Robot Learning Model to Play Ping Pong

Aadhithyavarman Manachai Shanmugham, Damian Padayachee, Boueny Folefack,
Abhinav Tiruveedhula, and Oscar Portillo

***Abstract*— This semester, we attempted to create a robot learning model that enables a Sawyer robot to autonomously play ping pong against a human opponent. The ability of robots to engage in dynamic and interactive activities such as sports has significant implications for human-robot interaction and collaborative robotics. Our model aims to integrate machine learning algorithms with real-time sensory data processing and motor control strategies to facilitate adaptive game play. Although we did not finish, we developed multiple algorithms to detect a ball's position in space, a faithful simulation, and a reinforcement learning algorithm template.**

I. INTRO

Our purpose for this model is to play ping pong against a human opponent. This will support the entertainment portion of the robot dinner party.

A. Main Goal

Our main goal is to develop a model that has at least a 70% success rate in detecting and returning a moving ball — a benchmark that falls reasonably in line with past work.[1] Such a success rate would mean that the model should be capable of maintaining a ball in play continuously with a human opponent for at least 2-3 consecutive returns.

B. Supplemental Goals

While not the main priority, some supplemental goals for the model that may be pursued include: extending the duration of play for 10+ consecutive rallies (requiring a successful return rate of $\approx 90\%$), implementing strategic decision-making in order to win rather than just keeping the ball in play, initiating the game by serving the ball autonomously, acting as an unbiased robot referee, and generalizing the model to other robots.

We were able to create two algorithms to detect the ball in space. These algorithms used the data input from an Intel RealSense D435 Depth Camera to return the x, y, and z coordinates of the ball.

We also created a faithful simulation of the Sawyer robot and table tennis system to be used for reinforcement learning in simulation. Using this, some final reinforcement learning in the real world would translate to a working ping-pong-playing product. While the restitution, friction, and dampening coefficients were estimates, they could be more accurately measured in the future to better fit the real-world system.

For reinforcement learning, we developed a Twin Delayed Deep Deterministic (TD3) policy gradient reinforcement learning (RL) architecture for our robot. While we were not able to train and fully integrate the system, our framework is only missing integration of ROS/Gazebo/MoveIt! through OpenAI's gym framework.

II. RELATED WORK

There has been an assortment of work in the field of table tennis robots, with some particularly exciting findings in the following four sections.

A. Existing Robots

Many robots exist to play table tennis, though they have varying designs. Some conventional arm-like robots, such as the KUKA robot, play in a human-like manner, exercising six or more degrees of freedom to swing full strokes.[2] Other models operate with a paddle fixed on rails with Three or fewer degrees of freedom, moving primarily along a plane either parallel or perpendicular to the table.[3] One of the more exciting examples we found was a quadrotor ping-pong drone which though quite novel, only managed to achieve about a 40% return rate.[4]

B. Accounting for Opponents and Ball Spin

Spin is a major component of ball trajectory and motion. Two leading approaches in determining spin have been a camera approach, where powerful cameras track features on the ball's surface,[5], and a physics approach, where we account for the Magnus Force, which is responsible for ball spin.[6]

C. Ball Detection Algorithms

Ball detection is integral to ball trajectory prediction and motion planning. The prevailing ball detection method involves implementing Convolutional Neural Networks, which create a "box" over the ball to signify the ball's position in the camera's image.[7]

D. Ball Reinforcement Learning

There are multiple approaches that researchers have pursued in creating an RL model for a ping pong robot. Typically, researchers have opted for a policy gradient approach rather than Q-Learning, as it is a better fit for continuous action spaces and varying robot/environment states. Some notable algorithms include Trust Region Policy Optimization (TRPO), Proximal Policy Optimization (PPO), Soft Actor-Critic (SAC), Deep Deterministic Policy Gradient (DDPG), and TD3.

III. METHOD

A. Vision and Tracking

Our ball detection is quite stable and capable of detecting and tracking the ping pong ball with great consistency. Because of the ball's fast velocity, the algorithm must have a very minimal runtime to get the most points and allow for calculations to occur. Because of this, we chose to use the OpenCV library rather than developing our own model. Using OpenCV2 and an Intel RealSense D435 Depth Camera, we were capable of producing X, Y, and Z coordinates of the ping pong ball with a refresh rate of mere milliseconds. The operating procedure of our ball detection method is quite simple. Upper and lower bound arrays of the desired BGR values are initialized. The program takes in both a color frame and a depth frame from the RealSense Camera. The frames are blurred so as to reduce noise from very small irregularities. The desired color of the ball is then masked so just the ball remains colored. OpenCV then uses the findContours() method in order to detect shapes within the color bounds and marks it as a ping pong ball. X, Y, and Z coordinates are then extracted from the center pixels of the contour (both the color and depth frame) and returned, allowing other methods to use them as needed. Our method shows satisfactory reliability in terms of accurately returning the correct coordinates. However, at times, the depth coordinate is not able to be returned properly and instead produces a 0, creating unneeded noise within our method. There are times at which other objects (such as a chair) are detected due to it having a similar color to the ball. However, this is a simple debug and just requires more calibration of the RGB arrays.

