, and durable systems for our AI air hockey platform. We successfully achieved these objectives through several key accomplishments:

Key Results

- Achieved exceptional puck tracking accuracy with a maximum error of 2.5 mm and an average error of just 1 mm, which is sufficiently precise for our application without requiring further refinement.
- Developed a vectorized simulation environment and reinforcement learning (RL) training pipeline that accurately represents our system identification while maintaining computational efficiency.

- Completed a robust electrical system with secure mounting and wiring of all components, minimizing the need for major electrical rework in future iterations.

Building upon these foundational achievements, we were able to:

- Train an RL model capable of professional-level play within the simulation environment.
- Successfully integrate all system components to run the complete control loop and evaluate real-world performance.
- Identify limitations in the simulation-to-reality transfer, primarily due to time constraints that prevented proper system identification and accurate modeling of system delays before retraining.

Overall, this first year of development has yielded significant progress toward our objective of creating a professional-level AI air hockey system.

4 Recommendations

Based on our findings, we recommend the following steps for future development:

- While the current setup appears capable of achieving professional-level play, it does not conform to official air hockey regulations. Future work should focus on transferring the gantry system to a regulation-compliant air hockey table.
- Reattach the super capacitor to the system to prevent voltage drops during play.
- Implement automated system identification at various voltage levels using gradient descent optimization between expected and recorded mallet trajectories.
- Establish statistical error bounds from systematic system identification.
- Precisely measure the total delay in the system loop with associated error margins.
- Enhance the RL model to operate directly on sensor data (i.e., sequences of position measurements) and extend training to longer episode durations.
- Implement comprehensive domain randomization in the simulation environment based on empirical noise profiles and system characteristics.
- Develop a more precise understanding of error distributions in puck dynamics, including improved modeling of collisions with mallet and table boundaries.

5 Deliverables

1. The AI Air Hockey table and all peripherals, including:
   (a) The electrical cabinet containing the capacitor and charge/discharge resistors;
   (b) FLIR Blackfly S camera, tripod, and lamp; and
   (c) Game items (pucks and mallets, with modifications) and spare parts
2. GitHub organization containing project files and code:
   (a) Reinforcement learning simulation and training code,
   (b) Microcontroller firmware,
   (c) Host PC scripts for system loop,
   (d) Puck dynamics analysis and system identification scripts, and
   (e) KiCad schematics

6 References

Jason-s. (2017). *Comment on "enh: Implement at least one good robust scalar root-finding algorithm (brent's? chandrupatla's?) in a vectorized approach": Comment on github issue* [March 30]. GitHub. https://github.com/scipy/scipy/issues/7242#issuecomment-290548427

Schulman, J., Moritz, P., Levine, S., Jordan, M. I., & Abbeel, P. (2015). High-dimensional continuous control using generalized advantage estimation [arXiv preprint arXiv:1506.02438]. https://arxiv.org/abs/1506.02438

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms [arXiv preprint arXiv:1707.06347]. https://arxiv.org/abs/1707.06347

Shular, B., Xu, G., Thurston, J., & Dolter, N. (2024). Robotic ai air hockey: Hardware & controller design [Contact the ENPH Project Lab for access].

7 Appendices

A Electromechanical

A.1 Overview of Hardware

1. A gantry with a carriage mounted on a crossbeam, which is driven by a series of belts and pulleys in a CoreXY configuration (commonly used in 3D printers);
2. AmpFlow brushed DC motors rated for 24V with a peak power output of 1.0HP;
3. Cytron MD30C motor drivers rated for 30A and 80A of continuous and peak current, respectively;
4. AMS-Osram AS5147 rotary encoders that use Hall effect sensors to measure the rotation of the motor shafts;
5. Altran Magnetics ALEV200 high voltage DC contactor, used to cut power to the motor drivers;
6. A 165F automotive capacitor that absorbs surge current during motor braking and suppresses voltage drops caused by sudden acceleration;
7. An STM32F103C8 ”BluePill” microcontroller that processes encoder readings, sends PWM signals to the motor drivers, and communicates with the host PC.

B Camera Calibration and Coordinate Transformation Details

B.1 Camera Intrinsic Calibration

The camera's intrinsic parameters were determined using the standard pinhole camera model. Multiple images of a checkerboard pattern (9×6 corners) displayed on a television screen were captured from various angles. Using OpenCV's `calibrateCamera` function, we obtained:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{11}$$

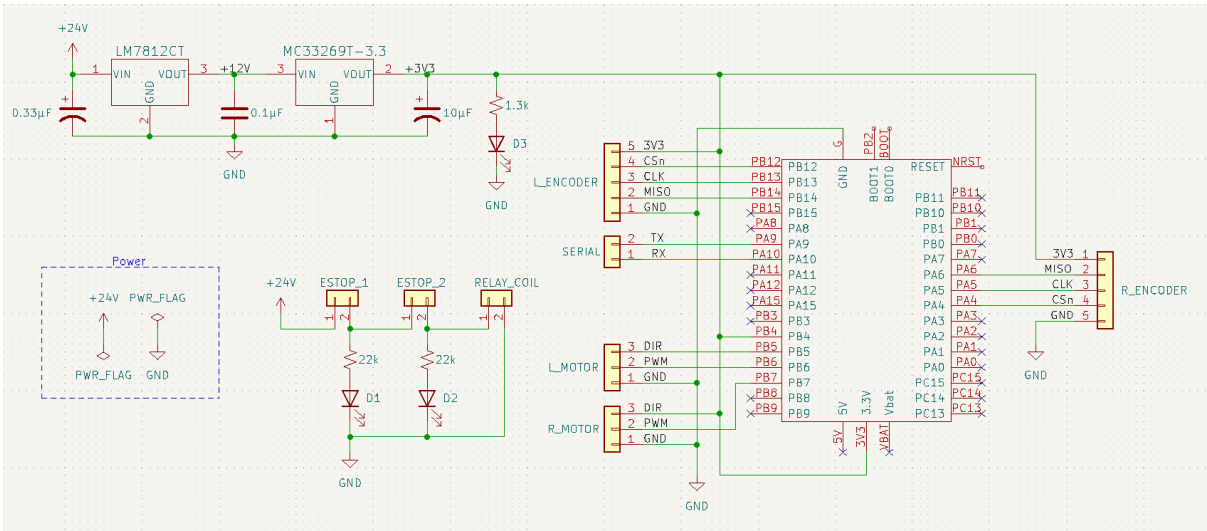


Figure 10: Schematic of the circuit board housing the BluePill microcontroller. The +24V PSU power rail is stepped-down to +3.3V for the microcontroller. LEDs D1 and D2 are powered when the E-stops are closed (i.e. not pressed); LED D3 indicates that the +3.3V rail is powered. The LED resistors were chosen to limit current to 100mA.

Where f_x and f_y represent the focal lengths, and (c_x, c_y) defines the principal point. Additionally, we obtained the distortion coefficients $(k_1, k_2, p_1, p_2, k_3)$ accounting for radial and tangential distortion effects.

B.2 Coordinate transformation using Intrinsic and Extrinsic Parameters

For this section we assume we are able to find the intrinsic and extrinsic parameters of the camera.

Converting a 2D pixel coordinate $\mathbf{p} = (u, v)$ to its corresponding physical location on the table surface $\mathbf{P} = (X, Y, Z)$ requires solving the camera pose and inverse projection problem, complicated by the non-planar table surface.

Note that the world coordinate system here has the origin at the top left corner of the table as seen by the camera. The x axis then points down and y axis points right both going alongside the table , again, as seen by the camera.

B.2.1 Camera Pose

Camera pose calculation refers to the process of determining the position and orientation of the camera with respect to a fixed world coordinate system. The extrinsic parameters are represented by a rigid body transformation that maps points from the

world coordinate frame to the camera coordinate frame. This transformation is described by a 3×4 matrix composed of a rotation matrix $\mathbf{R} \in \mathbb{R}^{3\times3}$ and a translation vector $\mathbf{t} \in \mathbb{R}^{3\times1}$:

$$\mathbf{T}_{\text{cam}\leftarrow\text{world}} = \begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix} \in \mathbb{R}^{3\times4} \tag{12}$$

Once the camera pose is known, we can project any 3D point in world coordinates onto the 2D image plane. This requires both the extrinsic parameters (to transform to the camera frame) and the intrinsic parameters (to perform the projection). The complete mapping from a 3D point in world coordinates to a 2D image point is given by:

$$\mathbf{p}_{\text{cam}} = \begin{bmatrix}x\\y\\z\end{bmatrix} = \mathbf{K} \cdot \mathbf{T}_{\text{cam}\leftarrow\text{world}} \cdot \mathbf{P}_{\text{world}} \tag{13}$$

where $\mathbf{P}_{\text{world}} = \begin{bmatrix}X & Y & Z & 1\end{bmatrix}^\top$ denotes the homogeneous world coordinate by simply adding an 1 in the last row. To recover the actual 2D pixel coordinates (u, v), we perform perspective division:

$$(u, v) = \left(\frac{x}{z}, \frac{y}{z}\right) \tag{14}$$

This process allows us to accurately project known 3D positions onto their corresponding 2D positions in the image frame.

B.2.2 Inverse Projection Algorithm

We implemented an optimization-based inverse projection method:

Input: Pixel coordinates (u, v) to inverse project and height of object h
Output: Table coordinates (X, Y, Z)
Initialize guess for table point (X_0, Y_0)
repeat
    Compute $Z = Z_{table}(X, Y) + h$ using polynomial height model
    Create 3D point $\mathbf{P} = (X, Y, Z)$ in world coordinates
    Project $\mathbf{P}$ to pixel coordinates (u', v') using camera model (Intrinsics and Extrinsics)
    Calculate error: $e = \|(u, v) - (u', v')\|^2$
    Update (X, Y) using gradient descent to minimize error
until $e < \epsilon$

Although this algorithm is not being used in real time, it is possible to instead search along the projection ray to find the intersection with the table which would result in a

faster computation.

B.2.3 Real-Time Implementation

To enable real-time operation, we precompute a matrix of Z-heights at 8-pixel intervals across the image. During operation, for a detected puck position (u_{puck}, v_{puck}), we:

1. Sample the nearest Z-height from our precomputed grid
2. Use this Z-value constraint to directly compute the corresponding (X, Y) coordinates

This approach eliminates the iterative optimization during runtime, reducing computational overhead while maintaining high accuracy in coordinate transformation.

B.3 Table Surface Parameterization

The table surface height was modeled as a second-order bivariate polynomial:

$$Z_{table}(X, Y) = a_0 + a_1X + a_2Y + a_3X^2 + a_4XY + a_5Y^2 \tag{15}$$

This parameterization captures the primary curvature characteristics while remaining computationally tractable.

B.4 Multi-View Optimization Framework

The previous section B.2 rely on having correct camera extrinsic's, however as mentioned in 2.2.2, there are many problems needed to be faced to obtain this. To address this, we developed a novel calibration procedure to solve for the 3D positions of the ArUco markers and the table surface height, modeled as a second-order 2D polynomial. The procedure follows these steps:

1. Multiple reference circles were drawn at measured positions across the table surface
2. The puck was placed sequentially at each reference position, creating a correspondence dataset
3. Data was collected from three distinct camera positions to resolve geometric ambiguity

We formulated an optimization problem with two parameterizations:

- The 3D positions of ArUco markers

- The table surface height, modeled as a second-order 2D polynomial

The optimization minimizes the error between the projected physical locations and measured reference positions. Using multiple camera viewpoints resolves the inherent ambiguity that would occur with a single view, where ArUco markers could be shifted along the projection ray without affecting the projection. While theoretically two views would suffice, three positions provided enhanced optimization stability.

