RallyBot: Exploring Physics-Based Approaches to Robotic Table Tennis

Chaitanya Ravuri
CSAIL
MIT
Cambridge, USA
cravuri@mit.edu

Dylan Zhou
SEAS
Harvard
Cambridge, USA
dylanzhou@college.harvard.edu

***Abstract*—Table tennis robots have been implemented by several research groups before, but many of those formulations rely heavily on reinforcement learning (RL), which achieves impressive results but is time- and resource-inefficient. However, many aspects of table tennis can be modeled with physics, and perhaps a more computationally efficient formulation of a table tennis robot could be implemented using a combination of physics modeling and reinforcement learning. In this paper, we present RallyBot, a purely physics-based tennis robot without any RL training. We test multiple physics-based approaches, including the mirror law, a custom PD controller, and built-in Drake functions, to see how far we can get with physics modeling alone. We show that with domain expertise and physics alone, we can still achieve reasonable rallies while also significantly reducing the number of learnable parameters. Videos are available at https://youtu.be/gNNZMLM_BrQ.**

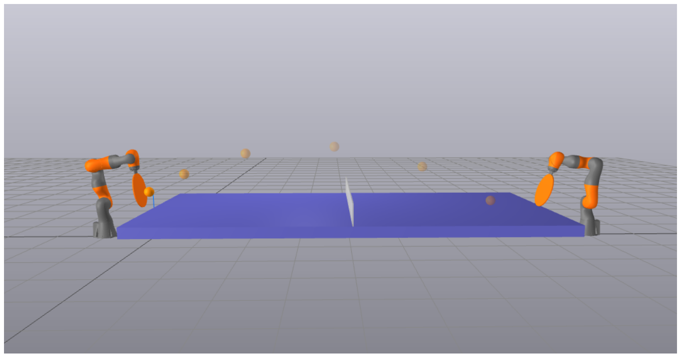


Fig. 1: **RallyBot** uses physics modeling to obtain the trajectory of the ping pong ball and hit it back to the other side of the table.

I. Introduction

Teaching a robot to play table tennis is a complex and challenging task. While traditional robotic manipulation tasks have leeway for hundreds of milliseconds of processing time, the game of table tennis distinguishes itself by a need for both *high-speed perception* to react to a fast-moving ball and a *high degree of precision* in the robot's joint angles, in how it holds the paddle, and in the motion of its stroke in order to hit the ball on a specific trajectory.

Despite these challenges, several research groups have already successfully implemented table tennis-playing robots [1] [2]. All of these implementations heavily exploit *reinforcement learning* (RL), a powerful general learning methodology that allows agents to solve problems in environments with all sorts of arbitrary conditions. In recent years, in addition to facilitating robotic table tennis, RL has achieved notable results in areas like autonomous navigation, robotic manipulation, and gameplay.

However, a big disadvantage of RL is that training takes a very long time and uses significant compute resources, as the learning agents typically must explore a large action space to determine what their optimal policy will be. For example, the RL-based chess-playing program AlphaGo Master trained for 800 hours to become good at chess [3]. This makes RL difficult to replicate and inaccessible to many groups who might not readily have these resources available.

For the specific task of table tennis, we envision a future where *domain expertise* of table tennis and *accurate predictions based on physics* can narrow the scope of the RL problem. That is, we believe it is possible to trade some degree of creativity from a completely RL-based pipeline for a more time- and resource-efficient hybrid pipeline based on a combination of physics and RL without sacrificing the accuracy of the robots' strokes or the length of their rallies. In this paper, we take a step towards this optimal solution and introduce RallyBot, a purely physics-based table tennis robot which we use to explore how successful we can be without exploiting RL methods.

At a high level, we use physics to obtain the trajectory of the ping pong ball and to compute the appropriate joint velocities in the robot arm for a desired trajectory of the paddle. We try three different approaches for our physics-based controller–implementing the mirror law, implementing a custom PD controller, and using built-in Drake functions–with each method improving upon the performance of the previous method. In each approach, we use our own expertise as table tennis players to manually select parameters that would result in a good hitting motion and maintain as long of a rally as possible (see Fig. 1).