B. Gazebo and Physics Simulation

Our physics simulation was mostly stable. Many of the physical component models and their attributes were either taken or adapted from another group's work at Berkeley and integrated into our use case.[8] The paddle model was integrated into the existing Sawyer robot model, and the rest was separately modeled. We also created a ball spawning node for the simulation, which would delete the ball once it had fallen off of the table or when the node shuts down. The ball was spawned with varying velocities from the same position to emulate the effect of different serves. Unfortunately, the ball spawner utilizes a Gazebo-ROS Service that becomes defective after about 3-4 calls, leading the ball to freeze in the air, so the environment must be reset. Alternatively, if future Gazebo patches come out, we could proceed as intended.

C. MoveIt! Motion Planning

Our proposed training method of using reinforcement learning in a simulated environment necessitates not only accurate physical simulation but also the rapid and precise passing of motion plans into such a simulation. To fulfill this requirement, we decided to use MoveIt as our motion planner. Our motion planner works by passing motion plans into MoveIt, via either the RViz user interface or the moveit_commander python package. These plans are converted to joint effort values, which are passed into the simulation, and emulated as accurately as possible.

D. Reinforcement Learning

We decided to go with the TD3 policy gradient algorithm, as it had the highest return and accuracy rate in one of the foremost papers we found on stroke learning.[9]

Algorithms	ϵ_d	ϵ_h	success rate
TRPO	47.0cm	31.0cm	84.8%
PPO	44.2cm	30.8cm	87.1%
SAC	43.5cm	29.0cm	89.2%
DDPG	25.6cm	22.1cm	95.6%
DDPG+argmax	23.0cm	22.3cm	97.4%
DDPG+argmax+3D Q-value	21.3cm	21.7cm	97.9%
TD3	25.2cm	22.3cm	97.2%
TD3+argmax	22.2cm	21.2cm	97.7%
TD3+argmax+3D Q-value	20.3cm	21.2cm	98.5%

Fig. 1. Reference Table used to decide what RL architecture we would implement.

In particular, according to the paper, DDPG was more successful than other algorithms like TRPO, PPO, and SAC, and given the enhanced stability and convergence that TD3 provides over DDPG, TD3 was a natural choice. The paper's novelty is found in its 3D Q-Value, which is used to update the actor and critic networks; this approach is in contrast to traditional table tennis algorithms, which employ a normal one-dimensional Q-value. The 3D Q-Value involves three individual Q-values Q_x, Q_y and Q_z representing x-y deviations from the target landing position and height (z) deviations from the net height in the middle. This Q-value made a significant difference in the overall effectiveness of the algorithm, as reflected in the table.

The TD3 architecture involved an actor and two critics, which were updated during training. The inputs of the actor are the current state of the ball (linear/angular velocity, position, and spin) before outputting the action, or motion plan, the robot should take. The inputs of the critics are the state and action taken by the actor, and they output the Q-value that the actor uses in its mapping of the current state to the action it should take. The more conservative of the two Q-values the critics generate are taken in order to increase the stability of the robot's training. Similarly, the actor will have its activation function updated every other episode while the critic's activation functions will be updated every episode. This process will increase the stability of the Q-values used by the actor and will cause there to be fewer missteps to be taken while training and therefore increase training speed and accuracy.

During training, an overall reward for a given state-action pair is calculated using immediate reward (which would be defined by OpenAI gym and ROS/Gazebo integration) and discounted future reward. Then, a loss function is defined for the actor and critics to compare the overall reward with the immediate reward by taking their norms, which is then used to perform gradient descent on our actor and critic neural networks. When using Q-values to perform gradient descent, a replay buffer that stores past states, actions, and rewards

is used. It is randomly sampled in order to remove temporal bias, which can lead to poor-performing models in evaluation and practice.

In addition to algorithm specifics, there was also a choice made between using a one-stage algorithm to map the ball position directly to the motion plan of the robot, or a two-stage algorithm of first determining the position, velocity, and spin of the ball before mapping those to the robot's motion plan. We decided to go with the two-stage algorithm as it appeared to have more success in multiple papers that we read[9][10]. While we did not manage to complete this stage, it is a spot for future work on this project, and guidelines found in multiple papers can be used[2][11].