The combined optimization problem encompasses both table surface parameters and ArUco marker positions. For n reference points viewed from m camera positions, we minimize:

$$E(\theta_{table}, \{\mathbf{M}_i\}) = \sum_{j=1}^{m} \sum_{k=1}^{n} \|\mathbf{P}_{jk}^{proj} - \mathbf{P}_{k}^{ref}\|^2 \tag{16}$$

Where:

- θ_{table} represents the table surface polynomial coefficients
- $\{\mathbf{M}_i\}$ are the ArUco marker positions
- $\mathbf{P}_{jk}^{proj}$ is the inverse-projected point for reference k from camera position j
- $\mathbf{P}_{k}^{ref}$ is the measured reference position on the table

Each call to this function requires

- Calling cv2.solvePnP to get the camera extrinsics for all m camera positions
- Calling $\mathbf{P}_{jk}^{proj}$ for all datasets and all points. Note that this is also an optimization problem.

Therefore the evaluation of this function is slow, and the total time to minimize this error takes more than 10 minutes. The optimization was performed using Powell's method implemented in scipy. A custom Jacobian function was written to allow quick convergence for inverse projecting multiple points in parallel.

Also note that some data points are set by placing the puck on the x, y, offset-x, or offset-y axis which define the edges of the table. In this case only the corresponding y or x values are used to compute error between the reference and projected as the other coordinate is not measured.

The solution to this minimization is the resulting 3D ArUco marker positions and polynomial coefficients.

C Puck Tracking Techniques

The goal of this section is to identify the center of the puck and opponent mallet then use the inverse projection explained in Appendix B.2.

C.1 Center Point Localization

We implemented real-time tracking of the puck position using camera imagery and contour-based detection methods. The process consists of the following sequential steps:

1. A camera captures continuous frames of the air hockey table, with the puck represented as a white circular object on a black background.
2. OpenCV's contour detection method identifies and isolates the puck by tracing its distinct boundary, indicated by the green outline in the diagram.
3. We process the contour to determine the pucks center in the image

In the end, the puck's pixel coordinate is inverse-projected to the 3D world coordinate using the method introduced earlier.

C.2 Occlusion-Robust Tracking Algorithm

The mallet gantry structure frequently occludes portions of the puck, particularly when the puck passes beneath the robot. We developed a specialized algorithm to maintain tracking accuracy under these conditions.

C.2.1 Contour Projection and Analysis

For a detected puck contour $\mathcal{C} = \{(u_i, v_i)\}_{i=1}^{n}$ in image space, we project each point to world coordinates:

$$\{(X_i, Y_i, Z_i)\} = \text{inverseProject}(\{(u_i, v_i)\}, Z_{puck}) \tag{17}$$

Where $Z_{puck} = 3.84$mm is the offset to the parameterized table height when inverse projecting. When projected to the (X, Y) plane, an unoccluded puck produces a near-circular arrangement of points, while an occluded puck produces a partial arc.

It is important to note here that under all possible positions of the puck, as long as it is slightly visible, the top of the puck (smallest X component) or the bottom of the puck (largest X component) is visible.

C.2.2 Edge Point Extraction

We then identify the extremal points of the contour:

$$X_{min} = \min_i X_i \tag{18}$$

$$X_{max} = \max_i X_i \tag{19}$$

We then extract two subsets of the contour:

$$\mathcal{C}_{min} = \{(X_i, Y_i) | X_i < X_{min} + 0.0012\} \tag{20}$$

$$\mathcal{C}_{max} = \{(X_i, Y_i) | X_i > X_{max} - 0.0012\} \tag{21}$$

This is to avoid cases where there is a line of points with almost the same x value, but the maximum x value does not lie in the center.

C.2.3 Center Candidate Generation

From these edge point subsets, we compute two candidate center positions:

$$\mathbf{c}_{min} = \text{mean}(\mathcal{C}_{min}) + (r_{puck}, 0) \tag{22}$$

$$\mathbf{c}_{max} = \text{mean}(\mathcal{C}_{max}) - (r_{puck}, 0) \tag{23}$$

Where r_{puck} is the puck radius (3.09 cm).

An example of these candidate solutions are the centers of the circles in image 8.

C.2.4 Candidate Validation

We validate each candidate by measuring how well the observed contour conforms to a circle centered at the candidate position:

$$S_j = \sum_{i=1}^{n} \delta(|||\mathbf{p}_i - \mathbf{c}_j|| - r_{puck}| < 2.5 \text{ mm}) \tag{24}$$

Where $\delta(\cdot)$ is the indicator function.

C.2.5 Final Position Selection

The final puck position is determined by:

$$\mathbf{p}_{puck} = \begin{cases} \text{centroid}(\mathcal{C}) & \text{if } ||\mathbf{c}_{min} - \mathbf{c}_{max}|| < 2 \text{ mm} \\ \mathbf{c}_j \text{ where } j = \arg\max_j S_j & \text{if } \max(S_{min}, S_{max}) > 1.5 \cdot \min(S_{min}, S_{max}) \\ \mathbf{c}_j \text{ where } j = \arg\min_j ||\mathbf{c}_j - \mathbf{p}_{prev}|| & \text{otherwise} \end{cases} \tag{25}$$

Where $\mathbf{p}_{prev}$ is the puck position from the previous frame.

This hierarchical decision process first checks if both candidates converge to approximately the same position (indicating minimal occlusion) and takes the centroid to avoid noise due to the pixels having a finite resolution. Then evaluates candidate quality based on contour conformity, and finally defaults to temporal consistency when confidence is low.

D Opponent Mallet Tracking

To track the opponent mallet, we designed an attachment to place on top of the mallet consisting of a flat circular disc covered with retroreflective material featuring a distinctive hollow center (Figure 11). This design creates a unique visual signature that allows our system to differentiate between the puck and opponent mallet using a simple yet effective image processing rule: contours with a dark centroid are classified as the opponent mallet, while solid bright contours are identified as the puck.

To accommodate different playing styles, we created two versions of the attachment. The standard version is designed for conventional players who hold the mallet with a vertical grip. The low-profile version is optimized for advanced players who adopt a professional stance with a lowered hand position closer to the table surface only holding it with two fingers. By having a lower profile we reduce the added weight and move the center of mass closer to the table, reducing the chance of the mallet tilting during rapid acceleration. Both attachments utilize the same tracking principle while preserving natural gameplay mechanics and minimizing interference with player technique.

Once the centroid is found (see Figure12), the opponent mallet position is determined using the same inverse projection methods applied to the puck (Appendix B.2.2) but with the mallet's height instead of the puck's height.

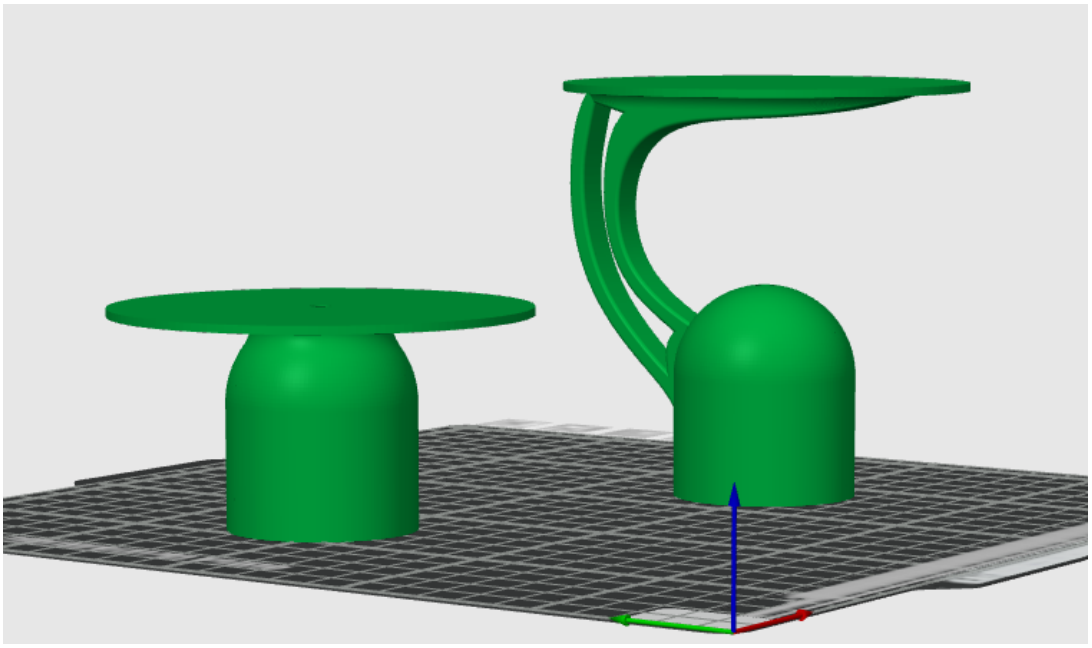


Figure 11: Opponent Mallet Designs
Mallet tracking attachments designed for different playing styles. Left: Standard attachment for conventional grip. Right: Low-profile attachment designed for professional-style grip with lowered hand position.

Figure 12: Opponent mallet localization pipeline using contour detection and moment calculation. Note that since the retroreflective surface is hollowed out in the center we get two coaxial contours.

E Simulation Mallet

E.1 Derivation of Cartesian Voltage Equations

Starting from Eq. (1) and the kinematic relations

$$y = \frac{R(-T_1 - T_2)}{2}, \quad x = \frac{R(T_1 - T_2)}{2}.$$

Note that these are different from the previous team due to a different coordinate system. In our system, from the agents side, the origin is in the bottom right corner with x going forward and y going to the left. Also note that, unfortunately, this is different from the coordinate system the camera uses to track objects.