We observe that our purely physics-based system, while not nearly as successful at playing table tennis as the RL-based systems, is still able to sustain a short rally but has significantly reduced the number of learnable parameters. And so the primary contribution of this paper is to show how we can significantly reduce the complexity of what needs to be

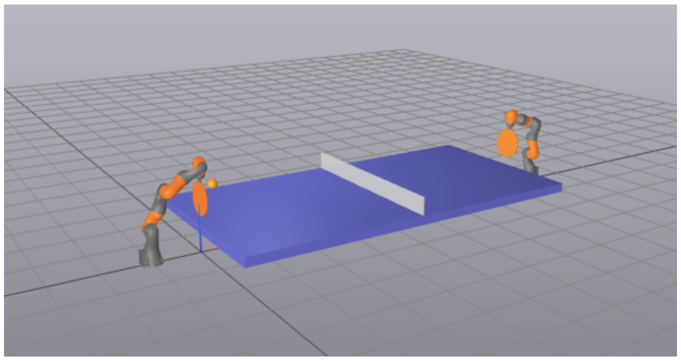


Fig. 2: **RallyBot simulation setup in Drake.** Comprised of 2 iiwa robots, a ping pong ball, 2 paddles, a table, and a net.

learned by RL for the task of robotic table tennis by handling many parts of the problem with physics and domain expertise.

Simplifying tasks using domain expertise and physics has implications beyond table tennis and could prove useful for faster, more efficient, and more accessible solutions to RL problems and learning problems more generally.

II. Related Work

A. RL-based table tennis robots

Prior work by W. Gao et al. [1] and Y. Gao et al. [2] are prime examples of RL-based table tennis robots. W. Gao et al. [1] implement a model-free algorithm to train a robot to play table tennis, and it uses a complete end-to-end reinforcement learning system that takes in the ball position and outputs joint velocities directly. Y. Gao et al. [2] also use an RL algorithm to train their robot. Like us, the authors of [2] recognize that there is a problem with the large amount of training time that RL algorithms usually require. To solve the issue of training time, the authors turn to realistic modeling via computer simulation in combination with training in the real world in order to teach their robot the optimal table tennis stroke in a reasonable amount of time.

In our approach, we take things several steps further. First, our robot is implemented completely in simulation, without any real-world interaction. Second, we impart our own domain knowledge onto the robot, implementing what we choose to be an optimal table tennis stroke for when it hits the ball. With this second step, we bypass the need for the robot to learn this part of the game, reducing the number of parameters that need to be optimized.

B. RL-based solutions to other games

Another paper by Silver et al. [3] describes a methodology in reinforcement learning which was previously used to achieve superhuman levels of play in games like chess and shogi. Specifically, self-play was found to do surprisingly well and performed better than methods that relied on human input. This paper inspired us to consider building a robot that could potentially achieve a high level of play in table tennis. It also gives us a sense of training times for other kinds of games.

C. Mirror law for robot juggling

A related problem to robotic table tennis is robotic juggling, where a robot tries to repeatedly bounce a ping pong ball vertically off a paddle as many times as it can in a row. One common algorithm used for this problem is known as the mirror law. The mirror law is a physics-based approach where the trajectory of the paddle simply mirrors the trajectory of the ball across a contact plane (up to a constant factor). Mirror law approaches to juggling were studied extensively by Rizzi and Koditschek [4].

Playing ping pong across a table is similar to juggling, so implementing a variation of the mirror law was our first attempt at RallyBot's physics-based controller.

III. Methods

For RallyBot, we use purely physics-based modeling to accomplish the task of playing table tennis. The task involves predicting the location of the ball at a given point in time and using the robotic arm to move the paddle to make contact with the ball at that time and in such a way that the ball bounces in the desired direction. A rally is comprised of repeating this action indefinitely.

We use physics to predict the ball's trajectory and physics to compute the joint velocities for the paddle's trajectory as the robot swings at the ball. We choose parameters for joint angles and velocities based on our own knowledge of table tennis to produce the longest rally possible.

We implement, train, and test our robot completely in simulation using the Drake software package. Within our simulation, we use two of the in-built iiwa robotic arms. Additionally, we create a mesh for the ping pong ball, the ping pong paddles held by the robotic arms, and the table surface (see Fig. 2). We specify the collision physics of our ball in such a way that it bounces realistically off the table and paddle surfaces.

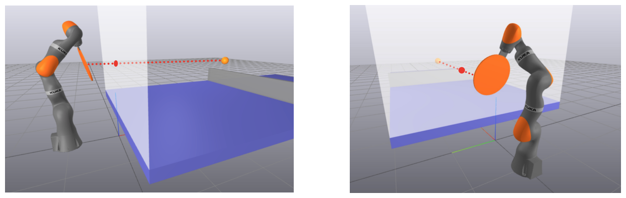
We try three different physics-based approaches: the mirror law, a custom PD controller with kinematic predictions for the ball, and Drake's built-in functions with kinematic predictions for the ball.