IV. EXPERIMENTS

To test our method, we performed various experiments on the different parts of our model.

A. Vision and Tracking

To test the effectiveness of our main ball-tracking algorithm, we set up an Intel RealSense camera in the Robot Manipulation Lab. By holding the ping pong ball in the frame of the camera, our algorithm senses where the ball is.

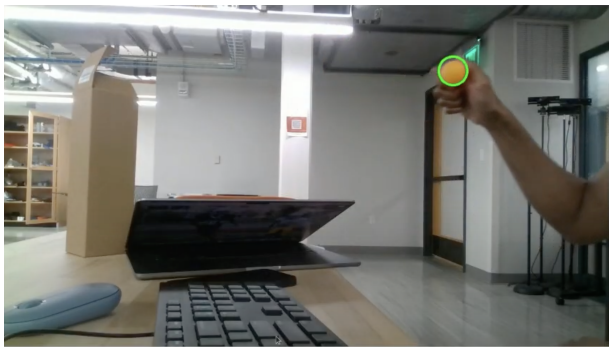

Fig. 2. Ball Tracking Algorithm

The above figure shows how the algorithm detects the ball in a 3D environment. Using the OpenCV library, we drew a circle on the frame to highlight the ball. Moving the ball manually, jerking it in an erratic manner, and bouncing it on the table yields similar results - the algorithm is able to return the X, Y, and Z of the center of the orange ball.

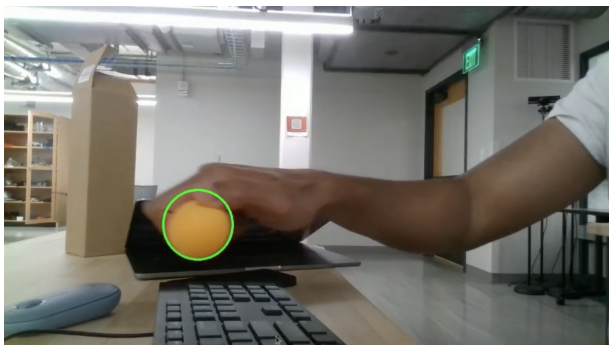

Fig. 3. Slightly Blocked Ball

In this figure, the ball is much closer to the camera, and our algorithm still detects it. The ball is also $\approx 10\%$ covered, which doesn't affect the output of coordinates. The algorithm still detects the ball if it is even closer to the screen.

In addition to detecting the ball by drawing a circle in the frame, the algorithm can detect what the X, Y, and Z of the ball are in space. This is necessary to supply the geographical location of the ball to plan the paddle movement.

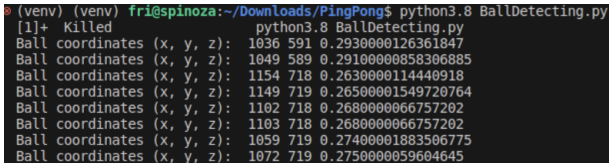

Fig. 4. Coordinates of Ball in Space

The above figure demonstrates our algorithm's ability to produce X, Y, and Z coordinates of the ball in space. The X and Y coordinates are in pixels, which can be easily translated into physical units when this algorithm is put into the Sawyer robot. The Z coordinate is in meters and is precise enough to allow calculations to be done.

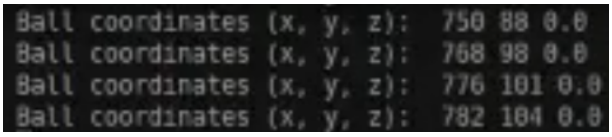

Fig. 5. Depth Perception Issue

One problem with our current algorithm is that $\approx 10\%$ of the time, it can't compute the depth coordinate. This is caused by the Intel RealSense camera not capturing the depth frame at the X and Y coordinates of the ball. As shown, this issue only presents itself when the ball is close to the edge of the frame. By placing the camera somewhere where the ball will stay in the center, this issue can be avoided altogether.