We start with this feedforward system:

$$\begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 & b_1 & b_2 & b_3 \\ b_1 & b_2 & b_3 & a_1 & a_2 & a_3 \end{bmatrix} \begin{bmatrix} T_1^{***} \\ T_1^{**} \\ T_1^{*} \\ T_2^{***} \\ T_2^{**} \\ T_2^{*} \end{bmatrix}$$

Adding and subtracting terms gives us:

$$\begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 & b_1 & b_2 & b_3 \\ b_1 & b_2 & b_3 & a_1 & a_2 & a_3 \end{bmatrix} \begin{bmatrix} \frac{1}{2}(T_1+T_2)^{***} \\ \frac{1}{2}(T_1+T_2)^{**} \\ \frac{1}{2}(T_1+T_2)^{*} \\ \frac{1}{2}(T_1+T_2)^{***} \\ \frac{1}{2}(T_1+T_2)^{**} \\ \frac{1}{2}(T_1+T_2)^{*} \end{bmatrix} + \begin{bmatrix} a_1 & a_2 & a_3 & b_1 & b_2 & b_3 \\ b_1 & b_2 & b_3 & a_1 & a_2 & a_3 \end{bmatrix} \begin{bmatrix} \frac{1}{2}(T_1-T_2)^{***} \\ \frac{1}{2}(T_1-T_2)^{**} \\ \frac{1}{2}(T_1-T_2)^{*} \\ -\frac{1}{2}(T_1-T_2)^{***} \\ -\frac{1}{2}(T_1-T_2)^{**} \\ -\frac{1}{2}(T_1-T_2)^{*} \end{bmatrix}$$

Substituting using x and y:

$$\begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = -\frac{1}{R} \begin{bmatrix} a_1+b_1 & a_2+b_2 & a_3+b_3 \\ a_1+b_1 & a_2+b_2 & a_3+b_3 \end{bmatrix} \begin{bmatrix} y^{***} \\ y^{**} \\ y^{*} \end{bmatrix} + \frac{1}{R} \begin{bmatrix} a_1-b_1 & a_2-b_2 & a_3-b_3 \\ b_1-a_1 & b_2-a_2 & b_3-a_3 \end{bmatrix} \begin{bmatrix} x^{***} \\ x^{**} \\ x^{*} \end{bmatrix}$$

Finally, adding and subtracting the voltage vectors:

$$V_y = -V_1 - V_2 = \frac{2}{R}\left[(a_1+b_1)\dddot{y} + (a_2+b_2)\ddot{y} + (a_3+b_3)\dot{y}\right]$$

$$V_x = V_1 - V_2 = \frac{2}{R}\left[(a_1-b_1)\dddot{x} + (a_2-b_2)\ddot{x} + (a_3-b_3)\dot{x}\right]$$

E.2 Position Trajectory Derivation via Laplace Transform

We solve Eq. (3) for $x(t)$ given the voltage profile Eq. (4). Taking the Laplace transform $\mathcal{L}\{\cdot\}$:

$$\mathcal{L}\{V_x(t)\} = \hat{V}_x \frac{1 - 2e^{-t_{x1}s} + e^{-t_{x2}s}}{s}$$

The transform of Eq. (3) is:

$$\begin{aligned}\mathcal{L}\{V_x(t)\} &= \frac{2}{R}\Big[(a_1-b_1)\mathcal{L}\{\dddot{x}\}+(a_2-b_2)\mathcal{L}\{\ddot{x}\}+(a_3-b_3)\mathcal{L}\{\dot{x}\}\Big] \\ &= \frac{2}{R}\Big[\Big((a_1-b_1)s^3+(a_2-b_2)s^2+(a_3-b_3)s\Big)X(s) \\ &\quad -\Big((a_1-b_1)s^2+(a_2-b_2)s+(a_3-b_3)\Big)x(0) \\ &\quad -\Big((a_1-b_1)s+(a_2-b_2)\Big)\dot{x}(0) \\ &\quad -(a_1-b_1)\ddot{x}(0)\Big]\end{aligned}$$

Let $C_5=(a_1-b_1)$, $C_6=(a_2-b_2)$, $C_7=(a_3-b_3)$. Define initial condition terms:

$$\begin{aligned}C_2 &= C_5\ddot{x}(0)+C_6\dot{x}(0)+C_7x(0)\\ C_3 &= C_5\dot{x}(0)+C_6x(0)\\ C_4 &= C_5x(0)\end{aligned}$$

Rearranging for $X(s)=\mathcal{L}\{x(t)\}$:

$$X(s)=\frac{\frac{R}{2}\mathcal{L}\{V_x(t)\}+C_2'+C_3's+C_4's^2}{s(C_5s^2+C_6s+C_7)}$$

Let $C_1=R\hat{V}_x/2$. Substituting $\mathcal{L}\{V_x(t)\}$:

$$X(s)=\frac{C_1(1-2e^{-t_{x1}s}+e^{-t_{x2}s})+C_2s+C_3s^2+C_4s^3}{s^2(C_5s^2+C_6s+C_7)}$$

Taking the inverse Laplace transform $\mathcal{L}^{-1}\{X(s)\}$ gives $x(t)$. The final form of $x(t)$ is as follows:

We define the following quantities based on previously introduced variables:

$$\begin{aligned}A &= \sqrt{C_6^2-4C_5C_7},\\ B_1 &= 2C_7^2A.\end{aligned}$$

Define the exponent base terms:

$$a,b=\frac{-C_6-A}{2C_5},\frac{-C_6+A}{2C_5}$$

Now define:

$$\boldsymbol{A1CE}=\left[\frac{-C_6^2+AC_6+2C_5C_7}{B_1},\quad\frac{C_6^2+AC_6-2C_5C_7}{B_1},\quad\frac{1}{C_7},\quad\frac{-C_6}{C_7^2}\right].$$

$$B_2 = 2C_7A,$$

$$\boldsymbol{A2CE} = \left[\frac{-(-C_6 + A)}{B_2}, \quad \frac{-(C_6 + A)}{B_2}, \quad \frac{1}{C_7}\right].$$

$$\boldsymbol{A3CE} = \left[\frac{-1}{A}, \quad \frac{1}{A}\right].$$

$$B_4 = 2C_5A,$$

$$\boldsymbol{A4CE} = \left[\frac{C_6 + A}{B_4}, \quad \frac{-C_6 + A}{B_4}\right].$$

Now, given time t, define the exponential terms as:

$$\boldsymbol{e^{abt}} = \begin{bmatrix} e^{at} & e^{bt} \end{bmatrix}.$$

Let the function $f(t)$ using $\boldsymbol{A1CE}$ be:

$$f(t) = \boldsymbol{CE}_0 e^{at} + \boldsymbol{A1CE}_1 e^{bt} + \boldsymbol{A1CE}_2 t + \boldsymbol{A1CE}_3.$$

Let $t_{\text{ms}} = \max(t - a, 0)$. Then define:

$$g(t_{\text{ms}}) = \begin{cases} f(t_{\text{ms}}), & \text{if } t_{\text{ms}} > 0, \\ 0, & \text{otherwise.} \end{cases}$$

Similarly define:

$$g_1 = g(t - t_{x1}), \qquad g_2 = g(t - t_{x2}),$$

Then the final position expression becomes:

$$\text{pos}(t) = C_1 \left(f(t) - 2g_1 + g_2\right) + C_2 \cdot \boldsymbol{A2} + C_3 \cdot \boldsymbol{A3} + C_4 \cdot \boldsymbol{A4},$$

where:

$$\begin{aligned} \boldsymbol{A2} &= \boldsymbol{A2CE}_0 e^{at} + \boldsymbol{A2CE}_1 e^{bt} + \boldsymbol{A2CE}_2, \\ \boldsymbol{A3} &= \boldsymbol{A3CE}_0 e^{at} + \boldsymbol{A3CE}_1 e^{bt}, \\ \boldsymbol{A4} &= \boldsymbol{A4CE}_0 e^{at} + \boldsymbol{A4CE}_1 e^{bt}. \end{aligned}$$

System parameters used: $a_1 = 3.579{\times}10^{-6}$, $a_2 = 0.00571$, $a_3 = (0.0596{+}0.0467)/2 =$

0.05315 $b_1 = -1.7165 \times 10^{-6}$, $b_2 = -0.002739$, $b_3 = 0$. Pulley Radius $R = 0.035306$.

F Puck Dynamics Experiments

To determine the coefficients to be used in the puck dynamics portion of the simulation (as discussed in 2.3.2), we collected experimental data using the vision system, containing a series of (x,y) coordinates, and the timestep between these data points.

F.1 Data Segmentation

The first step in our analysis is to segment the data into a series of trajectories, each interrupted by a collision with a wall or mallet. To do this, we use the timesteps to calculate the velocity and acceleration of the puck at all times. Since the surface is uniform, we can take any large change in acceleration to be a collision, and remove the points on either side of the collision to make a series of uninterrupted trajectories.

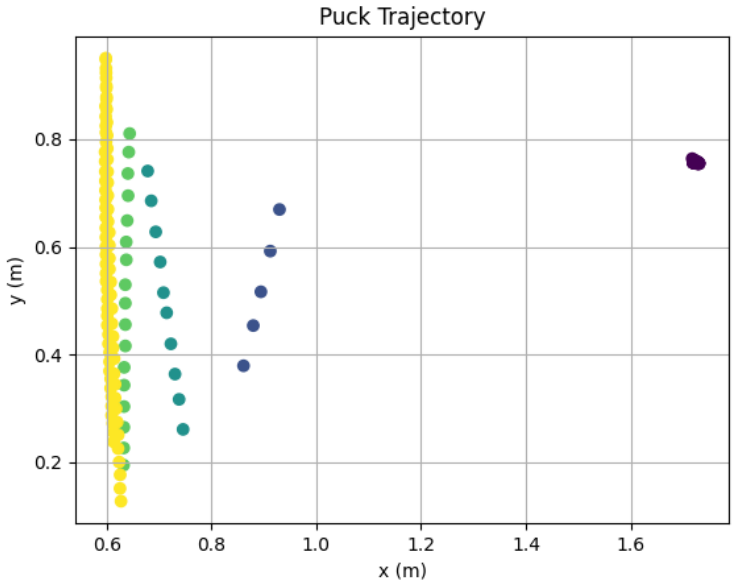


Figure 13: Example data after removing points surrounding collisions.

F.2 Mass, Friction, Air Resistance

To find values for mass, friction and air resistance, we use the analytical solution to the puck's equation of motion described in Section 2.3.2. We formulated an optimization problem to determine the three key parameters—mass (m), friction (f), and air resistance (B)—that best fit observed behavior across all trajectory segments. The mean squared error (MSE) between predicted and measured displacements served as our objective function:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{n_i}\sum_{j=1}^{n_i}(p_{\text{predicted},ij} - p_{\text{measured},ij})^2$$

For each trajectory segment, we calculated initial velocity using early displacement data rather than instantaneous measurements to reduce the impact of sensor noise. The L-BFGS-B algorithm performed the optimization within physically meaningful parameter bounds. Running this algorithm for a large number of segments gave us the values: mass (m) = 0.008kg, friction (f) = 0.001, air resistance (B) = 0.001.

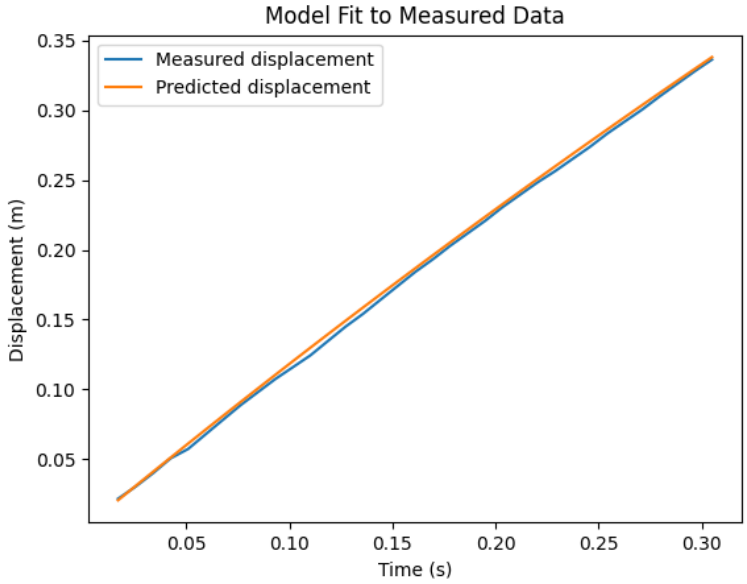


Figure 14: Example of final values for mass, friction, air resistance compared to the actual trajectory of a random segment of data.

