Training a Robot to Juggle an Unconstrained Ping Pong Ball

Travis Clauson, Srisharan Kolige, & Selina Spry

1 Introduction

Have you ever tried bouncing a ping pong ball on a paddle and counted how many times it would bounce before you lost control of it? Typically, when one attempts this task for the first time they are not very good at bouncing more than a couple of times. However, with practice humans are able to adapt and learn from their previous experiences to correct and adjust to try a different technique or skill to accomplish. Since this task is highly dexterous, this team was motivated to investigate this same phenomenon with robots. Using a similar concept, the team tasked the robot to learn how to bounce a ball on its end effector.

With this COMP 141 Probabilistic Robotics Final Project, the team sought to develop a solution for bouncing a ball on a platform in simulation. In a real-world environment, when a ball bounces on a hard surface, the successive bounces are physically distinct, and its speed reduces with every bounce. However, in a hypothetical world where there were to be an idealized perfectly elastic ball, the ball would not come to rest after a finite number of bounces [3]. Therefore, in simulation the robot would not necessarily experience the same physical characteristics of a real-world ball and the effects of even small fluctuations in the environment will eventually cause the real-world ball to stop bouncing. Given that this task would be more idealized in the simulation world where the variables could be more tightly controlled, the team decided to execute this robotics project in simulation.

2 Background Related Work

If one has played table tennis before, they have likely attempted to juggle the ball with their paddle. While it is an easy action for most people, developing this level of motor skill in a robot is difficult. This project attempts to simulate a robot juggling the ball using the simulation visualization package PyBullet.

2.1 Ball Physics

Ping pong balls are hollow, composed of plastic-like material that combines camphor and nitrocellulose (”Celluloid”). Ping balls have a coefficient of restitution ranging from 0.89 to 0.92 [1]. The coefficient of restitution refers to the proportion of an object's velocity before and after an impact — primarily utilized in quantifying an object's "bounce-ability" or elasticity [7].. Therefore, the ball will lose approximately 10% of its velocity or 19% of its kinetic energy with each bounce. The robotic solution must compensate for this energy loss with an upward strike on the ball.

On top of adding energy to the ball, the robot must keep the ball bouncing straight up and straight down. Slight fluctuations in air pressure, imperfections in the ball, noise in the sensors, and noise in motor movements will make this problem more complex than simply striking up on the ball each time. The system needs to be able to track the ball's position and react by outputting motor movements that direct the ball toward the middle of the paddle or surface. Once this control loop is perfect, the robot simulation should be able to bounce the ball forever.

2.2 Reinforcement Learning

Reinforcement learning is a category of machine learning and aims to "provide a potential optimal strategy in one or more schemas," [12]. The schema or environment will then influence the system via feedback. Reinforcement learning is a way of learning by mapping situations to actions that will help to maximize its reward signal rather than trying to understand the hidden structure like other machine learning paradigms [11]. The agent doing that learning is not aware of what actions to take, instead it must discover what actions will yield the higher reward. Therefore, reinforcement learning is primarily distinguished by its trial-and-error search and its delayed reward that serves as feedback to the system [11].

In general, reinforcement learning is a "simplified way of implementing a process that improves machine performance with time." [8] It is an approach that consists of 3 main parts: the agent, rewards, and punishments. The agent is an intelligent program that is in an environment where it must constantly adapt and maximize points based on feedback that is either positive (rewards) or negative (punishments). This feedback is commonly referred to as the reward system. Reinforcement learning relies on its environment to reach its goal by obtaining information to learn from and act accordingly. The process is very much based on trial and error because predicting which actions to take is very difficult.

Proximal Policy Optimization, or PPO, is a model-free reinforcement learning policy that is typically used to improve the agent's training stability. It is able to accomplish the task by purposefully avoiding updates to the policy that are too large [10]. In doing so, the smaller policy updates during the agent's training are more likely to converge to an optimal solution. In addition, the smaller policy updates can reduce the possibility of the agent entering scenarios where it either takes a long time or is sometimes no longer possible to recover from the state. The PPO updates the policy conservatively, increasing its chances of completing the training. Furthermore, with the clipped objective, the PPO implementation uses two policy networks: one of which contains the policy to refine, and the other is the policy that includes the last used samples [6]. Using importance sampling, the new policy with samples collected from an older policy can help improve sample efficiency [6]. Thus, the new objective function would be as follows:

$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \underset{\tau\sim\pi_k}{\mathrm{E}}\left[\sum_{t=0}^{T}\left[\min\left(r_t(\theta)\hat{A}_t^{\pi_k}, \mathrm{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{\pi_k}\right)\right]\right]$$

Figure 1: Objective function implementing importance sampling in PPO.