B. MoveIt! Motion Planning

In order to verify the accuracy of our motion plans as they were passed into the simulation, we elected for a visual observation comparing the RViz "ideal" final position to the actual final position generated by the gazebo simulation. As

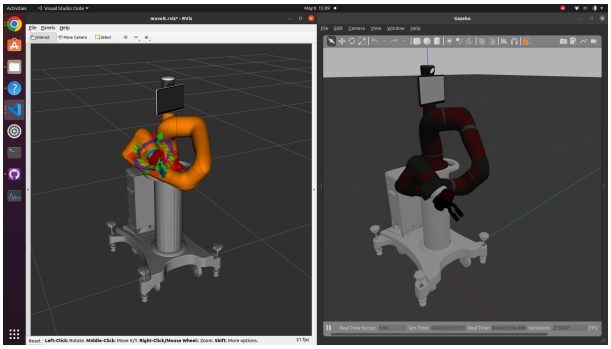

Fig. 6. RViz commands carrying over to Gazebo

is shown in Fig. 6, though the gazebo sim is able to closely emulate the desired arm position, there is a certain degree of ”sag” to the final position. Not only that, but when in motion the arm has a significant wobble to it, making its motion imprecise, and is therefore – in its current state – suboptimal for simulating ping-pong.

C. Gazebo and Physics Simulation

Since we were not able to test the simulation against real-life experimentation, we weren't able to determine the faithfulness of the model to the real world.

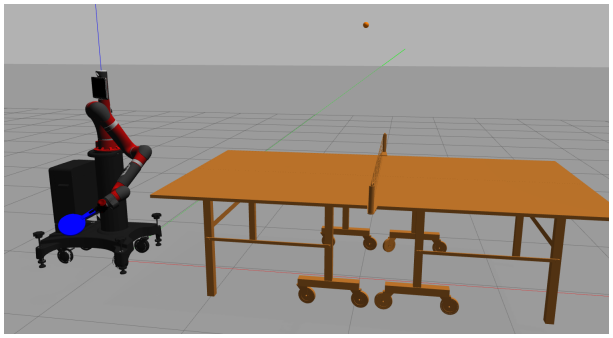

Fig. 7. An image of the simulated model of the robot holding the paddle while the ball falls to the table.

The way we would have evaluated the veracity of our model was by testing the restitution, friction, and dampening coefficients. We would find the x-y-z deviation in ball trajectory per stroke between the simulation and the real world so that proper adjustments to the simulation could be made and therefore facilitate the creation of a more accurate RL model.

D. Reinforcement Learning

While we were not able to train our model due to time constraints, theoretical evaluation of the model would rely on four core metrics. The model would be evaluated on the ball's distance from the target position, height above the net when passing over, and percentage of successful return strokes, and would additionally be punished for any invalid moves, such as the paddle colliding with the table or robot itself, or attempting to move the paddle outside of the arm's reach.

V. CONCLUSION

In all, our robot learning model to play ping pong resulted in algorithms to detect a ball in space, a simulation, and a reinforcement learning algorithm template.

A. Challenges

Like with any large-scale robotics project, we ran into a fair share of challenges.

1) Ball Tracking: With ball tracking specifically, we ran into a couple of issues. As we were using the Intel RealSense camera, we needed the RealSense package in our Python code. However, it failed to install on most of our personal devices along with the lab machine. After a lot of debugging, we found that the package doesn't work with Python 3.12, which is the version most of our devices were on. We ended up setting up a virtual environment on our machines, installing an older version of Python, and running our code there. Another issue we had was the program crashing when the ball left the frame. We figured out that there was a problem with how we were supplying the color and depth frame. We fixed this pretty quickly, and the ball detection algorithm started to work.

2) MoveIt! Motion Planning & Gazebo: By far the most difficult challenge when it came to integrating, ROS, MoveIt, and Gazebo was the lack of reliable documentation. Integrating three different environments and eliminating all errors that arose as a result of doing so took an inordinate amount of time away from the primary goals. In fact, often our most reliable information would come from decade-old forum posts, some of which could only be accessed through archives after their primary domains had been abandoned. Additionally, to reconcile the use of both MoveIt! and Gazebo we needed to use URDF (Universal Robot Description Format), as it was the only format supported on both; however, this is not the native format on Gazebo, which led to many limitations and complications in the process. Moreover, the debugger was rather poor and a simple mistake, such as leaving an empty, unnecessary tag, could cause the whole system to fail. Despite this, we are still satisfied with the progress we have made in both motion planning and simulation and feel we are ready to move into incorporating reinforcement learning into this part of our project.

3) Reinforcement Learning: The creation of the TD3 architecture was mainly inhibited by our ability to integrate the MoveIt and Gazebo with the OpenAI gym we wanted to use in simulation training. Beyond the issues mentioned in the previous section, we also initially struggled with how the relationship between the Q-values and the actual updates to the model itself related to each other as this architecture was a big takeoff from the versions of training algorithms we had been exposed to beforehand. Although we were not able to get OpenAI Gym set up and integrated with the rest of our simulation environment, we believe we were able to create an RL architecture that uses all the core functions that were mentioned in the papers we based our model around.