F.3 Restitution

The final parameter for the puck dynamics simulation we determined experimentally was the coefficient of restitution, e. The coefficient of restitution is defined as $e = \sqrt{\frac{KE_f}{KE_i}}$, where the initial and final kinetic energies are calculated in the timesteps directly before and after a collision with a wall, respectively. Computing this for a large number of wall collisions gave us a restitution coefficient of approximately 0.7, however, this value had a high variance.

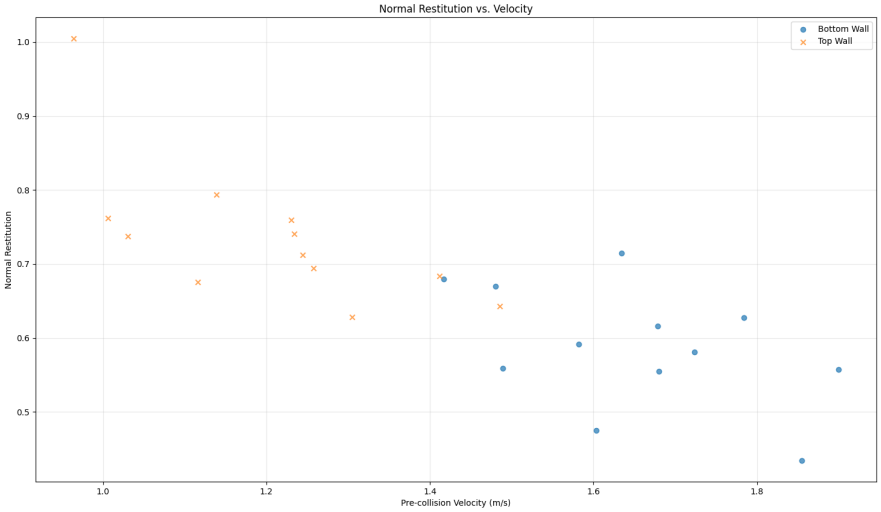


Figure 15: Data showing correlation between velocity component normal to collision plane and corresponding restitution coefficient.

After investigating the data further and based on qualitative observations, we determined that the variance in data came from two main sources: the influence of puck speed on restitution, and the differences in collisions where the velocity is primarily normal to the plane of incidence, versus the velocity being primarily parallel to the plane of incidence.

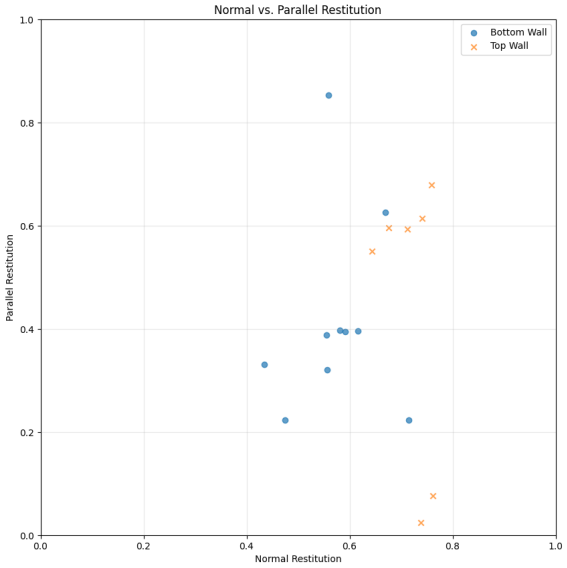


Figure 16: Data showing normal and parallel restitution computed separately.

We have yet to implement these changes into the simulation; however, preliminary analysis has been completed, and we found significant correlation between velocity and restitution and a large disparity in restitution values when considering only the normal or parallel components of restitution. These results should be implemented in next year's simulation.

G Simulating Puck Collisions

G.1 Classifying Puck Collisions

A ray casting method is employed. From the puck's current position P_0, two rays are cast parallel to the current velocity vector $\mathbf{v}$. These rays originate from points offset perpendicularly from the velocity vector by $\pm R_{puck}$ (the puck radius). The intersections of these rays with the rectangular boundary of the playing area (initially treating goal lines as solid walls) are calculated. (See Figure 9 for illustration).

The type of the first collision is determined as follows:

- **Wall Collision:** Both rays first intersect wall segments that are *not* part of the goal lines.
- **Corner Collision:** One ray first intersects a non-goal wall segment, while the other intersects the segment representing the goal line. The specific corner involved (top or bottom) is determined by comparing the average y-coordinate of the intersection points to the table's midline ($y = Width/2$).
- **Goal Entry / Pass Through:** Neither ray's first intersection is with a non-goal wall segment.

A special case is handled when the puck starts partially inside the goal area. To avoid erroneous detections, only the ray originating from the side of the puck that is outside the goal boundary is considered for determining potential corner collisions.

After identifying the collision type, the time t_{coll} is found by first finding the distance it must travel until collision and then solving Eq. (7). For more detail read Appendix G.2

G.2 Determining time to collision

Refer to 2.3.2 for context of the three collision types.

Wall Collisions: For a flat wall (e.g., at $x = X_{wall}$), the straight-line distance d_{wall} from the puck's edge to the wall along the velocity vector is calculated. The collision time t_{coll} is the time required to travel this distance according to the kinematic model,

found by solving $s(t_{coll}) = d_{wall}$ using Eq. (7). This equation is efficiently solved using a numerical root-finding algorithm applied in parrallel to $s(t) - d_{wall} = 0$ (Chandrupatla's method from Jason-s (2017)).

Corner Collisions: Collision with a corner point A occurs when the distance from the puck center $P(t)$ to A equals the puck radius R_{puck}. Assuming the initial velocity $\mathbf{v}$ dictates the direction of travel, we seek the distance s_{coll} along the path such that the puck at position $P(s_{coll})$ satisfies $|P(s_{coll}) - A| = R_{puck}$. Approximating the path locally as $P_0 + s\hat{\mathbf{v}}$, the condition becomes $|P_0 + s\hat{\mathbf{v}} - A|^2 = R_{puck}^2$. However, working directly with the non-normalized velocity $\mathbf{v}$ and finding the distance s, we solve for s using the following quadratic equation derived from $|(P_0 + s\frac{\mathbf{v}}{|\mathbf{v}|}) - A|^2 = R_{puck}^2$:

$$as^2 + bs + c = 0 \tag{26}$$

where $a = |\mathbf{v}|^2$, $b = 2\mathbf{v}\cdot(P_0 - A)$, and $c = |P_0 - A|^2 - R_{puck}^2$. The derivation and selection of the physically relevant positive root s_{coll} are in Appendix G.3. Once s_{coll} is known, the corresponding collision time t_{coll} is found by numerically solving $s(t_{coll}) = s_{coll}$ using Eq. (7).

G.3 Simulating Puck Corner Collisions

We want to find the distance s along the initial velocity direction $\hat{\mathbf{v}} = \mathbf{v}/|\mathbf{v}|$ such that the puck center $P(s) = P_0 + s\hat{\mathbf{v}}$ is exactly R_{puck} from the corner point A. The condition is $|P(s) - A|^2 = R_{puck}^2$.

$$|P_0 + s\hat{\mathbf{v}} - A|^2 = R_{puck}^2$$

Expanding the dot product: $(P_0 - A + s\hat{\mathbf{v}}) \cdot (P_0 - A + s\hat{\mathbf{v}}) = R_{puck}^2$

$$|P_0 - A|^2 + 2s\hat{\mathbf{v}} \cdot (P_0 - A) + s^2|\hat{\mathbf{v}}|^2 = R_{puck}^2$$

Since $|\hat{\mathbf{v}}|^2 = 1$, this is a quadratic equation in s:

$$s^2 + (2\hat{\mathbf{v}} \cdot (P_0 - A))s + (|P_0 - A|^2 - R_{puck}^2) = 0$$

This matches the form $s^2 + b's + c' = 0$.

Alternatively, using the non-normalized velocity $\mathbf{v}$ and finding the scaling factor k such that $|P_0 + k\mathbf{v} - A|^2 = R_{puck}^2$:

$$|P_0 - A|^2 + 2k\mathbf{v} \cdot (P_0 - A) + k^2|\mathbf{v}|^2 = R_{puck}^2$$

Rearranging into the form $ak^2 + bk + c = 0$:

- $a = |\mathbf{v}|^2 = v_x^2 + v_y^2$

- $b = 2\mathbf{v} \cdot (P_0 - A) = 2(v_x(x_0 - A_x) + v_y(y_0 - A_y))$
- $c = |P_0 - A|^2 - R_{puck}^2 = (x_0 - A_x)^2 + (y_0 - A_y)^2 - R_{puck}^2$

The distance is then $s = k|\mathbf{v}|$. The quadratic formula gives $k = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$. The physically relevant solution corresponds to forward motion ($k > 0$) and the first time the condition is met (smaller positive k). Usually, this corresponds to the $k = \frac{-b-\sqrt{b^2-4ac}}{2a}$ root, assuming $\mathbf{v}$ points generally towards A initially, but care must be taken, especially if the puck starts very close to the corner. The distance is $s = k|\mathbf{v}| = \frac{-b-\sqrt{b^2-4ac}}{2a}\sqrt{a} = \frac{-b-\sqrt{b^2-4ac}}{2\sqrt{a}}$.

H Derivation of Puck-Mallet Collision Lower Bound

This section details the derivation of the lower bound time t_{lb} for a collision between the puck and the mallet, used for the efficient collision detection algorithm described in Section 2.3.3.

Bounding States and Relative Frame for Lower Bound Calculation To calculate t_{lb} for a prospective time interval starting now, we first establish bounds on the relevant states:

- **Puck Velocity:** The puck's speed v_p changes according to Eq. (7). We determine the range of possible speeds $[v_p^{min}, v_p^{max}]$ it can have during the interval, considering its current speed and deceleration. For the lower bound calculation, we will select an optimal speed v_p within this range that minimizes potential collision time.
- **Mallet Velocity:** The mallet's maximum possible speed v_m^{max} is obtained based on its current velocity and the maximum acceleration capabilities derived from its dynamics model (detailed in Appendix E.2). This represents the fastest the mallet could possibly move to intercept.

We then analyze the situation in a relative coordinate system. Let the puck's (optimal bounded) velocity vector define the x-axis. The initial relative position of the mallet center with respect to the puck center is expressed as $(d_\parallel, d_\perp)$, where $d_\parallel$ is the component along the puck's velocity direction and $d_\perp$ is the perpendicular component. Collision occurs when the distance between centers becomes equal to $R = R_{puck} + R_{mallet}$.

Principle of Minimum Collision Time (Lower Bound) The lower bound t_{lb} represents the absolute minimum time required for a collision, assuming worst-case (fastest collision) conditions:

1. The mallet moves constantly at its maximum possible speed, v_m^{max}.
2. The puck moves constantly at the speed v_p (chosen from $[v_p^{min}, v_p^{max}]$) that, combined with the optimal mallet action, leads to the quickest collision. (The determination of this optimal v_p is detailed in Appendix H).
3. The mallet moves in the optimal constant direction (angle θ_m relative to the puck's velocity) required to achieve collision in minimum time.

