Learning to Strike for Robotic Table Tennis

Sakib Chowdhury and Yi Guo

Abstract— **Motivated by recent advances in robotic systems' ability to interact with dynamic environments, we study autonomous robotic table tennis in this paper. We design a high speed robotic arm to play table tennis, and develop a software system that predicts the incoming ball's trajectory and learns to control the robot arm for striking the ball to the opponent side of the table. Utilizing the idea of residual physics, we develop a residual predictor to predict a striking position for the incoming ball using sparse observation of the ball's starting trajectory. We control the arm to the contact-ready position, and then use a neural network to output the striking velocity to hit the ball. The neural network is trained through experience using a regression-based learning method. Simulation results in a robotic simulator demonstrate the superior performance, benefiting from efficient learning.**

I. Introduction

Recent advancements in robotic systems have significantly enhanced their ability to interact with dynamic environments. One critical area of focus has been the development of robots capable of accurately throwing objects [1] [2] [3]. Neural networks have demonstrated the potential of robotic systems to successfully throw objects at static [4] and dynamic [5] targets. However, if the object is a moving projectile, then receiving and striking the object becomes much challenging. An example of such a scenario is robotic manipulators playing table tennis.

The origins of robot table tennis research date back to the competitions initiated in 1983 [6], with early advancements made in the 1980s [7] [8]. In 1988, a notable development involved using a six-degree-of-freedom (DoF) PUMA 260 arm and a high-speed video streaming system to play table tennis under simplified rules [9]. At the start of the 21st century, learning-based approaches began to gain popularity in the field of robotic table tennis [10]. Although the robotic table tennis competitions concluded in 1993, they laid the groundwork for the discovery of new technologies.

Over the years, researchers have tackled numerous challenges in robotic table tennis, resulting in innovative solutions. While trajectory prediction for the ball can be done using the equations of projectile motion, these idealized models often fail to account for real-world factors such as frictional losses, air resistance and restitution, making accurate predictions difficult. To address this, a stereovision system was developed to track the ball's trajectory during table tennis, providing more robust data for trajectory estimation [11]. It was later improved by incorporating curve fitting techniques and an extended Kalman filter, which enhanced prediction accuracy by accounting for noisy measurements and non-ideal conditions [12]. Additionally, reinforcement learning has been applied to learn striking policies directly, bypassing the need for explicit trajectory prediction and offering an adaptive approach to striking the ball [13]. These methods all heavily rely on the historical position data of the ball to estimate its trajectory and to inform striking decisions.

Learning based approach proved successful in controlling joints for producing effective hit [13]. Biomimetic strategies for ball striking was proposed in [14]. More recently, wheel-based mobility with a robotic manipulator proved successful in receiving and striking the ball successfully for robotic wheel chair tennis [15]. Across these studies, various methods have been employed to predict the trajectory of the incoming ball and plan the paddle's hitting maneuver. The sooner robots predict the trajectory of the incoming ball, the quicker they can adjust their joint movements to strike it effectively.

In this study, we address the robotic table tennis problem using a learning-based approach aimed at early prediction of the ball's striking point and generating an optimal strike to return the ball to the opponent's side of the court. The system relies on a vision-based setup. Instead of predicting the entire trajectory, we directly predict the striking point using a shallow neural network that processes only two position samples of the ball to estimate the optimal striking point using a residual estimation method. For the strike planning task, we model the problem as an input-output system, where the desired landing position of the ball serves as the input and the robot's striking velocity is the output. We validate the performance of our approach in a robotic simulator, demonstrating that the learned models effectively ensure successful ball reception and striking within the table tennis environment.

The main contribution of the paper lies in the innovative learning methods for robotic table tennis: 1) a residual learning scheme to predict the landing position of the incoming ball with two position samples only from the beginning trajectory of the ball; and 2) a random forest regression based method to learn the appropriate velocity for striking the ball to reach the

The authors are with the Department of Electrical and Computer Engineering, Stevens Institute of Technology, Hoboken, NJ 07030, USA. Emails: {schowdhu1, yguo1}@stevens.edu

target location. The developed method contributes to the robotic ability to dynamically interact with the environment.

II. Robot Description and Problem Statement

Popular commercial robotic arms are designed for safe operation, which limits their ability to move the joints with high velocity. This makes most of the available commercial robotic arms unfit for table tennis. To overcome this problem, we design a custom robotic arm that allows faster motion. It includes precise joint actuation with high joint velocities and a high degree of freedom to enable complex and accurate movements. We present the design of the robotic arm below.