B. Future Goals

1) Ball Tracking: While our ball-tracking algorithm works with reasonable accuracy and speed, there is always room for improvement. The major aspect we plan to integrate is a learning model. There are a multitude of ways to approach this. However, the most ideal options would either be a YOLO or an SSD, due to the high reaction time that the robot needs to successfully play ping pong. Deciding which of the two to use requires additional research and

testing to determine which approach provides a sufficient balance of both accuracy and speed. The benefits of such a model would well be worth the effort, as it could potentially allow for accurate object detection, differentiation between similar objects, identification of optimal striking points, and reduction of noise provided by depth sensors.

2) MoveIt! Motion Planning & Gazebo: Although Gazebo's lack of maintenance caused no shortage of problems in the development of our ping-pong robot, we believe that its compatibility with the ROS ecosystem was still high enough for us to continue using the simulator. In order for proper training to occur with our simulation, we would need to fix the bug of the ball refusing to re-spawn after being de-spawned a couple of times as this would have prevented our program from training without constantly restarting the simulation. We would also have liked to take some real calculations of all the surfaces used in our simulation so that accurate friction and restitution coefficients could be used when training our model.

3) Reinforcement Learning: As we didn't get a chance to actually train our model due to issues mentioned in prior sections, we would have liked to fix all the problems surrounding the integration of the OpenAI gym with the rest of our simulation environment. Additionally, we would also implement ball trajectory prediction software for the first stage of our two-stage training. Finally, we would like to reach the point of re-training our model using the real robot so we can see how training using simulation and real life would differ.

C. Final Thoughts

Although we are all immensely disappointed that we did not reach the point of playing against the actual ping-pong robot, we feel proud that we both accomplished a huge amount of our project as well as learned and expanded on a wide range of topics in robotics. We believe that with a bit more time and perseverance, a fully functioning ping-pong robot is well within our reach.

REFERENCES

[1] S. Gomez-Gonzalez, Y. Nemmour, B. Schölkopf, and J. Peters, "Reliable real-time ball tracking for robot table tennis," *Robotics*, vol. 8, no. 4, 2019. [Online]. Available: https://www.mdpi.com/2218-6581/8/4/90

[2] J. Tebbe, Y. Gao, M. Sastre-Rienietz, and A. Zell, *A Table Tennis Robot System Using an Industrial KUKA Robot Arm: 40th German Conference, GCPR 2018, Stuttgart, Germany, October 9-12, 2018, Proceedings*, 02 2019, pp. 33–45.

[3] M. Matsushima, T. Hashimoto, M. Takeuchi, and F. Miyazaki, "A learning approach to robotic table tennis," *IEEE Transactions on Robotics*, vol. 21, no. 4, pp. 767–771, 2005.

[4] R. I. Popescu, M. Raison, G. M. Popescu, D. Saussié, and S. Achiche, "Design and development of a novel type of table tennis aerial robot player with tilting propellers," *Mechatronics*, vol. 74, p. 102483, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957415821000015

[5] T. Gossard, J. Tebbe, A. Ziegler, and A. Zell, "Spindoe: A ball spin estimation method for table tennis robot," 10 2023, pp. 5744–5750.

[6] Y. Huang, D. Xu, M. Tan, and H. Su, "Trajectory prediction of spinning ball for ping-pong player robot," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 3434–3439.

[7] H. Li, S. G. Ali, J. Zhang, B. Sheng, P. Li, Y. Jung, J. Wang, P. Yang, P. Lu, K. Muhammad, and L. Mao, "Video-Based Table Tennis Tracking and Trajectory Prediction Using Convolutional Neural Networks," *Fractals*, vol. 30, no. 5, pp. 2 240 156–587, Jan. 2022.

[8] A. Vangipuram and W. Loo, "table-tennis-robot," https://github.com/AVSurfer123/table-tennis-robot/tree/master, 2021.

[9] Y. Gao, J. Tebbe, and A. Zell, "Optimal stroke learning with policy gradient approach for robotic table tennis," *Applied Intelligence*, vol. 53, no. 11, pp. 13 309–13 322, Jun 2023. [Online]. Available: https://doi.org/10.1007/s10489-022-04131-w

[10] Y. Ji, Y. Mao, F. Suo, X. Hu, Y. Hou, and Y. Yuan, "Opponent hitting behavior prediction and ball location control for a table tennis robot," *Biomimetics*, vol. 8, no. 2, 2023. [Online]. Available: https://www.mdpi.com/2313-7673/8/2/229

[11] J. Tebbe, L. Klamt, Y. Gao, and A. Zell, "Spin detection in robotic table tennis *," 05 2020, pp. 9694–9700.