Geometrically, the minimum time occurs when the mallet's velocity vector $\mathbf{v_m}$ is such that $\mathbf{x_m^c} - \mathbf{x_p^c} = \lambda \mathbf{v_m}$ for some λ where $\mathbf{x_m^c}, \mathbf{x_p^c}$ are the mallet and puck positions at the collision.

Calculation of Optimal Angle and Lower Bound Time Finding the optimal mallet angle θ_m and the minimum time t_{lb} requires solving the kinematic equations under the minimum time condition. The derivation (later in this section) yields the following calculations: First, intermediate coefficients q_a, q_b, q_c are calculated:

$$q_a = v_m^{max} d_\perp \tag{27}$$
$$q_b = v_m^{max} d_\parallel + R v_p \tag{28}$$
$$q_c = -v_p d_\perp \tag{29}$$

An angle θ_n, related to the optimal mallet velocity direction, is found using:

$$\theta_n = \begin{cases} -2\arctan(q_b/|q_a|) & \text{if } |q_b - q_c| < \epsilon \text{ (handle singularity)} \\ 2\arctan\left(\frac{q_a - \sqrt{q_a^2 + q_b^2 - q_c^2}}{q_b - q_c}\right) & \text{otherwise} \end{cases} \tag{30}$$

where ϵ is a small tolerance. Potential errors (e.g., negative value under the square root) indicate no real solution exists under these conditions. The optimal mallet angle in the relative frame is $\theta_m = \theta_n - \pi/2$.

With θ_m determined, the time t_{lb} is the smallest positive real root of the quadratic equation $s_a t^2 + s_b t + s_c = 0$, derived from the condition that the squared distance between centers equals R_{sum}^2 at time t. The coefficients are:

$$s_a = v_p^2 + (v_m^{max})^2 - 2 v_m^{max} v_p \cos(\theta_m) \tag{31}$$
$$s_b = -2\Big((v_p - v_m^{max}\cos(\theta_m))(-d_\parallel) + (-v_m^{max}\sin(\theta_m))(-d_\perp)\Big) \tag{32}$$
$$s_c = d_\parallel^2 + d_\perp^2 - R^2 \tag{33}$$

The lower bound time is:

$$t_{lb} = \frac{-s_b - \sqrt{s_b^2 - 4 s_a s_c}}{2 s_a} \tag{34}$$

If the discriminant $s_b^2 - 4s_a s_c$ is negative or if no positive real root exists, it implies that a collision cannot occur under these worst-case assumptions within this framework, meaning t_{lb} is effectively infinite for the purpose of this step. The implementation includes checks for such error conditions.

H.1 Problem Setup and Kinematics

We aim to find the minimum time t for the distance between the puck center $P(t)$ and mallet center $M(t)$ to become $R = R_{puck} + R_{mallet}$.

Coordinate System: We use a relative coordinate system where the puck's velocity defines the positive x-axis. **Initial State:** At $t = 0$, the mallet's position relative to the puck is $\mathbf{d}_0 = (d_\parallel, d_\perp)$. **Velocities (Constant Approximation for Lower Bound):**

- Puck velocity: $\mathbf{v}_p = (v_p, 0)$. v_p is assumed constant for this bound calculation, chosen optimally within its allowed range $[v_p^{min}, v_p^{max}]$.
- Mallet velocity: $\mathbf{v}_m = (v_{m,max}\cos\theta_m, v_{m,max}\sin\theta_m)$. The mallet moves at its assumed maximum possible speed $v_{m,max}$ at an angle θ_m relative to the puck's velocity direction (the x-axis). θ_m is the angle to be optimized.

Relative Velocity: $\mathbf{v}_{rel} = \mathbf{v}_p - \mathbf{v}_m = (v_p - v_{m,max}\cos\theta_m, -v_{m,max}\sin\theta_m)$. **Relative Position at time t:** The mallet's position relative to the puck at time t is $\mathbf{d}(t) = \mathbf{d}_0 + (\mathbf{v}_m - \mathbf{v}_p)t = \mathbf{d}_0 - \mathbf{v}_{rel}t$.

H.2 Collision Time Quadratic Equation

Collision occurs when $|\mathbf{d}(t)| = R$, or $|\mathbf{d}(t)|^2 = R^2$.

$$|\mathbf{d}_0 - \mathbf{v}_{rel}t|^2 = R^2$$

Expanding the dot product: $(\mathbf{d}_0 - \mathbf{v}_{rel}t)\cdot(\mathbf{d}_0 - \mathbf{v}_{rel}t) = R^2$

$$|\mathbf{d}_0|^2 - 2t(\mathbf{d}_0\cdot\mathbf{v}_{rel}) + t^2|\mathbf{v}_{rel}|^2 = R^2$$

This is a quadratic equation for the collision time t of the form $s_a t^2 + s_b t + s_c = 0$, where the coefficients depend on the mallet angle θ_m:

- $s_a(\theta_m) = |\mathbf{v}_{rel}|^2 = (v_p - v_{m,max}\cos\theta_m)^2 + (-v_{m,max}\sin\theta_m)^2$

$$\begin{aligned} s_a(\theta_m) &= v_p^2 - 2v_p v_{m,max}\cos\theta_m + v_{m,max}^2\cos^2\theta_m + v_{m,max}^2\sin^2\theta_m \\ &= v_p^2 + v_{m,max}^2 - 2v_p v_{m,max}\cos\theta_m \quad \text{(Matches Eq. (31))} \end{aligned}$$

- $s_b(\theta_m) = -2(\mathbf{d}_0 \cdot \mathbf{v}_{rel}) = -2((d_\parallel, d_\perp) \cdot (v_p - v_{m,max}\cos\theta_m, -v_{m,max}\sin\theta_m))$

  $$\begin{aligned} s_b(\theta_m) &= -2[d_\parallel(v_p - v_{m,max}\cos\theta_m) + d_\perp(-v_{m,max}\sin\theta_m)] \\ &= -2d_\parallel v_p + 2d_\parallel v_{m,max}\cos\theta_m + 2d_\perp v_{m,max}\sin\theta_m \quad \text{(Matches structure of Eq. (32))} \end{aligned}$$

  Note: Eq. (32) in the main text is $s_b = -2\Big((v_p - v_{m,max}\cos(\theta_m))(-d_\parallel) + (-v_{m,max}\sin(\theta_m))(-d_\perp)\Big)$. This is equivalent.

- $s_c = |\mathbf{d}_0|^2 - R^2 = d_\parallel^2 + d_\perp^2 - R^2$ (Matches Eq. (33))

The collision time for a given θ_m is the smallest positive real root: $t(\theta_m) = \frac{-s_b(\theta_m) - \sqrt{s_b(\theta_m)^2 - 4s_a(\theta_m)s_c}}{2s_a(\theta_m)}$.

H.3 Minimizing Collision Time t via Geometric Condition

To find the mallet angle θ_m that minimizes the collision time $t(\theta_m)$ derived from the quadratic equation $s_a t^2 + s_b t + s_c = 0$, we apply the provided geometric condition for minimum time: The vector connecting the mallet center to the puck center *at the moment of collision*, $\mathbf{d}_c = P(t_{lb}) - M(t_{lb})$, must be parallel to the mallet's velocity vector $\mathbf{v}_m$.

$$\mathbf{d}_c = \lambda \mathbf{v}_m \quad \text{for some scalar } \lambda > 0 \tag{35}$$

Since $|\mathbf{d}_c| = R$ (sum of radii) and $|\mathbf{v}_m| = v_{m,max}$, taking the magnitude gives $|\lambda| v_{m,max} = R$, so $\lambda = R/v_{m,max}$. Let $\hat{\theta} = \mathbf{v}_m / v_{m,max}$ be the unit vector in the direction of mallet velocity. Then $\mathbf{d}_c = (R)\hat{\theta}$.

Recall the relative position vector at time t: $\mathbf{d}(t) = \mathbf{d}_0 - \mathbf{v}_{rel} t = \mathbf{d}_0 - (\mathbf{v}_p - \mathbf{v}_m)t$. At the collision time t_{lb}, $\mathbf{d}(t_{lb}) = \mathbf{d}_c$. Substituting $\mathbf{d}_c = \lambda \mathbf{v}_m$:

$$\begin{aligned} \lambda \mathbf{v}_m &= \mathbf{d}_0 - \mathbf{v}_p t_{lb} + \mathbf{v}_m t_{lb} \\ \mathbf{v}_m(\lambda - t_{lb}) &= \mathbf{d}_0 - \mathbf{v}_p t_{lb} \end{aligned}$$

This equation shows that the vector $\mathbf{d}_0 - \mathbf{v}_p t_{lb}$ must be parallel to $\mathbf{v}_m$. Let's express this using vector components in our reference frame (puck velocity along x-axis): $\mathbf{d}_0 = (d_\parallel, d_\perp)$ $\mathbf{v}_p = (v_p, 0)$ $\mathbf{v}_m = (v_{m,max}\cos\theta_m, v_{m,max}\sin\theta_m)$

This results in

$$v_{m,max}\cos\theta_m(R/v_{m,max} - t_{lb}) = d_\parallel - v_p t_{lb}$$

$$v_{m,max}\sin\theta_m(R/v_{m,max} - t_{lb}) = d_\perp$$

Using substitution we get

$$v_{m,max}\cos\theta_m\left(\frac{d_\perp}{v_{m,max}\sin\theta_m}\right) = d_\parallel - v_p\left(R/v_{m,max} - \frac{d_\perp}{v_{m,max}\sin\theta_m}\right)$$

so

$$\cos\theta_m(d_\perp) = d_\parallel \sin(\theta_m) - \frac{v_p}{v_{m,max}}(R\sin(\theta_m) - d_\perp)$$

Rearranging this give us

$$q_a \cos\theta_m - q_b \sin\theta_m + q_c = 0$$

where

$$\begin{aligned} q_a &= v_{m,max} d_\perp \\ q_b &= v_{m,max} d_\parallel + R v_p \\ q_c &= -v_p d_\perp \end{aligned}$$

Substituting $\theta_n = \theta_m + \pi/2$ (so $\cos\theta_m = \sin\theta_n$ and $\sin\theta_m = -\cos\theta_n$):

$$\begin{aligned} q_a \sin\theta_n - q_b(-\cos\theta_n) + q_c &= 0 \\ q_a \sin\theta_n + q_b \cos\theta_n + q_c &= 0 \end{aligned}$$

This is the equation that is solved using the tangent half-angle substitution to find θ_n, as shown in the next subsection. The geometric condition provided, therefore, leads correctly to the equation solved for the optimal angle.

