Training a Robot to Juggle an Unconstrained Ping Pong Ball

Travis Clauson, Srisharan Kolige, & Selina Spry

1 Introduction

With this COMP 141 Probabilistic Robotics Final Project, the team seeks to develop a solution for infinite ball bouncing 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 [4]. However, that is an idealized phenomenon. In the real world, the physical characteristics of the ball and the effects of even small fluctuations in the environment will eventually cause the ball to stop bouncing.

The project team seeks to accomplish the task of juggling a ball in a physics-based simulation using a robot. The group will utilize algorithms and paradigms such as Kalman filters and reinforcement learning to accomplish this task.

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.

Ping pong balls are hollow, composed 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 "bounceability" or elasticity [7]. Therefore, with each bounce, the ball will lose approximately 10% of its velocity or 19% of its kinetic energy. 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.

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 [6]. 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. By leveraging just a portion of the trajectory readings, the system can optimize the robot's joint movement within a desired cycle time. 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.

3 Technical Approach

The team determined that implementing a Kalman filter and reinforcement learning in a simulation environment will solve the above mentioned problem. The team plans to test the project in a 2D simulation space in the early stages to simplify the problem. Since PyBullet is natively a 3D space, the team will accomplish this by constraining possible ball movement in either the X or Y dimension. The team will implement the third dimension of the ball movement once the 2D problem is perfected. The configurations described for both 2D and 3D environments are illustrated below in Figure 1.

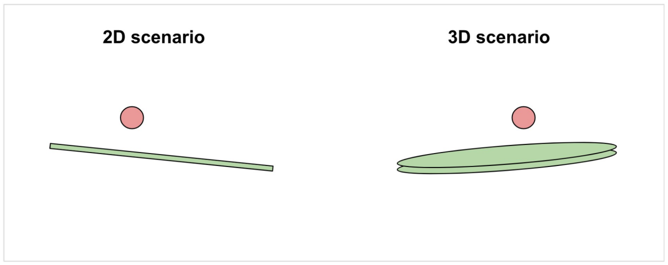
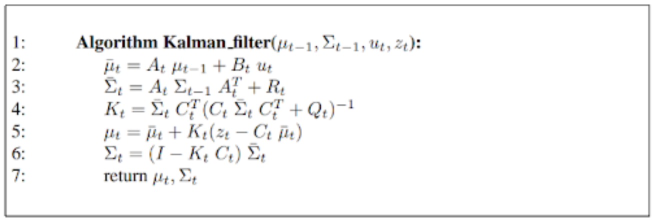


Figure 1: Ball bouncing task in 2D space (left) and 3D space (right)

One of the main advantages of carrying out this project in simulation is the project team is able to manage external forces and variables. For instance, the team will be able to start with more idealized situations and slowly introduce external forces such as noise due to air friction, and wind. The advantage of gradually introducing these external forces and variables is the effects of them can also be more closely monitored. Thus, it will be easier for the team to identify the impacts of the noise on the system. In addition, the simulation will allow the team to have more control over incrementally increasing the complexity of the system. Since the project will begin in a 2 dimensional space, the ball's movement will only need to be tracked along 2 axes.

3.1 Kalman Filter

As mentioned briefly above, the project will be utilizing a Kalman filter in conjunction with reinforcement learning. The Kalman filter will be used to track the movement of the ball as it bounces in the space above the platform. Kalman filters are one of the most common estimation algorithms [2] It is able to use past estimates to predict future system states. In addition, the predictions are able to filter through inaccurate and uncertain measurements [2]. The Kalman filter algorithm can be implemented following the fundamentals of the pseudocode below in Figure 2.



```
Algorithm Kalman_filter($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):
    $\bar{\mu}_t = A_t\, \mu_{t-1} + B_t\, u_t$
    $\bar{\Sigma}_t = A_t\, \Sigma_{t-1}\, A_t^T + R_t$
    $K_t = \bar{\Sigma}_t\, C_t^T (C_t\, \bar{\Sigma}_t\, C_t^T + Q_t)^{-1}$
    $\mu_t = \bar{\mu}_t + K_t(z_t - C_t\, \bar{\mu}_t)$
    $\Sigma_t = (I - K_t\, C_t)\, \bar{\Sigma}_t$
    return $\mu_t, \Sigma_t$
```

Figure 2: Pseudocode for Kalman Filter [10]

As can be seen, the Kalman filter requires inputs of: the previous state, the previous uncertainty, control input and measurement vector. In our implementation, the Kalman filter will be utilizing the previous states and uncertainties recursively from the system as expected. The control input would be null as there is no controlled input. However, the measurement vector would be the coordinates of the ball from the perspective of the camera(s).The output of the Kalman filter are the updated state and uncertainty matrices, which would represent the latest prediction of the ball's movement based on readings from the camera. Therefore, even if the camera had a lower frame rate or became disconnected temporarily the ball's position could still be predicted by the system.

Subsequently, the output of the Kalman filter will be supplied as part of the input into the reinforcement learning paradigm to properly actuate the appropriate motors to maximize the potential for the ball to keep bouncing indefinitely. 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 reward signal rather than trying to understand the hidden structure like other machine learning paradigms [11]. The agent doing that is 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 along with its delayed reward that serves as feedback to the system [11].