A. Mirror law

We first implement a mirror law controller in which the end effector mirrors the velocity and motion of the ball across the plane of contact in order to hit the ball. This mirroring is shown in Fig. 3, where the plane of contact is at the end of the table, or at $x = 0$.

As we are in simulation, we can get the ball's position directly. If its position is (x, y, z), then mirroring across the plane $x = 0$ would mean that we should move the paddle to $(-x, y, z)$. If the ball is too far from the plane of contact, it will likely be out of the range of the robot, and so we scale the x-coordinate first. We found that a $1/8$ scale allowed the desired location always to be within range.

We initialize the robot so that the center of the paddle is correctly mirroring the ball. Then, at each time step we do the following: Let the ball's current position be $p^B =$



(a) Side view. (b) Back view.

Fig. 3: **Side and back view of mirror law.** The paddle mirrors the ball's position across the $x = 0$ plane, scaled by a constant factor.

(p_x^B, p_y^B, p_z^B) and let the paddle's current position be p^P. We want the paddle to move to $(-p_x^B/8, p_y^B, p_z^B)$, so using a simple proportional controller, the paddle velocity should be:

$$v_x^P = K_p \cdot (-p_x^B/8 - p_x^P) \tag{1}$$
$$v_y^P = K_p \cdot (p_y^B - p_y^P) \tag{2}$$
$$v_z^P = K_p \cdot (p_y^B - p_z^P) \tag{3}$$

We then use differential inverse kinematics to get the joint velocities needed to achieve this paddle velocity. If J^P is the kinematic Jacobian for the paddle, then the joint velocities are:

$$v = [J^P]^+ v^P \tag{4}$$

where $[J^P]^+$ is the pseudo-inverse of the Jacobian.

While this method enables the robot to hit the ball, it is difficult to control where the ball goes, as the trajectory of the paddle always mirrors the trajectory of the ball. If the ball is going to the right, the paddle will also move right, and so we cannot hit the ball to the left. To give our physics model more flexibility, we attempt a different method that allows us to have greater control over the trajectory of the paddle.

B. Custom PD Control

Our second method involves predicting the trajectory of the ball and moving the paddle to intersect with that trajectory. Our algorithm can be described with the state diagram shown in Fig. 4. The robot starts in its ready state. In this state, the paddle is at the center of the table tilted downwards. The robot waits in this state until it detects the ball coming towards it. In simulation, we have access to the ball's velocity, so this transition occurs when the velocity in the x-direction is negative. At that point, we calculate where the ball will cross the plane of interest and move the paddle to a location just behind that spot.

We find the ball's trajectory using a simple model of projectile motion. If the ball's current location is (x_i, y_i, z_i) and the ball's current velocity is (v_x, v_y, v_z), then the ball's position after time t is:

$$x = x_i + v_x t \tag{5}$$
$$y = y_i + v_y t \tag{6}$$
$$z = z_i + v_z t - \frac{1}{2} g t^2 \tag{7}$$

The ball may bounce on the table zero or more times. Our model of bouncing is as follows: if the ball's z position is

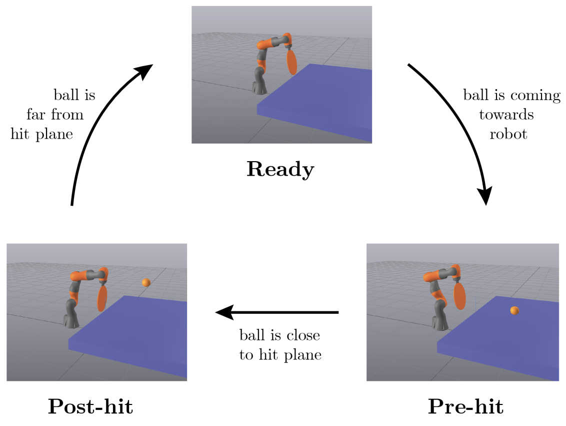

ball is far from hit plane
Ready
ball is coming towards robot
Post-hit
ball is close to hit plane
Pre-hit

Fig. 4: **State transition diagram for methods *B* and *C*.** The robot can be in one of three states corresponding to a position and rotation of the paddle. The transition between states is controlled through a custom PD controller in Sec. III-B and a built-in Drake function in Sec. III-C.

equal to the height of the table, we reflect the z velocity and multiply it by a constant. So, if the velocity before bounce is v_z, the velocity after bounce is $-kv_z$. We then continue using (7) with the new velocity. In our experiments, we set k to be 0.4, as we found that best approximated the true behavior of a ping pong ball.