H.4 Solving for the Optimal Angle

The equation $q_a \sin\theta_n + q_b \cos\theta_n + q_c = 0$ is solved for θ_n. We use the tangent half-angle substitution $T = \tan(\theta_n/2)$, which gives $\sin\theta_n = \frac{2T}{1+T^2}$ and $\cos\theta_n = \frac{1-T^2}{1+T^2}$. Substituting into the equation:

$$\begin{aligned} q_a\left(\frac{2T}{1+T^2}\right) + q_b\left(\frac{1-T^2}{1+T^2}\right) + q_c &= 0 \\ 2q_aT + q_b(1-T^2) + q_c(1+T^2) &= 0 \\ (q_c - q_b)T^2 + (2q_a)T + (q_b + q_c) &= 0 \end{aligned}$$

This is a quadratic equation for $T = \tan(\theta_n/2)$. Solving for T:

$$T = \frac{-2q_a \pm \sqrt{(2q_a)^2 - 4(q_c - q_b)(q_b + q_c)}}{2(q_c - q_b)} = \frac{-q_a \pm \sqrt{q_a^2 - (q_c^2 - q_b^2)}}{q_c - q_b} = \frac{-q_a \pm \sqrt{q_a^2 + q_b^2 - q_c^2}}{q_c - q_b}$$

Then $\theta_n = 2\arctan(T)$. Choosing the appropriate root for T gives the solution presented in Eq. (30):

$$\theta_n = 2\arctan\left(\frac{q_a - \sqrt{q_a^2 + q_b^2 - q_c^2}}{q_b - q_c}\right)$$

(Note: The specific form handles numerical stability and selects the correct physical solution. The case $q_b \approx q_c$ corresponds to $q_c - q_b \approx 0$, where the quadratic equation simplifies, leading to the alternate form in Eq. (30).) The optimal mallet angle is then $\theta_m = \theta_n - \pi/2$.

H.5 Selecting Optimal Puck Speed v_p

The derivations above assume a fixed puck speed v_p. However, v_p can vary within $[v_p^{min}, v_p^{max}]$. The value used in the q_a, q_b, q_c formulas should be the one (within the bounds) that helps minimize the collision time t_{lb}. First, calculate an "ideal" unbounded puck speed v_p^{ideal}. This is the speed required for the puck to reach the collision longitude ($d_\parallel$) exactly when the mallet reaches the collision latitude ($d_\perp$) assuming the mallet takes the shortest path (perpendicular to puck motion, angle $\theta_m = \pm\pi/2$), and $|d_\perp| > R_{puck}$. Mallet time to cover $d_\perp$: $t_{ideal} = (|d_\perp| - |R_{puck}|)/v_{m,max}$. Puck speed needed to cover $d_\parallel$ in t_{ideal}: $v_p^{ideal} = d_\parallel / t_{ideal}$. (Handle $|d_\perp| < R_{puck}$ case appropriately, e.g., infinite ideal speed). The actual v_p used in calculations is clamped to the allowed range:

$$v_p = \text{clip}(v_p^{ideal}, v_p^{min}, v_p^{max})$$

This clipped value v_p is used in the definitions of q_a, q_b, q_c and s_a, s_b, s_c.

H.6 Calculating the Lower Bound Time t_{lb}

With the optimal angle θ_m (derived via θ_n) and the optimal bounded puck speed v_p determined, the coefficients s_a, s_b, s_c are calculated using Eqs. (31)-(33). The minimum collision time (the lower bound t_{lb}) is then found using the quadratic formula, taking the smaller positive real root:

$$t_{lb} = \frac{-s_b - \sqrt{s_b^2 - 4s_a s_c}}{2s_a}$$

If the discriminant $s_b^2 - 4s_a s_c$ is negative or zero, or if s_a is zero and s_b doesn't yield a positive time, no collision is possible under these optimized, constant-velocity assumptions, and t_{lb} is considered infinite.

I Action Space Derivations

Refer to 2.4.2 for context on the agents action space. This section outlines how to relate the agents output $(|V_x|, |V_y|, \mathbf{x_f})$ to the simulation voltage function $(V_x, V_y, t_{x1}, t_{x2}, t_{y1}, t_{y2})$.

Firstly, the voltage constraint $|V_x|+|V_y| \leq 2V_{\text{max}}$ must be satisfied. If the agent's outputted magnitudes $|V_x|$ and $|V_y|$ violate this, the algorithm projects the vector $(|V_x|, |V_y|)$ onto the boundary $|V_x| + |V_y| = 2V_{\text{max}}$ by finding the closest point satisfying the constraint.

Secondly, the signs of V_x and V_y are determined. Taking the x-dimension as an example (the y-dimension follows analogously), the initial sign of V_x is chosen based on the target direction: positive if $x_f > x_0$ and negative if $x_f < x_0$. However, in certain cases where the mallet is traveling too fast towards its final position we require to instead decelerate then accelerate (see Figure 17). To check for this "overshooting" scenario, we calculate a theoretical overshoot position, x_{over}. This position represents where the mallet would come to rest if it immediately started decelerating. The derivation of x_{over} is detailed in Appendix I.1. If the target position x_f lies between the initial position x_0 and the overshoot position x_{over} (i.e., $x_{\text{over}} > x_f > x_0$ or $x_0 > x_f > x_{\text{over}}$ as seen on the middle of Figure 17), it means the mallet would pass x_f even while maximally braking. In this overshoot case, the sign of V_x is flipped, commanding the mallet to initially accelerate *away* from x_f before braking towards it (right image in Figure 17).

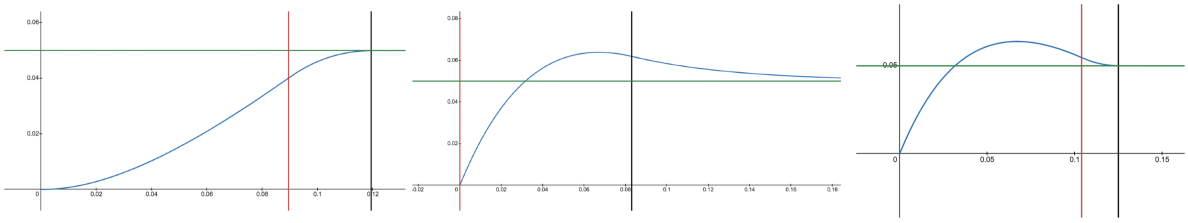


Figure 17: Position Curves Under Different Initial Conditions
Green line: Final position. Blue line: Position over time. Red/Black verticle line: ending period of acceleration/deceleration (i.e. t_{x1} and t_{x2}).
Left: Initial velocity is low and so no overshoot occurs, in this case we accelerate then decelerate
Middle: Initial velocity is high causing overshoot, without changing the sign of Vx the best solution is to only start decelerating then slowly converge to the final position
Right: Same as middle but changing the sign of Vx, in this case we decelerate more than previously and accelerate again to converge to the final position

Thirdly, a crucial safety check is performed using the calculated overshoot position

x_{over}. While operating with constant V_x, V_y we can guarantee no collisions with the table walls. However, with variable voltages, it is possible to get a high-magnitude voltage action towards a boundary, followed by a low-magnitude action; the low-magnitude action might lack sufficient braking force to stop the mallet (still coasting from the previous high-velocity state) before it hits the boundary during the theoretical overshoot path. Therefore, if x_{over} falls outside the playable area $[x_{\text{min}}, x_{\text{max}}]$, the algorithm modifies the magnitude $|V_x|$ to the minimum value required such that the recalculated x_{over} lies exactly at the boundary. The corresponding $|V_y|$ magnitude is then adjusted (decreased) to ensure the constraint $|V_x| + |V_y| \leq 2V_{\text{max}}$ remains satisfied. This intervention happens infrequently, primarily because the system can decelerate relatively quickly even at lower voltage commands. We hypothesize that these rare adjustments do not significantly impede the RL agent's learning process.

Finally, with the definitive signed voltages V_x and V_y determined and safety checks passed, the timing parameters (t_{x1}, t_{x2} for the x-dimension, t_{y1}, t_{y2} for the y-dimension) are calculated. Taking the x-dimension again, two conditions are imposed:

1. The mallet must asymptotically approach the target position: $\lim_{t\to\infty} x(t) = x_f$. Applying this condition to the system dynamics yields a direct relationship between t_{x1} and t_{x2}, as shown in Appendix I.2:

$$t_{x2} = 2t_{x1} - \frac{x_f C_7}{C_1} + \frac{C_2}{C_1} \tag{36}$$

2. The mallet must be stationary at the target position exactly at the moment deceleration ends, i.e., $x(t_{x2}) = x_f$. This ensures fastest possible converges to x_f.

Substituting the expression for t_{x2} from condition 1 into the equation from condition 2 results in a single equation involving t_{x1}. This equation is solved numerically for t_{x1} using Newton's method. Once t_{x1} is found, t_{x2} is calculated directly. The same procedure is applied independently for the y-dimension to find t_{y1} and t_{y2}.

I.1 Derivation of Overshoot Position x_{over}

We use the notation and dynamics equations outlined in Appendix E.2.

Recall the position expression for the mallet:

$$\text{pos}(t) = C_1 \left(f(t) - 2g(t - t_1) + g(t - t_2)\right) + C_2 \cdot \boldsymbol{A2}(t) + C_3 \cdot \boldsymbol{A3}(t) + C_4 \cdot \boldsymbol{A4}(t),$$

where the subscript x or y is omitted for generality (e.g., t_1 represents t_{x1} or t_{y1}).

The overshoot position x_{over} is defined as the position where the mallet comes to rest under the condition that it begins decelerating immediately away from the final position. This corresponds to setting $t_{x1} = 0$ and solving $x_{\text{overshoot}}$ such that $\text{pos}(t_{x2}) = x_{\text{overshoot}}$, where $t_{x2} = -x_{\text{overshoot}}\frac{C7}{C1} + \frac{C2}{C1}$ is given in Appendix I.2

With $t_{x1} = 0$, we have $g(t - t_{x1}) = g(t) = f(t)$ for $t > 0$. Additionally, $g(t_{x2} - t_{x2}) = g(0) = 0$.