2.3 Related Work

Previously, research in a similar application has investigated policies in reinforcement learning that have maximized the capability of the robot to return a table tennis ball [5]. The research outlines that when generating the robot's joint trajectories in response to the ball trajectory, the system should have a mapping that helps to predict the trajectory based on a collection of measured initial ball positions. The system can optimize the robot's joint movement within a desired cycle time by leveraging just a portion of the trajectory readings. Similarly, in this project, it will be imperative for the system to understand the trajectory of the ball as it is in flight to ensure the robot is properly positioned ahead of the ball bounce. Furthermore, in the work of D'Andrea et al. [9], their objective is to bounce an unconstrained ball forever without any sensors. They use a slightly concave parabolic shape which redirects the ball toward the center. In addition, a decelerating paddle motion at impact creates a consistent apex height on every bounce.

Another research project emphasized that with the help of reinforcement learning, finding inverse kinematic solutions for robots can be more streamlined and allow the robot to determine the optimal solution in a shorter time [13]. In particular, the research by Zhang et al. highlights the use of an improved PPO or Proximal Policy Optimization algorithm to address the problem of sparse rewards in deep reinforcement learning applications. The researchers obtained results by building the simulation environment, obtaining a network about the Actor and Critic, designing a hierarchical reward function, and comparing the improved PPO to the traditional PPO algorithm. Ultimately, the improved PPO algorithm was determined to have improved convergence speed and operating accuracy over the traditional PPO algorithm [13]. The improved PPO, along with the hierarchical reward function, significantly increased the performance of the system. For instance, Zhang et al. demonstrate that across just 7 cycles of training, there was a gradual increase in reward value in the system with the hierarchical reward function.

Additionally, this observation was able to demonstrate that this allowed the system to converge faster and the overall training effect is better. The hierarchical reward function divides the rewards for successfully reaching the target position into steps. This allows the robot in the environment to receive more helpful feedback through the entire sequence of decision-making.

3 Technical Approach

To teach an articulated robot to bounce a ball in simulation, the team decided to implement a Proximal Policy Optimization reinforcement learning algorithm. Specifically, the team used Pybullet for the simulation environment, Open AI gym for the RL environment, and an open-source PPO Algorithm for the learning agent.

One of the main advantages of carrying out this project in a simulation was that the project team could manage external forces and variables. For instance, the team started with more idealized situations and later introduced other factors that change the environment, such as balls with different coefficients of restitutions. By doing so, it was easier for the team to identify the impacts of the variables on the trained model.

3.1 The Customized Robot

The team needed a robot arm with 7 degrees of freedom (DOF) so that it could reach any point in a 3D space. We considered many different robot arms, including the Kinova and Universal Robot arms, but ultimately decided on the Franka Panda arm.

The Franka Panda arm comes with two separate URDF files: panda_arm and panda_hand. The panda_arm file contains the robot arm model, including 8 links and all their joints. The panda_hand file contains 2 links: the hand and a gripper as an end-effector. The two files can be combined to create a complete robot assembly.

The project required a robot arm with a flat plate or platform as its end-effector so that it could bounce a ping-pong ball. We were lucky enough to find the meshes and the .urdf files for a frying pan in the Pybullet library and decided to use the frying pan as the end-effector. To do this, the team had to edit the .xacro files of the robot hand to change the end effector from grippers to a frying pan. The team then used this to generate the .urdf file, which was attached back to the robot assembly.

3.2 The Environment

The team used AI Gym to initialize the custom environment. In doing so, the team had to identify the action space that would define the actions that the robot would perform in the environment. These are the parameters that control the environment, which in this case are the robot and its end effector.