A. Robotic Arm

The design of the robotic arm is depicted in Fig. 1. It features 6 Degrees of Freedom (DOF), providing a high level of maneuverability and precision as shown in Fig. 1. Joints J_1, J_2, J_3, and J_4 are actuated by Xiaomi Cybergear motors, which allow for continuous rotation of 360 degrees. These joints have a maximum speed of 296 rpm and a peak torque of 12 N.m. Joints J_5 and J_6 are actuated by RDS5160 servo motors, capable of rotating from 0 to 270 degrees. The peak angular velocity of these joints is 461 degrees/sec. with a peak torque of 6.87 N.m.. The paddle is connected to the

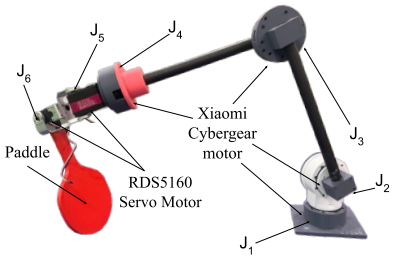

Fig. 1: Design of the robot arm to play table tennis

robotic arm via J_6. J_4, J_5 and J_6 are placed along the roll, pitch, and yaw axes of the paddle which enables the robot to precisely control the paddle's orientation only with these three joints. This precise control is crucial for accurately striking the ball and directing it to the desired location on the table tennis court.

B. Problem Statement

Robotic arms designed for table tennis must exhibit fast speeds in both actuation and decision-making. To execute a successful hit, humans estimate the ball's trajectory and position the paddle at a striking point. This process is refined through the player's experience and constant visual tracking of the ball. However, robotic systems are constrained by computational limitations, which means that continuous monitoring of the ball via cameras can reduce the time available for control tasks, such as motion planning, joint actuation, and hitting the ball.

We consider a robotic table tennis setup consisting of a ball, a tennis table, and the robotic arm, as depicted in Fig. 2. We assume that a ball is being thrown from a ball thrower and the first two position samples (when the ball launches from the thrower) with their corresponding time instants are known from the camera feeds. We denote these positions as $\mathbf{p_0} \in \mathbb{R}^3$ and $\mathbf{p_1} \in \mathbb{R}^3$, considering the frame rate of the camera system to be F_s. Since the robotic arm has many degrees of freedom, we constraint the striking point $\mathbf{p_s} \in \mathbb{R}^3$ on the $X = a$ plane in a global X-Y-Z coordinate system where a is a constant. Additionally, we define a pre-strike paddle orientation, $\boldsymbol{\theta_d}$, which represents the angular configuration of the paddle before it makes contact with the ball. We determine the joint angular velocity $\boldsymbol{\omega} = [\omega_1 - \omega_6]$ to effectively strike the ball ensuring the ball lands at the desired landing position, $\mathbf{p_L} \in \mathbb{R}^2$ on the table surface. Our problem is defined as: given two

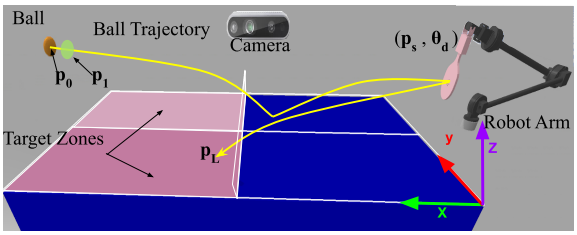

Fig. 2: The setup of the robotic table tennis

point observation $(\mathbf{p_0}, \mathbf{p_1})$ of the table tennis ball, design robot joint controller $w_i, i = 1, ..., 6$, such as the paddle attached to the robot arm strikes the ball to the desired location $\mathbf{p_L}$.

III. Methodology

In this section, we present our developed software system to solve the proposed problem, which contains the following:

1) *Striking Point Estimation Module* estimates the striking point, $\mathbf{p_s}$;
2) *Contact Ready Module* brings the paddle to $\mathbf{p_s}$ and keeps the robot standby for hitting the ball;
3) *Ball Strike Module* is responsible for estimating the arm's joint angular velocities: ω_4, ω_5 and ω_6, ensuring the ball lands at $\mathbf{p_L}$.

Fig. 2 depicts the architecture of the overall system. In the following, we explain each of the modules in details.

A. Striking Point Estimation Module

Given that $\mathbf{p_0}$ and $\mathbf{p_1}$ represent the first two position samples of the ball immediately after it is thrown from the ball thrower, we can express the initial velocity as:

$$\mathbf{v_0} = \frac{\mathbf{p_1} - \mathbf{p_0}}{\Delta t} \tag{1}$$

where Δt represents the time interval between $\mathbf{p_0}$ and $\mathbf{p_1}$.