Substituting these into the position equation for $t > 0$:

$$\begin{aligned}\text{pos}(t_{x2}) &= C_1\left(f(t_{x2}) - 2f(t_{x2}) + 0\right) + C_2\boldsymbol{A2}(t_{x2}) + C_3\boldsymbol{A3}(t_{x2}) + C_4\boldsymbol{A4}(t_{x2}) \\ &= -C_1 f(t_{x2}) + C_2\boldsymbol{A2}(t_{x2}) + C_3\boldsymbol{A3}(t_{x2}) + C_4\boldsymbol{A4}(t_{x2}) = x_{\text{overshoot}}\end{aligned}$$

Substituting the full expressions for $f(t_{x2}), \boldsymbol{A2}, \boldsymbol{A3}, \boldsymbol{A4}$:

$$\begin{aligned}\text{pos}(t_{x2}) = -C_1(\boldsymbol{CE}_0 e^{at_{x2}} + \boldsymbol{A1CE}_1 e^{bt_{x2}} + \boldsymbol{A1CE}_2 t_{x2} + \boldsymbol{A1CE}_3) \\ +C_2(\boldsymbol{A2CE}_0 e^{at_{x2}} + \boldsymbol{A2CE}_1 e^{bt_{x2}} + \boldsymbol{A2CE}_2) \\ +C_3(\boldsymbol{A3CE}_0 e^{at_{x2}} + \boldsymbol{A3CE}_1 e^{bt_{x2}}) \\ +C_4(\boldsymbol{A4CE}_0 e^{at_{x2}} + \boldsymbol{A4CE}_1 e^{bt_{x2}}) \\ = x_{\text{overshoot}}\end{aligned}$$

However, note that for the feedforward model we are using, $a \approx -1500$, so for our purposes we have $e^{at_{x2}} \approx 0$ giving us

$$\begin{aligned}\text{pos}(t_{x2}) = -C_1(\boldsymbol{A1CE}_1 e^{bt_{x2}} + \boldsymbol{A1CE}_2 t_{x2} + \boldsymbol{A1CE}_3) \\ +C_2(\boldsymbol{A2CE}_1 e^{bt_{x2}} + \boldsymbol{A2CE}_2) \\ +C_3(\boldsymbol{A3CE}_1 e^{bt_{x2}}) \\ +C_4(\boldsymbol{A4CE}_1 e^{bt_{x2}}) \\ = x_{\text{overshoot}}\end{aligned}$$

Subbing in the t_{x2} not in the exponential and simplifying gives:

$$\begin{aligned}\text{pos}(t_{x2}) = -C_1(\boldsymbol{A1CE}_1 e^{bt_{x2}}) \\ +C_2(\boldsymbol{A2CE}_1 e^{bt_{x2}}) \\ +C_3(\boldsymbol{A3CE}_1 e^{bt_{x2}}) \\ +C_4(\boldsymbol{A4CE}_1 e^{bt_{x2}}) \\ = C_1\boldsymbol{A1CE}_3\end{aligned}$$

Isolating for t_{x2} gives:

$$t_{x2} = -x_{\text{overshoot}}\frac{C7}{C1}+\frac{C2}{C1} = \frac{1}{b}\ln\left(\frac{C1\boldsymbol{A1CE}_3}{-C1\boldsymbol{A1CE}_1 + C_2\boldsymbol{A2CE}_1 + C_3\boldsymbol{A3CE}_1 + C4\boldsymbol{A4CE}_1}\right)$$

Isolating for $x_{\text{overshoot}}$ results in

$$x_{\text{overshoot}} = \frac{C_2}{C_7} - \frac{C_1}{bC_7}\ln\left(\frac{C_1\boldsymbol{A1CE}_3}{-C_1\boldsymbol{A1CE}_1 + C_2\boldsymbol{A2CE}_1 + C_3\boldsymbol{A3CE}_1 + C_4\boldsymbol{A4CE}_1}\right)$$

I.2 Derivation of Relationship between t_{x1} and t_{x2}

We require the mallet to settle at the final position x_f as $t \to \infty$. This means $\lim_{t\to\infty} x(t) = x_f$. We analyze the asymptotic behavior of the terms in the position equation, assuming the system is stable ($a < 0$ and $b < 0$).

As $t \to \infty$:

- $e^{at} \to 0$ and $e^{bt} \to 0$.
- $f(t) = \boldsymbol{CE}_0e^{at} + \boldsymbol{A1CE}_1e^{bt} + \boldsymbol{A1CE}_2t + \boldsymbol{A1CE}_3 \longrightarrow \boldsymbol{A1CE}_2t + \boldsymbol{A1CE}_3$.
- $g(t-t_1)$: Since $t \to \infty$, $t - t_1 \to \infty$. So $g(t-t_1) \approx f(t-t_1) \longrightarrow \boldsymbol{A1CE}_2(t-t_1) + \boldsymbol{A1CE}_3$.
- $g(t-t_2)$: Similarly, $g(t-t_2) \approx f(t-t_2) \longrightarrow \boldsymbol{A1CE}_2(t-t_2) + \boldsymbol{A1CE}_3$.
- $\boldsymbol{A2}(t) = \boldsymbol{A2CE}_0e^{at} + \boldsymbol{A2CE}_1e^{bt} + \boldsymbol{A2CE}_2 \longrightarrow \boldsymbol{A2CE}_2$.
- $\boldsymbol{A3}(t) = \boldsymbol{A3CE}_0e^{at} + \boldsymbol{A3CE}_1e^{bt} \longrightarrow 0$.
- $\boldsymbol{A4}(t) = \boldsymbol{A4CE}_0e^{at} + \boldsymbol{A4CE}_1e^{bt} \longrightarrow 0$.

Substituting these limits into the position equation:

$$\begin{aligned}
x_f = \lim_{t\to\infty} x(t) &= C_1[(\boldsymbol{A1CE}_2 t + \boldsymbol{A1CE}_3) - 2(\boldsymbol{A1CE}_2(t - t_1) \\
&\quad + \boldsymbol{A1CE}_3) + (\boldsymbol{A1CE}_2(t - t_2) + \boldsymbol{A1CE}_3)] \\
&\quad + C_2 \cdot \boldsymbol{A2CE}_2 + C_3 \cdot 0 + C_4 \cdot 0 \\
&= C_1[\boldsymbol{A1CE}_2 t + \boldsymbol{A1CE}_3 - 2\boldsymbol{A1CE}_2 t + 2\boldsymbol{A1CE}_2 t_1 \\
&\quad - 2\boldsymbol{A1CE}_3 + \boldsymbol{A1CE}_2 t - \boldsymbol{A1CE}_2 t_2 + \boldsymbol{A1CE}_3] \\
&\quad + C_2 \boldsymbol{A2CE}_2 \\
&= C_1[(\boldsymbol{A1CE}_2 - 2\boldsymbol{A1CE}_2 + \boldsymbol{A1CE}_2)t + (\boldsymbol{A1CE}_3 \\
&\quad - 2\boldsymbol{A1CE}_3 + \boldsymbol{A1CE}_3) + (2\boldsymbol{A1CE}_2 t_1 - \boldsymbol{A1CE}_2 t_2)] \\
&\quad + C_2 \boldsymbol{A2CE}_2 \\
&= C_1\,[0 \cdot t + 0 + \boldsymbol{A1CE}_2(2t_1 - t_2)] + C_2 \boldsymbol{A2CE}_2 \\
&= C_1 \boldsymbol{A1CE}_2(2t_1 - t_2) + C_2 \boldsymbol{A2CE}_2
\end{aligned}$$

Rearranging to solve for t_2 (representing t_{x2} or t_{y2}):

$$C_1 \boldsymbol{A1CE}_2 t_2 = C_1 \boldsymbol{A1CE}_2(2t_1) + C_2 \boldsymbol{A2CE}_2 - x_f$$

$$t_2 = 2t_1 + \frac{C_2 \boldsymbol{A2CE}_2 - x_f}{C_1 \boldsymbol{A1CE}_2}$$

Subbing in $\boldsymbol{A1CE}_2 = \frac{1}{C_7}$ and $\boldsymbol{A2CE}_2 = \frac{1}{C_7}$ results in:
$t_{x2} = 2t_{x1} - x_f \frac{C_7}{C_1} + \frac{C_2}{C_1}$

J Reward Structure Details

We assume the table coordinates are $x \in [0, 2]$ (agent goal at $x = 0$, opponent goal at $x = 2$, midline $x = 1$) and $y \in [-W/2, W/2]$ where W is the table width. Puck state is $\mathbf{p} = (x_p, y_p)$, $\mathbf{v}_p = (v_{px}, v_{py})$.

J.1 Defence Mode Episode

A defence episode is active when the defence flag is true.

Start Condition The episode begins when the system state indicates the opponent has effective control or possession of the puck (e.g., puck is on opponent's side $x_p > 1$ after a reset or failed agent attack).

Termination and Rewards The episode terminates under the following conditions:

1. **Successful Defence Clear:** The episode terminates with **reward = +2** if, after an action by the agent, the puck satisfies all of the following conditions:

   - Puck is on agent's side near midline: $x_p \in [0.35, 0.95]$ (assuming table length 2, agent goal at 0).
   - Puck has sufficient speed: $|\mathbf{v}_p| > 0.4$.
   - Puck direction is near-vertical: $\frac{|v_{py}|}{|v_{px}|} > 7$ (to avoid division by zero, check $v_{px} \neq 0$).

   The simulation continues without reset, and a new episode starts with an attack flag.

2. **Goal Conceded:** The episode terminates with **reward = -1** if the puck enters the agent's goal (e.g., $x_p < 0$). The simulation environment is typically reset. A new Defence episode begins after reset.

3. **Possession Lost:** The episode terminates if the puck crosses the midline into the opponent's half ($x_p > 1$). The outcome determines the reward and the start of an Attack episode (defence flag becomes false).

   - Let $P(\text{Projected score}|\neg\text{OppMallet})$ be the event that the puck's projected trajectory enters the opponent's goal at $x = 2$ ignoring the opponent's mallet. 4
   - Let $P(\text{Projected score}|\text{OppMallet})$ be the event that the puck's projected trajectory enters the opponent's goal at $x = 2$ considering the opponent mallet's current position when the puck crosses the half way line.
   - If $P(\text{Projected score}|\text{OppMallet})$ is true: **Reward =** $+5|v_p|/120$.
   - Else if $P(\text{Projected score}|\neg\text{OppMallet})$ is true: **Reward =** $+|v_p|/120$.
   - Otherwise (projected miss): **Reward = -0.05**.

   In all sub-cases, the simulation continues without reset, and a new Attack episode begins.

J.2 Attack Mode Episode

An attack episode is active when the defence flag is false.

Start Condition The episode beings immediately after a successful defence clear via condition 1.

Termination and Rewards The episode terminates under the following conditions:

1. **Shot Outcome Check:** The episode terminates when the puck crosses the midline into the opponent's half ($x_p > 1$). Reward depends on the projected outcome. Using the same projection definitions as in Defence mode:

   - If $P(\text{Projected score}|\text{OppMallet})$ is true: **Reward =** $+|v_p|^2/10$.
   - Else if $P(\text{Projected score}|\neg\text{OppMallet})$ is true: **Reward =** $+|v_p|^2/50$.
   - Otherwise (projected miss): **Reward = +0**.

   In all sub-cases, the simulation continues without reset, and a new Defence episode begins (defence_flag becomes true).

2. **Own Goal:** The episode terminates with **reward = -1** if the puck enters the agent's own goal ($x_p < 0$). The simulation environment is likely reset. A new Defence episode begins after reset.

J.3 Continuous Penalties

These penalties are applied at every timestep t, regardless of the active mode (Attack or Defence). The total reward at step t includes these penalties added to any terminal reward if the episode ends at step t.

Let $m_{t-1} \in \mathbb{R}^2$ be the actual position of the agent's mallet at the end of the previous timestep. Let $x_{f,t} = (x_{f,t}, y_{f,t}) \in \mathbb{R}^2$ be the target position component of the agent's action at timestep t. Let $v_{\text{cmd},t} = (|V_x|_t, |V_y|_t) \in \mathbb{R}^2$ be the voltage magnitude components outputted by the agent's policy at timestep t. (Note: These are the magnitudes before algorithmic processing like sign determination or projection).

The continuous penalty $R_{\text{cont},t}$ is calculated as:

$$
\begin{aligned}
d_{\text{move},t} &= ||\mathbf{m}_{t-1} - \mathbf{x}_{f,t}||_2 = \sqrt{(x_{m,t-1} - x_{f,t})^2 + (y_{m,t-1} - y_{f,t})^2} \\
v_{\text{cost},t} &= ||\mathbf{v}_{\text{cmd},t}||_2 = \sqrt{|V_x|_t^2 + |V_y|_t^2} \\
R_{\text{cont},t} &= -\frac{d_{\text{move},t}}{50} - \frac{v_{\text{cost},t}}{1400}
\end{aligned}
$$

These penalties encourage the agent to command smaller movements and lower speeds unless necessary for task completion.