3.2 Reinforcement Learning

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 it is very difficult to predict which actions to take. In our case, the final project will implement reinforcement learning through the adaptive controllers of the actuators by adjusting the linear movement it produces based on the trajectory and state of the ball. Thus, the system will be optimizing the system to bounce the ball as many times as possible. Specifically, the team will use a model-free reinforcement learning algorithm such as Q-learning. The algorithm will have 2 input parameters: project location of the falling ball described in X and Y coordinates. On the other hand, it will have three output parameters, one for each degree of freedom of the paddle: roll, pitch, Z velocity.

The algorithm will leverage a binary reward system for its learning. Each successful strike of the ball results in a reward of one, while a miss of the ball will result in a reward of zero. A binary system works for this case because it is computationally simple and represents the team's goal well: bounce the ball as long as possible. One trade-off of the simple reward system is that the algorithm may take longer to find a solution than one with a complex or dense reward system.

3.3 Simulation

The team concluded that these simulations and tests would best be performed in PyBullet. Pybullet is a "fast and easy to use Python module for robotics simulation and machine learning," [3]. Since this simulation platform is focused on robotics simulation as well as machine learning, we believe that this platform would help to 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 [5].

4 Evaluation

The primary metric that will be used to measure the success of the experiment is the number of consecutive bounces above a threshold height until the ball falls off the platform. Similarly, time until the ball falls off could be used to measure success, but total bounces are favored because it is a discrete measurement. Furthermore, total bounces cannot be manipulated by the machine learning algorithm by bouncing the ball extremely high. The team will be evaluating the performance of our learning model by looking at the number of bounces before failure, after the algorithm was trained for 1000, 5000, 10000, and 20000 episodes (times) [13].

Since there is manufactured random noise in the simulations, there should be significant variability between trials. We will use N = 100 trials to obtain a meaningful evaluation unaffected by noise but not computationally burdensome.

Other metrics will be calculated for further evaluation and understanding. These include the mean and variance contact location (the error from the center) and mean bounce height. The mean hit height will demonstrate the algorithm's accuracy, while the variance will demonstrate the algorithm's precision. Below in Figure 3, is an illustration of a hypothetical mean and variance of contact location for a trial that scored ten bounces before the ball fell off the platform.

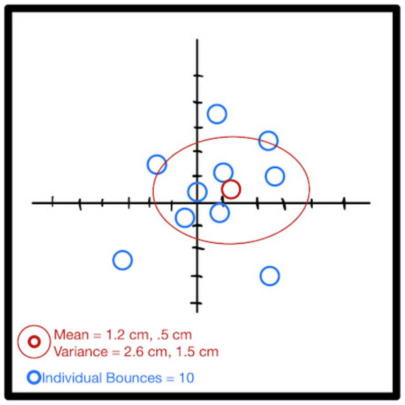


Figure 3: Illustration of a hypothetical mean and variance of contact locations of a bouncing ball on a platform

5 Timeline and Individual Responsibilities

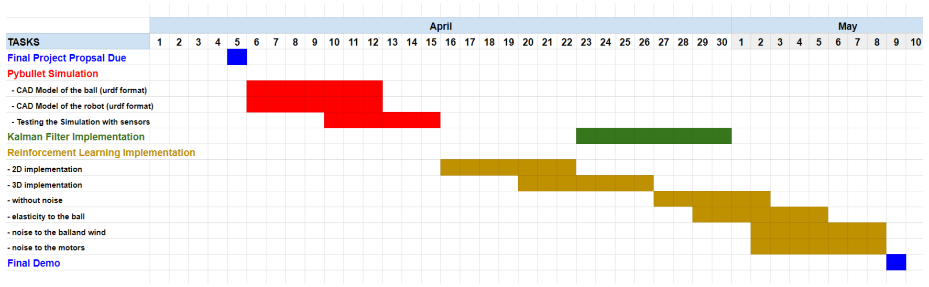


Figure 4: Gantt Chart of Final Project timeline

We have created a Gantt chart that outlines the timeline for our project, which is due in approximately one month. The chart specifies the allocated time for each task to be completed. The 3 major tasks in our project are listed below:

- the URDF files for our robot and ball
- the Kalman filter
- the Reinforcement Learning
- tasks are further divided into subtasks as represented in the Gantt Chart above, Table 1

In general, Srisharan will be mainly responsible for identifying the URDF filed required for our simulation as well as developing any objects or end effectors required for the project to be used in PyBullet. Travis will be primarily focused on developing the Kalman filter that will be used to track the ball's trajectory as well as helping to tune the reinforcement learning algorithm. Lastly, Selina will focus on developing the reinforcement learning algorithm to be applied to the robot system to learn how to juggle the ping pong ball. All members of the team will be partnering to integrate each piece of this project together.

References

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

[2] Alex Becker. Kalman filter tutorial.

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

[4] 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.

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

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

[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] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction.* MIT press, 2005.

[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] Yifeng Zhu, Yongsheng Zhao, Lisen Jin, Jun Wu, and Rong Xiong. Towards high level skill learning: Learn to return table tennis ball using monte-carlo based policy gradient method. pages 34–41, 08 2018.