With $\mathbf{p_0}$ and $\mathbf{v_0}$, the ideal trajectory can be determined using the equations of projectile motion. Assuming constant velocities in the horizontal plane (X and Y axis) and gravitational acceleration (g) acting in the vertical direction (Z axis), the ideal trajectory, $\mathbf{P_t}$ can be described as:

$$\mathbf{P_t(t)} = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + \begin{bmatrix} v_{0x} \\ v_{0y} \\ v_{0z} \end{bmatrix} t + \begin{bmatrix} 0 \\ 0 \\ -\frac{1}{2}g \end{bmatrix} t^2 \tag{2}$$

This approach takes only $\mathbf{p_0}$ and $\mathbf{p_1}$ to calculate the trajectory for the ball. While this methodology appears to accurately predict the ideal trajectory of the incoming ball, it fails to account for energy loss during the ball's bounce on the table, air-resistance, friction between the ball and the table surface, and other environmental factors prevalent in practical-world conditions. For instance, the coefficient of restitution, which measures how much energy is conserved in a collision by comparing velocities before and after impact, for a typical wooden table tennis court ranges from 0.74 to 0.78 [16]. These factors significantly influence the actual trajectory of the ball. We consider such deviations to be the residual impact of all the other physical parameters experienced in practical conditions. Given the complexity of estimating these residual impact and incorporating them into the actual trajectory calculations, we adopt a learning-based approach. This idea of estimating residual physics is inspired by [4]. We

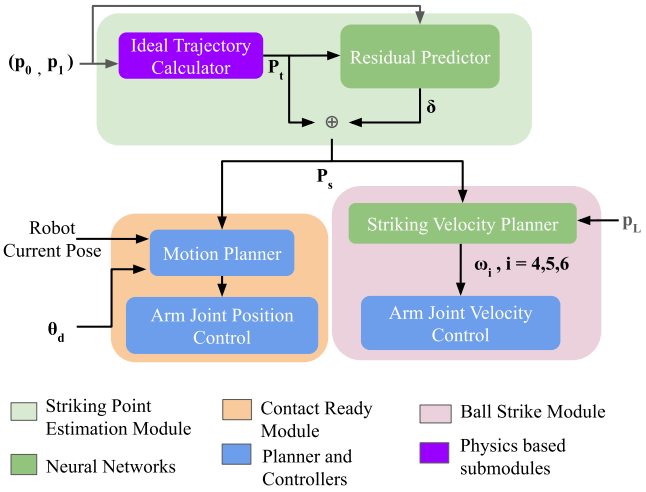

Fig. 3: Architecture of the proposed system

define the ball's actual trajectory as $\mathbf{P_p} \in \mathbb{R}^3$. The relation between $\mathbf{P_t}$ and $\mathbf{P_p}$ on the $X = a$ plane can then be defined as:

$$\mathbf{P_p} = \mathbf{P_t} + \boldsymbol{\delta} \tag{3}$$

where $\boldsymbol{\delta}$ represents the deviation of the trajectory due to the residual effects of the aforementioned physical factors. To estimate the residual effects, $\boldsymbol{\delta}$, we develop a learning-based system that models $\boldsymbol{\delta}$ in a supervised fashion. Details of this residual estimation is discussed in the following section.

1) Residual Predictor: We model the deviation, $\boldsymbol{\delta}$ of actual trajectory on the $x = a$ plane, as a function of $\mathbf{p_0}$, $\mathbf{v_0}$, d (distance between the base of the robot and $\mathbf{p_0}$) and the striking point on the ideal trajectory, $\mathbf{p_k}$ (which is constrained in the $X = a$ plane).

$$\boldsymbol{\delta} = \mathbf{F}(\mathbf{p_0}, \mathbf{v_0}, d, \mathbf{p_k}) \tag{4}$$

The residual predictor sub-module estimates the deviation $\boldsymbol{\delta}$ as defined in (4). We estimate the function $\mathbf{F}$ in (4) using a neural network. The architecture of the neural network used for this purpose is depicted in Fig. 4. This model is designed as a feedforward neural network, which consists of two sequentially connected linear layers. The first linear layer is followed by a Rectified Linear Unit (ReLU) [17] activation function, which introduces non-linearity into the network, allowing it to learn more complex representations of the input data. The ReLU function is particularly advantageous due to its simplicity and efficiency, as it only activates neurons that have a positive input. The second linear layer