We solve (5-7) when $x = 0$ to find the position of the ball when it crosses the end of the table. If this position is $(0, y_f, z_f)$, then the pre-hit position of the paddle is $(-0.1, y_f, z_f)$, so a bit behind where the ball will be hit.

The robot moves to this position, then waits until the ball nears the point of contact. We define "near" to be when the x position is less than 0.2, at which point the robot begins moving forwards to its post-hit position. As it is moving forward, it also tilts its paddle downwards so the ball has a flatter trajectory.

Finally, once the ball is sufficiently far away (again, we set this point to be when $x = 0.2$), the robot moves back to its ready state and waits until the next hit.

We again use differential inverse kinematics combined with a PD controller for the paddle trajectory to get the joint velocities. Unlike with the mirror law case, we need to control both the position and angle of the paddle. To get the angular velocity of the paddle, we work directly in Euler angles. Our PD controller looks like:

$$\omega^P = K_p(\theta_d^P - \theta^P) + K_d(-\dot{\theta}^P) \tag{8}$$

where ω^P is the desired angular velocity of the paddle, θ^P is the current rotation of the paddle (in Euler angles), and θ_d^P is the desired rotation of the paddle.

The problem with this method is that it is susceptible to gimbal lock, where one of the degrees of freedom is lost, and certain angles cannot be reached. We need our robot to have full control over its angular rotation, as the most minute changes in angle make a big difference in trajectory. So, to gain this rotational control, we modify our controller slightly.

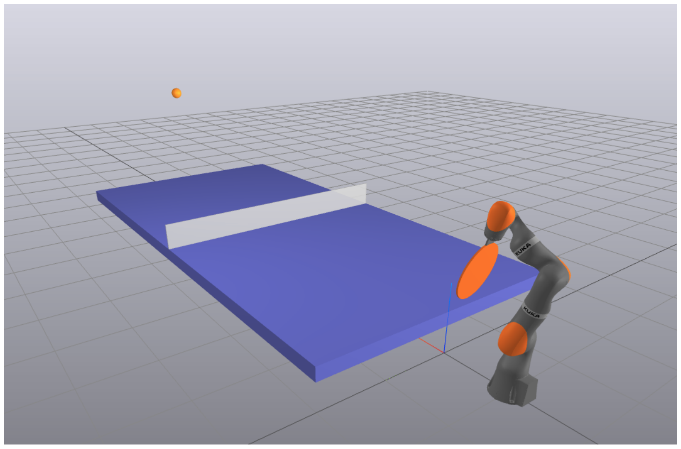


Fig. 5: **Setup for accuracy test.** We started the ball at the same position with same initial velocity for all tests and manually changed the robot's parameters until it could return the ball as close to the $y = 0$ line as possible.

C. *Built-in Drake functions*

In our third implementation, the majority of our controller is the same as in Sec. III-B. We use the same state transition diagram to decide how the paddle hits the ball and the same differential inverse kinematics controller to find the robot joint velocities. However, we change the PD controller connecting these two parts.

Rather than using Euler angles for our controller, we use quaternions. SLERP, or spherical linear interpolation, allows us to find a trajectory between any two quaternions on the unit sphere. Thus, if we use SLERP between the paddle's current angle and the desired angle, we can avoid any gimbal lock problems, as rotation will always be possible.

We did not have time to implement SLERP on our own, so we decided to use Drake's built-in PiecewisePose class. This class uses SLERP to smoothly interpolate between poses, and from it, we can directly get the desired angular velocity. Replacing our custom PD controller with this system enables much greater control of the paddle's angle.

IV. Experiments

A. *Accuracy*

Our first experiment tests the accuracy of each of the three implementations above for a single robot. We start the ball from a fixed position (at the point $(4, 0, 1.2)$ in the world frame) and the same initial starting velocity $(-8, 0, 0)$ for all the tests (see Fig. 5). These conditions mean the ball flies straight down the table (along $y = 0$) and bounce once before entering the hitting range of the robot. We then manually change parameters in our implementation of the robot to make it hit the ball as close as possible down the middle (along $y = 0$). We measure the y-coordinate of the ball at the end of the other side of the table.