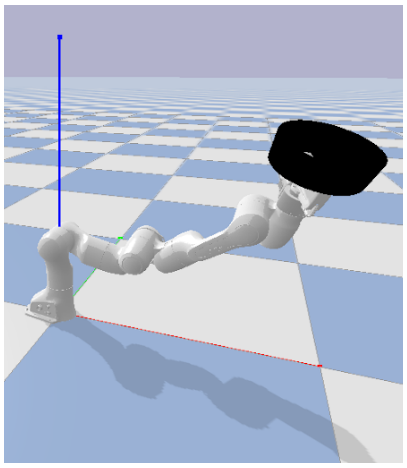

Figure 2: Graphical rendering of the team's Pybullet simulation setup.

The action space covered the robot's end effector x-y position as well as its orientation in roll, pitch, and yaw directions. Therefore, the team created a 5-dimensional action space. Meanwhile, the observation space, which observes how the state of the environment changes is a 9 dimension space in the project. The dimensions of the observation space included the robot end effector's x-y-z position, the ball's x-y-z positions, and the ball's x-y-z velocities. These values are then updated each timestep.

In this project, the robot's goal was to bounce a ping pong ball on its end effector. To effectively do that, the agent must produce a policy that is capable of navigating the robot towards its goal and this policy is guided by the rewards that it receives from the environment. The team leveraged a hierarchical dense reward system which was initialized in the environment to aid the robot in reaching its goal. The team determined that if the robot's end effector comes within 0.2 meters of the robot's x-y position, the reward would be inversely proportional to that distance. In addition, the robot was able to gain an additional 10 points for each successful strike of the ball with the pan. Thus, the reward system was as follows:

$$d = \sqrt{(robot_x - ball_x)^2 + (robot_y - ball_y)^2}$$

$$reward = \begin{cases} 1 - d, & d < 0.2m \\ 10, & ball\ bounces\ on\ pan \end{cases}$$

Figure 3: The teams functions for distance and dense and sparse rewards.

Although the team initially intended to implement a binary reward system due to its simplicity in computation, the team quickly realized it would take much longer for the robot to find a solution. Hence, a hierarchical dense reward system was identified to be the most optimal reward system for this application. A small reward is offered for keeping the pan underneath the ball, which is helpful in the agent's early learning when it may struggle to find larger rewards from making contact with the ball. However, once that overall exploration decreases and the robot is able to achieve a more stable bounce, the higher 10-point reward becomes more beneficial to the system's training.

3.3 The Agent

Given that this team was not aiming to develop a completely new policy model, the team used the PPO for Beginners Tutorial as a basis for implementing PPO as the agent for our environment (Yu). The agent was responsible for performing actions in the environment based on the feedback from the environment seen as rewards. Throughout the training process, the PPO agent produces actor and critic models that are periodically saved for continued training or testing.

3.4 Simulations

The team used PyBullet to perform simulations and tests. Pybullet is a "fast and easy-to-use Python module for robotics simulation and machine learning" [2]. Since this simulation platform is focused on robotics simulation and machine learning, the team believed that this platform would help reduce complications with implementing our project in an environment that would potentially not be optimized for our application. For instance, one of the features of PyBullet that our team would greatly benefit from this physics simulation environment is the ease of use and integration with the Open AI Gymnasium, which is an API capable of representing general reinforcement learning problems [4].

4 Evaluation Methodology

First, the team compared the agent's average rewards over the course of training. This demonstrates how the agent's ability to gather rewards changed from batch to batch. To reiterate, the training consisted of episodes that took approximately 100-200 time steps, 10,000 timesteps per batch, and 100 total batches for approximately 1,000,000 total time steps.

The team also measured the average rewards of the trained agent across different environments to understand the robustness of the agent. In these alternate environments, the team edited the coefficient of restitution, or the elasticity, of the ball from 1.0 down to .9 and .,8, respectively. The average rewards for each environment were measured across five batches or 50,000 time steps.

Finally, the team looked to evaluate the agent's performance at accomplishing the primary goal of the experiment: to juggle the ball as many times as possible. Hence, the most explanatory metric for the success of the experiment is to compare the robot's average number of consecutive hits before and after training. Specifically, the team compared the average hits per episode in the first and hundredth (final) batches.