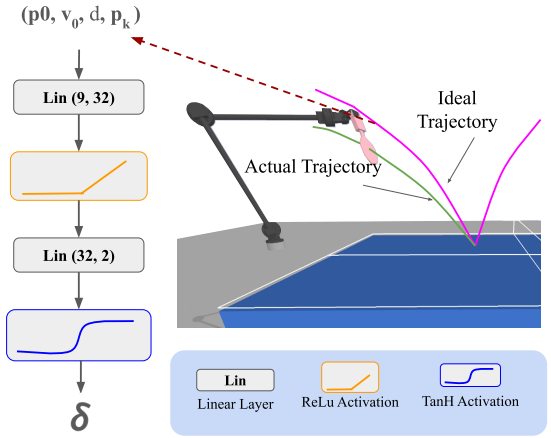

Fig. 4: The architecture of the residual predictor

is followed by a hyperbolic tangent (TanH) activation function, which squashes the output of the layer into the range $[-1, 1]$, as we normalize $\boldsymbol{\delta}$ during training. We also assume $\boldsymbol{\delta}$ having both positive and negative nature. The TanH activation function ensures the model can learn both these natures as output. This shallow architecture of the neural network is deliberately designed to ensure fast inference (about 4.3 ms on an Intel Xeon processor), which is critical for real-time operations.

To collect data for training, we throw the ball from random $\mathbf{p_0}$ with random $\mathbf{v_0}$ and we record the striking point $\mathbf{p_s}$ on the actual trajectory $\mathbf{P_p}$. The detailed training procedure of the residual predictor is shown in Algorithm 1. As shown in the algorithm, the system uses the training data $(\mathbf{p_0}, \mathbf{v_0}, \mathbf{p_s})$ as input. Based on this data, it estimates $\mathbf{v_0}$ and calculates $\mathbf{p_k}$ and d. Subsequently, the neural network is trained predicting $\hat{\boldsymbol{\delta}}$ and calculating the associated loss. Finally, the model parameters are updated using the gradients derived from the loss. Once the residual predictor is trained,

it estimates $\boldsymbol{\delta}$ and the trajectory estimation module computes $\mathbf{p_s}$ using (3) as the estimation of the strike point.

Algorithm 1 Training Procedure for Residual Predictor

Require: learning rate α, number of training samples N, training data $(\mathbf{p_0}, \mathbf{v_0}, \mathbf{p_s})$, number of epochs E
Ensure: Trained neural network parameters $\mathbf{F}$
Initialize environment
for epoch = 1 to E **do**
  Load data $D \leftarrow \mathbf{p_0}, \mathbf{p_1}, \mathbf{p_s}$
  Calculate $\mathbf{v_0}$ using (1)
  Calculate $\mathbf{p_k}$ and d
  $\boldsymbol{\delta} \leftarrow \mathbf{p_s} - \mathbf{p_k}$
  $\mathbf{X} \leftarrow \mathbf{p_0}, \mathbf{v_0}, d, \mathbf{p_k}$
  Initialize Residual Predictor model, $\mathbf{F}$
  Forward Pass: Compute Prediction $\hat{\boldsymbol{\delta}} \leftarrow \mathbf{F}(\mathbf{X})$
  Mean Squared Error Loss, $L \leftarrow (\hat{\boldsymbol{\delta}} - \boldsymbol{\delta})$
  Compute gradients $\nabla_{\mathbf{F}} L$
  Update model parameters: $\mathbf{F} \leftarrow \mathbf{F} - \alpha \nabla_F L$
end for
return F

B. Contact Ready Module

The contact ready module ensures that the robot's paddle is positioned accurately at the strike point, estimated by the strike point estimator described above. The process begins with the module adopting the pre-strike paddle orientation, $\theta_\mathbf{d}$, at $\mathbf{p_s}$, which forms the target pose for the robot.

The motion planner in the contact ready module takes both the current pose of the robot and the target pose as the input. It then computes the sequence of intermediate joint states necessary to transition the robot from its current configuration to the target pose. We use RRT-Connect motion planner [18] to effectively plan the robot arm's joint trajectory. These computed sequence of joint states are then sent to the arm joint position controller, which executes the required movements to achieve the desired contact angle of the paddle at the collision point. This sequence of operations ensures precise positioning of the paddle at $\mathbf{p_s}$, which is essential for the robot's successful interaction with the ball.

C. Ball Strike Module

As the robot reaches $\mathbf{p_s}$, it estimates the requisite joint velocities, $\boldsymbol{\omega}$, needed to position the ball on the desired landing position, $\mathbf{p_L}$ on the table.

As the joints $J_i (i = 4, 5, 6)$ are placed along the roll, pitch and yaw axes of the paddle, predicting their corresponding joint velocities $\omega_i (i = 4, 5, 6)$ is sufficient to exert force on the ball in any direction. To accurately determine the behavior, $\omega_i (i = 4, 5, 6)$, we formulate it as a function of $\mathbf{v_0}, \mathbf{p_s}$ and $\mathbf{p_L}$.

$$\boldsymbol{\omega}_e = \mathbf{T}(\mathbf{v_0}, \mathbf{p_s}, \mathbf{p_L}) \tag{5}$$