K Training Architecture Details

This section provides specifics on the neural network architectures, hyperparameters, and training procedure used for the PPO agent, as mentioned in Section 2.4.4.

K.1 Network Architectures

Let obs_dim be the dimensionality of the observation space and action_dim be the dimensionality of the action space (here, action_dim = 4 corresponding to $(x_f, y_f, |V_x|, |V_y|)$).

Policy Network (Actor) The policy network is implemented as a sequential MLP followed by a custom parameter extractor and wrapped for probabilistic action sampling.

1. **MLP Backbone**:

```
nn.Linear(obs_dim, 128), nn.ReLU(),
nn.Linear(128, 64), nn.ReLU(),
nn.Linear(64, 64), nn.ReLU(),
nn.Linear(64, 32), nn.ReLU(),
nn.Linear(32, 32), nn.ReLU(),
nn.Linear(32, action_dim * 2) # Output: raw loc and scale params
```

2. **Parameter Extractor** (ScaledNormalParamExtractor): This module takes the action_dim * 2 output from the MLP, splits it into location (loc) and raw scale (scale_raw) vectors (each of size action_dim), and transforms the raw scale as follows:

   $$\text{scale} = (1 + e^{-\text{scale\_raw}})^{-1} - 0.5 + 0.001$$

   This transformation bounds the standard deviation parameter fed into the ‘TanhNormal‘ distribution, preventing excessively large or small exploration noise.

3. **Probabilistic Actor Wrapper**: The MLP and extractor are wrapped using ‘torchrl‘‘s ‘TensorDictModule‘ and ‘ProbabilisticActor‘.

   - Input key: ‘”observation”‘.
   - Intermediate keys: ‘”loc”‘, ‘”scale”‘.
   - Distribution: ‘TanhNormal‘. Samples from $\mathcal{N}(\text{loc}, \text{scale})$ are passed through $\tanh()$ and then linearly scaled to the desired action range.
   - Distribution Bounds:
     - ‘low = [mallet_r, mallet_r, 0.3, 0.3]‘
     - ‘high = [x_bound/2 - mallet_r, y_bound - mallet_r, 10, 10]‘

     These correspond to the minimum and maximum allowed values for $(x_f, y_f, |V_x|, |V_y|)$, respectively.
   - Returns log probability: True.

Value Network (Critic) The value network uses the same MLP backbone structure as the policy network, but with a different final layer to output a single scalar value.

1. **MLP Backbone**:

```
nn.Linear(obs_dim, 128), nn.ReLU(),
nn.Linear(128, 64), nn.ReLU(),
nn.Linear(64, 64), nn.ReLU(),
nn.Linear(64, 32), nn.ReLU(),
nn.Linear(32, 32), nn.ReLU(),
nn.Linear(32, 1) # Output: single state value estimate
```

2. **Wrapper**: Wrapped using ‘torchrl‘’s ‘ValueOperator‘.

Parameter Count The total number of trainable parameters across both the actor and critic networks is approximately 30,000.

K.2 Hyperparameters and Training Loop

Key Hyperparameters The following hyperparameters were used for the PPO algorithm and GAE:

- Discount Factor (γ): 0.99
- GAE Parameter (λ): 0.5
- PPO Clipping Epsilon (ϵ): 0.03
- Entropy Coefficient: 0.0003
- Learning Rate (Adam Optimizer): 3e-4

The low values for λ, ϵ, and the entropy coefficient were chosen to enhance training stability.

Training Procedure

- **Parallel Environments**: $N_{\text{env}} = 2048$ simulations run in parallel.
- **Action Period**: Time between consecutive agent actions in simulation $T_s = 0.1$ seconds.
- **Experience Collection**: In each iteration, collect $N_{\text{collect}} = 50$ steps of experience from each parallel environment (total $2048 \times 50 = 102,400$ transitions).

- **Replay Buffer**: Store collected experience in a replay buffer of size 200,000 transitions.
- **Training Updates**: After each collection phase, perform $N_{\text{train}} = 250$ gradient update steps using batches sampled uniformly from the replay buffer with a batch size of 1024.
- **Data Augmentation**: For each transition (s, a, r, s') sampled from the buffer, generate a second symmetric transition $(\text{flip}(s), \text{flip}(a), r, \text{flip}(s'))$ by flipping relevant state and action components along the table's central y-axis (e.g., $y \leftrightarrow W - y$, $v_y \leftrightarrow -v_y$, where W is table width). This doubles the effective data available for training.
- **Simulation Enhancement**: Random impulses were occasionally applied to the puck during simulation runs to prevent it from getting stuck and to encourage robustness against minor perturbations.

L Firmware

L.1 Host-Microcontroller Communication

Data Packet Structure The host computer prepares and sends data packets containing the necessary parameters for the Blue Pill to execute the next segment of the mallet's trajectory. These parameters are packed into a binary format using Python's `struct.pack` function with the format string `'<iiiiiiiiiiii?'`. This specifies little-endian byte order and packs the following data (refer to Appendix E.2 for the meaning behind the variables):

- `vt_1`: Two `int32` values, representing t_{x1} and t_{y1}.
- `vt_2`: Two `int32` values, representing t_{y1} and t_{y2}.
- `Vf`: Two `int32` values, representing the target signed velocity components (V_x, V_y) for the feedforward control.
- `C2`, `C3`, `C4`: Six `int32` values (two for each constant).
- `sum`: One `int32` value, containing the sum of all preceding integer values in the packet, used for error checking.
- `idle`: One boolean value (`'?'`), indicating whether the robot should enter an idle state (True) or remain active (False).

Data Type Conversion Many of the control parameters are naturally floating-point numbers. To transmit them efficiently using integer formats, they are scaled before packing. `vt_1`, `vt_2`, and `Vf` are multiplied by $10,000$, while the constants `C2`, `C3`, `C4` are multiplied by a larger factor of $100,000,000$. These scaled values are then cast to `int32` and packed. The firmware on the Blue Pill receives these integers and divides them by the corresponding scaling factors to recover the approximate floating-point values needed for its calculations.

Framing and Error Checking Each packed binary data structure is framed by newline characters (\n). The host sends \n + `packed_data` + \n. The Blue Pill firmware waits until its serial receive buffer contains a sufficient number of bytes. It then reads the first character, verifying it is a newline. If confirmed, it continues reading bytes until the next newline character is encountered, extracting the binary data payload.

To ensure data integrity and detect potential corruption during transmission, a simple checksum mechanism is employed. The firmware calculates the sum of the first 12 received integer values (`vt_1` through `C4`) and compares it against the 13th integer value (`sum`) transmitted in the packet. If the sums do not match, the received packet is discarded, preventing the use of corrupted motion commands.

Firmware Functionality Upon receiving a valid data packet, the firmware uses the unpacked and reconstructed parameters to execute the desired mallet movement using its internal feedforward control logic (refer to Appendix E.2). The `idle` flag allows the host computer to command the robot to cease active movement.

L.2 Calibration and Homing

Coordinate Transformation The relationship between Cartesian coordinates and encoder angles is assumed to follow the kinematic model:

$$x = (\theta_0 - \theta_1) \times \frac{\texttt{PULLEY\_RADIUS} \times \pi}{360} \tag{37}$$

$$y = (-\theta_0 - \theta_1) \times \frac{\texttt{PULLEY\_RADIUS} \times \pi}{360} \tag{38}$$

where θ_0 and θ_1 are the true encoder angles in degrees (measured angle + offset).

Calibration Challenge and Jig Let these known distances provided by the jig be `wall_0` (offset from the y-axis) and `wall_1` (offset from the x-axis). When using the jig in a corner, the mallet's center is effectively located at $(x, y) = (\texttt{mallet\_r}+\texttt{wall\_0}, \texttt{mallet\_r}+\texttt{wall\_1})$, where `mallet_r` is the radius of the mallet.

Two-Point Measurement Procedure

1. **Right Corner Measurement:** The mallet is placed using the jig in one corner (e.g., the "right" corner, assumed to correspond to the known position $(x_R, y_R) = (\texttt{mallet\_r} + \texttt{wall\_0}, \texttt{mallet\_r} + \texttt{wall\_1})$). The measured encoder angles $(\texttt{right\_point}_0, \texttt{right\_point}_1)$ are recorded.
2. **Left Corner Measurement:** The mallet is placed using the jig in the opposite corner (e.g., the "left" corner, along the same y axis). Assuming this corresponds to a known position involving the table width (denoted as `width`), such as $(x_L, y_L) = (\texttt{mallet\_r} + \texttt{wall\_0}, \texttt{width} - \texttt{mallet\_r} - \texttt{wall\_1})$ or similar, the measured encoder angles $(\texttt{left\_point}_0, \texttt{left\_point}_1)$ are recorded.

Solving for Parameters Substituting the known coordinates and measured angles (plus the unknown offsets, `offset_0`, `offset_1`) into the coordinate transformation equations (37-38) yields a system of equations. Based on the provided derivation, the relevant equations using the measured points are:

$$\begin{aligned}
\texttt{mallet_r} + \texttt{wall_0} &= ((\texttt{right_point}_0 + \texttt{offset}_0) - (\texttt{right_point}_1 + \texttt{offset}_1)) \\
&\quad \times \frac{\texttt{PULLEY_RADIUS} \times \pi}{360} \\
\texttt{mallet_r} + \texttt{wall_1} &= -((\texttt{right_point}_0 + \texttt{offset}_0) + (\texttt{right_point}_1 + \texttt{offset}_1)) \\
&\quad \times \frac{\texttt{PULLEY_RADIUS} \times \pi}{360} \\
\texttt{width} - \texttt{mallet_r} - \texttt{wall_1} &= -((\texttt{left_point}_0 + \texttt{offset}_0) + (\texttt{left_point}_1 + \texttt{offset}_1)) \\
&\quad \times \frac{\texttt{PULLEY_RADIUS} \times \pi}{360}
\end{aligned}$$

Solving this system of three equations for the three unknowns (`offset_0`, `offset_1`, `PULLEY_RADIUS`) yields the following results:

$$\texttt{PULLEY\_RADIUS} = \frac{360}{\pi} \times \frac{\texttt{width} - 2(\texttt{mallet\_r} + \texttt{wall\_1})}{\texttt{right\_point}_0 + \texttt{right\_point}_1 - \texttt{left\_point}_0 - \texttt{left\_point}_1} \tag{39}$$

$$\texttt{offset}_1 = \frac{1}{-2}\left(\frac{360 \times (2 \times \texttt{mallet\_r} + \texttt{wall\_0} + \texttt{wall\_1})}{\texttt{PULLEY\_RADIUS} \times \pi} + 2 \times \texttt{right\_point}_1\right) \tag{40}$$

$$\texttt{offset}_0 = \frac{1}{2}\left(\frac{360 \times (\texttt{wall\_0} - \texttt{wall\_1})}{\texttt{PULLEY\_RADIUS} \times \pi} - 2 \times \texttt{right\_point}_0\right) \tag{41}$$

These calculated values for `PULLEY_RADIUS`, `offset_0`, and `offset_1` are stored and used by the firmware for all subsequent coordinate transformations during operation.