5 Results

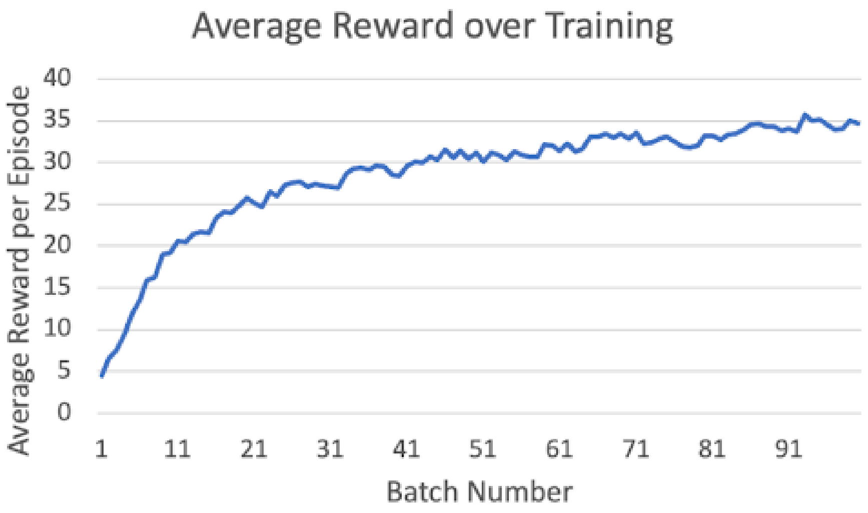

Figure 4: Line plot illustrating the agent's average reward over its 100 batches of training.

Figure 4 shows that the agent continuously improved over the course of training but that it was converging to an asymptote of maximum reward. When the team tested the trained agent in the graphical environment, it appeared that the robot specifically converged to a behavior where it used the pan's side to hit the ball. This behavior was able to guarantee a second strike each episode but made it difficult to achieve further strikes. This convergence can likely be avoided in the future by constraining the pan to maximum absolute roll, pitch, and yaw angles.

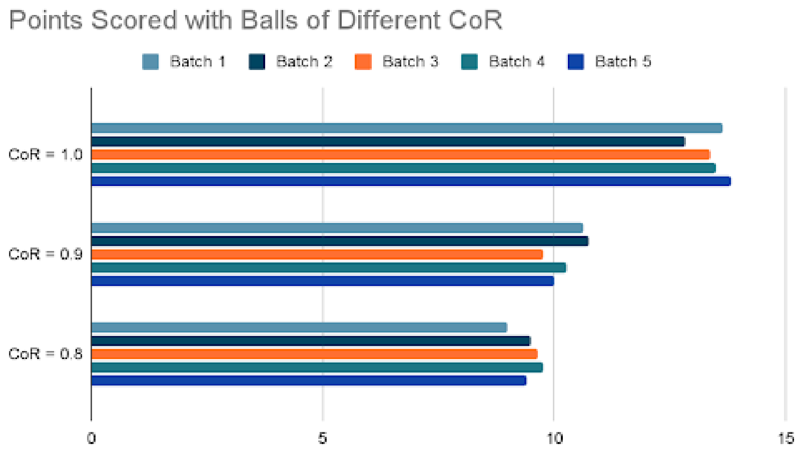

Figure 5: Clustered histogram demonstrating how the trained agent performs when the elasticity of the ball is dampened.

Next, Figure 5 illustrates that the agent averaged the most rewards in the environment in which it was trained, where the ball's coefficient of restitution, or elasticity, equals 1.0. The data shows that the average reward was lowest when the coefficient of restitution was the lowest. This aligned with the team's expectations that the average reward would be lowest in the environment that is most different from the training.

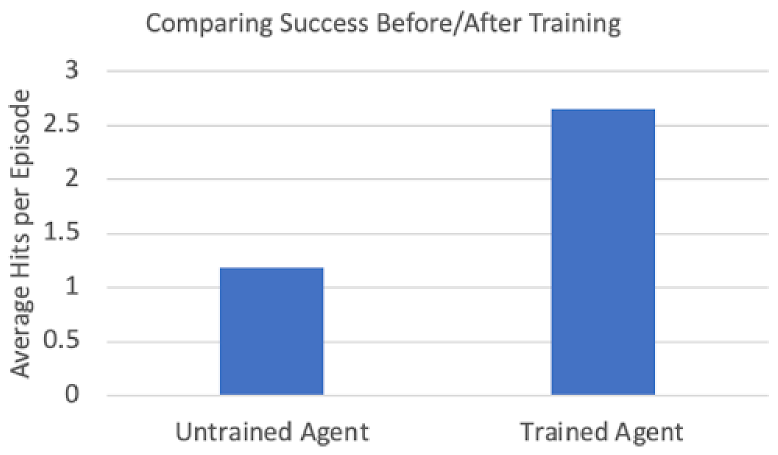

Figure 6: Binary histogram comparing the average number of hits per episode for the untrained and trained agent.