Algorithm 2 Training Procedure for striking velocity planner

Require: training data $(\mathbf{v_0}, \mathbf{p_s}, \mathbf{p_L}, \omega_i (i = 4, 5, 6))$, number of estimators α, number of training samples N,
Initialize environment
Collect data $D \leftarrow \mathbf{v_0}, \mathbf{p_s}, \mathbf{p_L}, \omega_i (i = 4, 5, 6)$
Initialize Random Forest Regressor, T, with α number of estimators
$X \leftarrow \mathbf{v_0}, \mathbf{p_s}, \mathbf{p_L}$
$Y \leftarrow \omega_i (i = 4, 5, 6)$
Fit $\mathbf{T}$ on (X, Y)
return T

where $\boldsymbol{\omega}_e = [\omega_4, \omega_5, \omega_6]$. $\mathbf{T}$ represents the striking velocity planner depicted in Fig. 3. To estimate the function $\mathbf{T}$, we model it with the Random Forest Regression algorithm [19]. To collect expert data for training the striking velocity planner, we throw the ball from a ball launcher with random throwing velocities $\mathbf{v_0}$, receive the ball at $\mathbf{p_s}$, perform random actions (hit the ball with random joint velocities $\omega_i, i = 4, 5, 6$) and record the corresponding landing position $\mathbf{p_L}$ as observations. With these action and observation pairs, the striking velocity planner is trained. The training procedure of the striking velocity planner is shown in Algorithm 2. After collecting the data, we initialize random forest regressor with α number of estimators. We fit the regressor with the inputs $\mathbf{v_0}, \mathbf{p_s}, \mathbf{p_L}$ and output $\omega_i, i = 4, 5, 6$. We found best prediction of $\omega_i, i = 4, 5, 6$ at $\alpha = 160$.

Once the hitting velocities are determined by the planner, the system transitions to a wait state, where it remains ready for action. The robot stays in this state until the the ball reaches the $X = a$ plane. Once the ball reaches the $X = a$ plane, the predetermined joint velocities are executed by the hitting controller. The hitting controller, implemented as a Proportional-Integral-Derivative (PID) controller, is responsible for precisely exerting the joint velocities as determined by the striking velocity planner.

IV. Simulation Results

We present the simulation results to validate the developed system in the following subsections.

A. Simulation Environment

The simulation environment is implemented using the PyBullet physics simulation engine [20]. The ball's radius and mass are set to replicate those of a standard table tennis ball, with a radius of 20 mm and a mass of 2.7 g, respectively. The dimensions of the table are configured to match those of a standard table tennis court, measuring 2.7 m in length, 1.53 m in width, and 1 m in height.

To further increase the realism of the simulation, the restitution coefficient between the ball and the table is

set to 0.75, which aligns with the typical restitution coefficient found in wooden table tennis courts [16]. Additionally, the lateral, spinning and rolling friction coefficients of the table is defined as 0.5, 0.05 and 0.05 respectively, ensuring a high-quality representation of real-world dynamics within the simulation environment. We further modeled air resistance as a velocity-dependent force $\mathbf{F}_{\text{air}} = -c_{\text{air}}\mathbf{v_b}$, where c_{air} is the air resistance coefficient and $\mathbf{v_b}$ is the ball's instantaneous velocity. This formulation follows the standard drag model for low Reynolds number regimes, where the resistive force is directly proportional to velocity. To mimic indoor environment, we set c_{air} to be 0.001. Note that the spin of the ball was not included as a state variable in these experiments, and only the linear velocity was considered. This choice was based on empirical observations that the ball launcher introduces minimal spin in practical settings.

We sample $\mathbf{p_0}$ and $\mathbf{p_1}$ from the simulator at a rate of 240Hz. All simulations and computations are conducted on an Intel Xeon Gold 6226R CPU, which operates at a maximum clock frequency of 3.9 GHz, ensuring high performance and reliability in processing intensive tasks.

B. Residual Predictor

In Table I, we compare the performance of the proposed residual predictor model with other popular estimators. In these experiments, the ball was thrown from random initial positions with varying initial velocities and $\mathbf{p_s}$ was estimated on the plane $X = a = 40$ cm in a global coordinate where $(0, 0, 0)$ is placed at the leftmost corner near the robot arm.

We conducted a total of 1000 experiments. The performance of each model was evaluated using three criteria: Mean Squared Error (MSE), Mean Absolute Error (MAE), and the R^2 score on estimating $\boldsymbol{\delta}$. The results clearly indicate that our proposed model outperforms all other models across these metrics. Specifically, our model achieves the lowest MSE and MAE values, as well as the highest R^2 scores. Although the R^2 score for the Y position is relatively modest at 0.1991, this result should be contextualized. Qualitative analysis reveals that the residual error δ in the Y axis is significantly smaller than that in the Z axis. This observation suggests that the lower R^2 score in the Y axis does not substantially affect the overall performance of the system. Thus, despite the comparatively lower R^2 score in the Y axis, our model effectively predicts the collision point, ensuring robust overall performance.