Table I shows our results for the three methods.

Our results show that Drake's trajectory controller is 10^{14} times more accurate than the custom PD controller and 10^{16} times more accurate than the mirror law controller.

TABLE I: **Comparing accuracy across methods.** Drake's trajectory controller is significantly more accurate than the other two methods.

Model	y-coordinate
Mirror	0.603
Custom PD	0.00523
Drake	$\mathbf{-5.42 \times 10^{-17}}$

A related insight from this experiment is that the Drake controller allows for much finer adjustments of the robot's position and joint angles and a greater level of control over the hit than the other two methods. For the mirror law implementation, the motion of the robot is dominated by the physics of following the ball, and adjustments in the roll, pitch, and yaw of the paddle have little effect on the final trajectory of the ball. Comparing the custom PD controller to the Drake version, our custom implementation lacks the precision of the Drake controller, and changes in values for our custom PD controller have a less noticeable effect than changes in values in the Drake controller.

B. *Rally length*

Our second experiment measures the length of rallies that two of our robots can sustain against one another. We find that the mirror law and custom PD controller implementations can only hit the ball back once (not really a rally) but our Drake implementation can reliably hit the ball back and forth 3 times.

TABLE II: **Comparing rally length across methods.** While our first two methods could only hit the ball once, Drake's trajectory controller allowed a back-and-forth between the robots.

Model	Rally Length
Mirror	1
Custom PD	1
Drake	**3**

These results are consistent with the results of our first experiment, where we found that the Drake trajectory controller could most reliably hit the ball straight down the $y = 0$ middle line compared to the other two controllers. Given that our robots behave deterministically, we would expect the most predictable and consistent robot to be able to sustain the longest rally.

V. Conclusion and Future Work

Our work shows that while we get different results with different physics-based models, we can sustain a short rally between two robots using physics and domain knowledge alone. We have successfully solved parts of the table tennis problem which in previous formulations had been learned exclusively by a reinforcement learning algorithm. With our physics-based approach, we have successfully reduced the scope of the problem that RL needs to solve.

A major conclusion we can draw from comparing the results of our physics models for table tennis is that it is important to strike an appropriate balance between reducing the number of learnable parameters and the complexity of the physics model.

As an example, while the mirror law successfully reduces the number of learnable parameters to just the roll, pitch, and yaw of the paddle, the model itself is too simple to perform well in an actual rally. On the other hand, while our Drake controller is more complex than the mirror law implementation and has more parameters to consider, it still performs better than the mirror law implementation.

Future work in the area of table tennis robots should leverage our developments to build more computationally efficient RL solutions. These solutions would ideally learn policies for selecting some of the parameters we hard-coded and for dynamically changing them given the incoming ball's trajectory, spin, etc.

ACKNOWLEDGEMENTS

We would like to acknowledge the contributions of each individual team member. Both of us worked together on setting up the Drake environment and writing the mirror law and PD controller. Chaitanya implemented the third method using built-in Drake functions, and Dylan ran the accuracy and rally length experiments to evaluate our three systems. We both worked together to write the paper and make the final video presentation.

Additionally, we would like to express our appreciation for Professor Russ Tedrake, Dani White, Rachel Holladay, and Nora Jackson for an instructive and exciting semester of 6.800 and for providing invaluable guidance throughout the class and this project.

REFERENCES

[1] W. Gao et al., "Robotic Table Tennis with Model-Free Reinforcement Learning," arXiv:2003.14398 [cs, stat], May 2020, Accessed: Oct. 15, 2021. [Online]. Available: http://arxiv.org/abs/2003.14398.

[2] Y. Gao et al., "Optimal Stroke Learning with Policy Gradient Approach for Robotic Table Tennis," arXiv:2109.03100 [cs], Nov 2021, Accessed: Nov. 15, 2021. [Online]. Available: https://arxiv.org/abs/2109.03100.

[3] D. Silver et al., "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm," arXiv:1712.01815 [cs], Dec. 2017, Accessed: Oct. 15, 2021. [Online]. Available: http://arxiv.org/abs/1712.01815.

[4] A. Rizzi en D. Koditschek, "Further Progress in Robot Juggling: Solvable Mirror Laws", 05 1994, bll 2935–2940.