Finally, Figure 6 demonstrates that the trained agent performed significantly better at the final task than before training. Although this was expected, the team was very proud of this statistic. The fact that the final agent was consistently and statistically improved proved efficacy for the team's reinforcement learning agent. The future work section outlines how the team would change the agent to further improve its ability to learn to juggle.

6 Conclusion and Future Work

The team concluded the project with a robot that was able to bounce a ball consistently a few times with the custom end effector. The code developed and used in this project can be found in the team's GitHub repository[1]. Although the team was able to accomplish bouncing the ball at least a few times in the end, there are a number of areas that the team has identified as items to improve upon in future work.

For instance, the end effector should be upgraded to an actual paddle rather than a pan. As can be seen in the final demonstration, the final solution converged to a policy that sometimes used the side of the pan to bounce the ball rather than the center of the pan. Therefore, this phenomenon would be much less likely to occur with a paddle.

Next, as suggested by a couple of people experienced with reinforcement learning, the team would like to implement a new control approach. Instead of telling the agent to set an absolute goal position for the end effector, it should output a goal change in the position and orientation: ΔX, ΔY, ΔZ, and $\Delta\ \theta$, $\Delta\ \phi\ \Delta\ \psi$. This should help the agent move smoother and converge onto a more natural solution.

Additionally, the team would like to implement more realistic physics in the environment. For example, the coefficients of restitution of all objects would ideally be more realistic and resemble the real-world coefficients for a more life-like simulation. This would potentially include reducing the ball's coefficient of restitution to be closer to 0.9. By doing so, the robot would also need to learn how to keep bouncing the ball by potentially implementing an upward striking motion.

Lastly, another idea for future work is introducing variability of where the initial ball is spawned. Thus, the policy would be more robust for different scenarios. These are all avenues that would greatly benefit the project if further research were conducted.

[1]https://github.com/travisclauson/ProbabilisticRoboticsGroup

References

[1] *The Table: Technical Leaflet T1.* The International Table Tennis Federation, 2004.

[2] Erwin Coumans and Bai Yunfei. Bullet real-time physics simulation.

[3] John Earman and John Norton. Infinite pains: The trouble with supertasks. In Adam Morton and Stephen P. Stich, editors, *Benacerraf and His Critics*, pages 11–271. Blackwell, 1996.

[4] Farama Foundation. Gymnasium is a standard api for reinforcement learning, and a diverse collection of reference environments.

[5] Yapeng Gao, Jonas Tebbe, and Andreas Zell. Optimal stroke learning with policy gradient approach for robotic table tennis. *Applied Intelligence*, 10 2022.

[6] Jonathan Hui. Rl — proximal policy optimization (ppo) explained.

[7] Peter Merton McGinnis. *Biomechanics of sport and exercise.* Human Kinetics, Champaign, IL, 2nd ed. edition, 2005.

[8] Abhishek Nandy and Manisha Biswas. *Reinforcement Learning Basics*, pages 1–18. Apress, Berkeley, CA, 2018.

[9] Philipp Reist and Raffaello D'Andrea. Bouncing an unconstrained ball in three dimensions with a blind juggling robot. In *2009 IEEE International Conference on Robotics and Automation*, pages 1774–1781, 2009.

[10] Thomas Simonini. Proximal policy optimization (ppo).

[11] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, second edition, 2018.

[12] Nuo Xu. Understanding the reinforcement learning. *Journal of Physics: Conference Series*, 1207:012014, 2019.

[13] Zhizhuo Zhang and Change Zheng. Simulation of robotic arm grasping control based on proximal policy optimization algorithm. In *Journal of Physics: Conference Series*, volume 2203, page 012065. IOP Publishing, 2022.