C. Motion Planning

Fig. 5 presents a comparison of the planning times for a range of motion planners where we choose 1000 random target points for each of the motion planners and recorded the planning times. The boxplot displayed in the figure provides a visual summary of the planning times for eight popular motion planners:

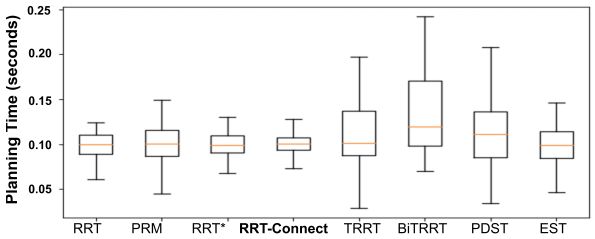

Fig. 5: Comparison of planning times for different motion planners. The line in the box shows the median, the box represents the interquartile range (IQR), and the whiskers indicate the maximum and minimum planning times.

RRT [24], RRT-Connect [18], PRM [25], RRT* [26], TRRT [27], BiTRRT [28], PDST [29] and EST [30]. From the boxplot, it is evident that among the motion planners evaluated, RRT-Connect demonstrates the most favorable performance. Specifically, RRT-Connect exhibits the lowest mean planning time, indicating that, on average, it generates solutions faster than the other planners. Additionally, RRT-Connect has the shortest interquartile range (IQR) of planning times, suggesting that its planning times are more consistent and less variable compared to the other planners. This implies that RRT-Connect not only provides quick solutions but also exhibits reliable and predictable performance across different instances.

D. Striking Velocity Planner

To evaluate the performance of the striking velocity planner, we conducted a series of experiments where the ball was thrown with varying initial velocities from a ball launcher. The striking point estimation module estimated $\mathbf{p_s}$ and RRT-Connect motion planner were used for motion planning. In these experiments, various target landing positions, $\mathbf{p_L}$, were selected on the table to provide a diverse range of scenarios for the ball strike module to handle. This experiment was repeated 1000 times to ensure robust performance evaluation.

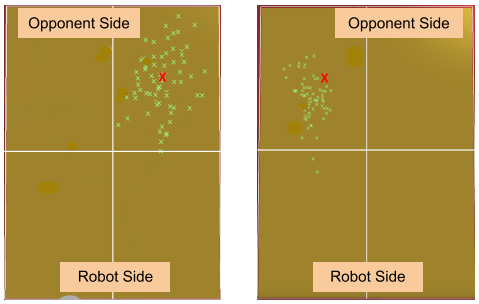

Fig. 6: Qualitative analysis of the striking velocity planner: The red 'X' marks the intended hitting point, while the green marks show where the ball actually landed after being hit.

The results of these experiments are summarized in Table II. The table clearly demonstrates that the Random Forest regressor with 160 estimators consistently achieves the highest R^2 scores across all axes.

TABLE I: Performance Comparison of Different Models for Residual Predictor

Model Name	MSE (y)	MAE (y)	R^2 score (y)	MSE (z)	MAE (z)	R^2 score (z)
SVR [21]	0.0011	0.0274	-4.2884	0.0069	0.0683	0.7753
Decision Tree [22]	0.0004	0.0074	-0.9394	0.0069	0.0483	0.7773
Random Forest (20 estimators) [19]	0.00023	0.00688	-0.0511	0.00409	0.03870	0.86868
Random Forest (40 estimators)	0.0002	0.0065	0.0077	0.0040	0.0385	0.8702
Random Forest (80 estimators)	0.0002	0.0063	0.0217	0.0039	0.0380	0.8729
KNN [23]	0.0002	0.0061	-0.3281	0.0051	0.0475	0.8352
OURS	***0.0001***	***0.0053***	***0.1991***	***0.0041***	***0.0346***	***0.8879***

TABLE II: Performance comparison of various regressor models acting as striking velocity planner

Model Name	Velocity (Roll)			Velocity (Pitch)			Velocity (Yaw)		
	R^2	MAE	MSE	R^2	MAE	MSE	R^2	MAE	MSE
Decision Tree	0.6809	2.4894	20.3232	0.7019	4.2688	38.5453	0.1687	2.1318	8.0570
KNN	0.7955	2.3220	13.0225	0.6523	4.3267	44.9649	0.6624	1.3665	3.2724
SVR	0.7998	2.7016	12.7478	0.6652	4.1480	43.2922	0.5958	1.5183	3.9179
Random Forest (20 estimators)	0.8700	1.7800	8.2792	0.7135	4.2880	37.0549	0.6473	1.4500	3.4180
Random Forest (40 estimators)	0.8864	1.4956	7.2346	0.6988	4.2282	38.9459	0.6760	1.3445	3.1404
Random Forest (80 estimators)	*0.8805*	*1.5479*	*7.6102*	*0.6736*	*4.3819*	*42.2076*	0.6857	1.3619	3.0461
Random Forest (160 estimators)	**0.8868**	**1.5371**	**7.2075**	**0.6891**	**4.3337**	**40.2061**	**0.6867**	**1.3539**	**3.0360**

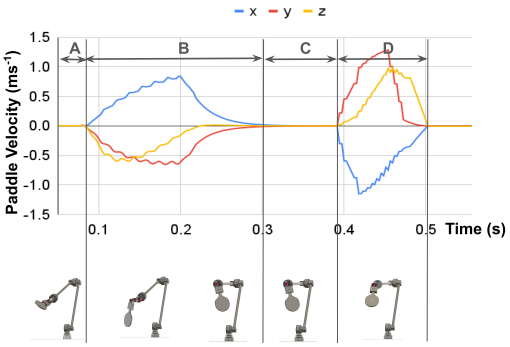

Fig. 7: Paddle velocity across operational stages: In stage 'A', the robot estimates the ball's striking point and plans joint movements to position the paddle at the predicted striking point. In stage 'B', the robot executes the joint movements. In stage 'C', the robot computes the striking velocity while waiting for the ball's arrival. Finally, in stage 'D', the robot executes the strike.

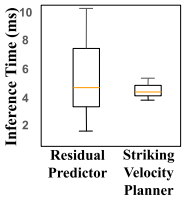

Fig. 8: Inference times of the residual predictor and striking velocity planner.

A qualitative analysis is shown in Fig. 6. It is clearly visible that the balls land much close to the intended landing site of the ball. It shows that the robot serves the purpose of delivering the ball on the desired side of the opponent court.

E. Complete System Performance

To thoroughly assess the performance of the complete system, we conducted an extensive evaluation involving 1000 experiments. In these experiments, we randomly threw balls from a ball launcher, and the system's success in intercepting these balls was recorded. When the residual predictor was used, the paddle successfully intercepted 985 out of the 1000 thrown balls, resulting in an impressive success rate of 98.5%. This high success rate demonstrates the effectiveness of the residual predictor in accurately intercepting the ball. In contrast, with no residual predictor the system relied solely on the physics-based trajectory predictions (P_p), the performance significantly deteriorated. The system consistently failed to intercept the incoming balls because the ball's trajectory altered with each bounce on the table due to frictional loss and other residual factors, leading to inaccuracies in the predicted interception points.

We evaluated the inference times for both the residual predictor and the striking velocity planner, as shown in Fig. 8. The residual predictor's inference time is under 10 milliseconds, with a median of 4.3 milliseconds, while the striking velocity planner's time is below 6 milliseconds, with a median of 4.2 milliseconds. These results demonstrate that incorporating the proposed method would add no more than 16 milliseconds, leaving ample time for motion planning and joint control, thus proving its efficiency for real-time applications in any sufficiently fast robotic systems.

F. Discussion and Future Work

The primary objective of this paper is to introduce a novel residual learning approach that learns from physics residual for autonomous robotic table tennis. In addition, we employed a learning from demonstration (LfD) approach to train the robot's ball-striking behavior. Compared with reinforcement learning (RL) methods that need extensive exploration and trial-and-error interactions with the environment, LfD approaches are more sample-efficient. Existing RL-based approaches ([13], [31]) require extensive real-world trials that may pose safety risks due to high-speed ball interactions

and potential damage to the robot. In contrast, our LfD-based method directly learns from the demonstration data, eliminating unsafe exploration.

We presented simulation validation using a physics engine, assuming two initial-point detection from a perception system. In our future work, we plan to conduct real robot arm experiments in a real-world table tennis setup. This involves integrating a high-speed 3D perception system for real-time ball tracking. To ensure low-latency operation of the contact-ready and the ball-strike modules, we will build a real-time Linux kernel that allows the robot to communicate with the workstation for real-time operations. The striking point estimation module will be adapted to account for sensor noise in real-world conditions. Additionally, we will fine-tune the motion planning and control strategies to accommodate hardware constraints and optimize execution speed. Due to the page limit and the complexity involved in hardware integration, an experimental demonstration is beyond the scope of this paper.

V. Conclusion

In this paper, we presented methods for controlling a table tennis-playing robot, comprising three modules: striking point estimation module, contact ready module, and ball strike module. The system uses the first few trajectory points of the ball to calculate the initial velocity and derive the physics-based trajectory. The striking point estimation module refines this trajectory by estimating the residual errors using a neural network model, ensuring accurate prediction of the contact point between the ball and the paddle. The contact-ready module employs the RRT-Connect motion planner for positioning the paddle at the predicted contact point. The ball strike module ensures precise impact velocity, directing the ball to the desired location on the opponent's court. This work contributes to the advancement of robotic systems capable of performing complex dynamic tasks with high precision and reliability.

References

[1] H. Miyashita, T. Yamawaki, and M. Yashima, "Control for throwing manipulation by one joint robot," in *IEEE International Conference on Robotics and Automation*, pp. 1273–1278.

[2] T. Frank, U. Janoske, and A. Mittnacht, "Automated throwing and capturing of cylinder-shaped objects," in *IEEE International Conference on Robotics and Automation*, pp. 5264–5270.

[3] D. Bianchi, M. G. Antonelli, C. Laschi, A. M. Sabatini, and E. Falotico, "Softoss: Learning to throw objects with a soft robot," *IEEE Robotics & Automation Magazine*, 2023.

[4] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser, "Tossingbot: Learning to throw arbitrary objects with residual physics," *IEEE Transactions on Robotics*, vol. 36, no. 4, pp. 1307–1319, 2020.

[5] H. Kasaei and M. Kasaei, "Throwing objects into a moving basket while avoiding obstacles," in *2023 IEEE International Conference on Robotics and Automation*, pp. 3051–3057.

[6] J. Billingsley, "Robot ping pong," *Practical Computing*, vol. 6, no. 5, 1983.

[7] J. Knight and D. Lowery, "Pingpong-playing robot controlled by a microcomputer," *Microprocessors and Microsystems*, vol. 10, no. 6, pp. 332–335, 1986.

[8] T. Broaden, "Toshiba progresses towards sensory control in real-time," *The Industrial Robot*, vol. 14, no. 1, pp. 50–52, 1987.

[9] R. L. Anderson, *A robot ping-pong player: experiment in real-time intelligent control*. MIT press, 1988.

[10] 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.

[11] Z. Zhang, D. Xu, and M. Tan, "Visual measurement and prediction of ball trajectory for table tennis robot," *IEEE Transactions on Instrumentation and Measurement*, vol. 59, no. 12, pp. 3195–3205, 2010.

[12] J. Tebbe, Y. Gao, M. Sastre-Rienietz, and A. Zell, "A table tennis robot system using an industrial kuka robot arm," in *Pattern Recognition, Stuttgart, Germany, October 9-12, 2018*. Springer, 2019, pp. 33–45.

[13] W. Gao, L. Graesser, K. Choromanski, X. Song, N. Lazic, P. Sanketi, V. Sindhwani, and N. Jaitly, "Robotic table tennis with model-free reinforcement learning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5556–5563.

[14] K. Mülling, J. Kober, and J. Peters, "A biomimetic approach to robot table tennis," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1921–1926.

[15] Z. Zaidi, D. Martin, N. Belles, V. Zakharov, A. Krishna, K. M. Lee, P. Wagstaff, S. Naik, M. Sklar, S. Choi *et al.*, "Athletic mobile manipulator system for robotic wheelchair tennis," *IEEE Robotics and Automation Letters*, vol. 8, no. 4, pp. 2245–2252, 2023.

[16] A. Haron and K. Ismail, "Coefficient of restitution of sports balls: A normal drop test," in *IOP conference series: materials science and engineering*, vol. 36, no. 1, 2012, p. 012038.

[17] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[18] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2000, pp. 995–1001.

[19] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[20] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2022.

[21] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and computing*, vol. 14, no. 3, pp. 199–222, 2004.

[22] J. R. Quinlan, *Programs for Machine Learning*. Morgan Kaufmann, 1993.

[23] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[24] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: A new tool for path planning," in *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 1, 1998, pp. 662–673.

[25] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," in *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, 1996, pp. 566–580.

[26] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[27] L. Jaillet, J. Cortés, and T. Siméon, "Transition-based rrt for path planning in continuous cost spaces," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 2145–2150.

[28] X. Tang and F. Chen, "Robot path planning algorithm based on bi-rrt and potential field," in *IEEE International Conference on Mechatronics and Automation (ICMA)*, 2020, pp. 1251–1256.

[29] J. G. Ziegler and N. B. Nichols, "Optimum settings for automatic controllers," *Transactions of the American society of mechanical engineers*, vol. 64, no. 8, pp. 759–765, 1942.

[30] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *The International Journal of Robotics Research*, vol. 21, no. 3, pp. 233–255, 2002.

[31] L. Yang, H. Zhang, X. Zhu, and X. Sheng, "Ball motion control in the table tennis robot system using time-series deep reinforcement learning," *IEEE Access*, vol. 9, pp. 99 816–99 827, 2021.