2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

Safe Table Tennis Swing Stroke with Low-Cost Hardware

Francesco Cursi, Marcus Kalander, Shuang Wu, Xidi Xue, Yu Tian,
Guangjian Tian, Xingyue Quan, Jianye Hao

***Abstract*— Playing table tennis with a human player is a challenging robotic task due to its dynamic nature. Despite a number of researches being devoted to developing robotic table tennis systems, most of the works have demanding hardware requirements and ignore safety measures when generating the swing stoke. To address these issues, we propose a safe motion planning framework that fully pushes the robotic hardware performance limits to play table tennis. In particular, we propose a pipeline to generate manipulator joint trajectories with environmental safety constraints and scale the trajectories to satisfy joint movement limitations. We use three different agents to validate the planning algorithm with our handmade robot platform in both simulation and real-world environments.**

I. Introduction

Playing table tennis is a challenging task for a robot manipulator. This is because it is a highly dynamic task with strong requirements on both robot control and perception, including high-speed motions, precision, correct timing, efficient and accurate ball tracking, and state estimations. Due to these challenges, researchers have focused on developing different strategies to allow for effective play. Some of the earliest works date back to Andersson et al. [1], [2] using a robot manipulator mounted on the ceiling and four different cameras to track the ball. These early approaches were mainly based on optimal control techniques, with model-based physics both for the robot and the ball.

In recent years, data-driven strategies like Reinforcement Learning (RL) have gained popularity in solving optimal control problems. Thanks to the advancement in computing power [3], [4], efficient algorithms [5], and improved simulators for sim-to-real transfer [6], as well as further researches on model-based approaches [7], [8], RL has become an alternative approach in robotic table tennis systems [9]–[14]. Researchers adopt RL in robotic table tennis primarily due to the high dynamic requirements and the need to be able to adapt to different playing strategies, players, models, and environment uncertainties.

Despite the emerging literature on robot table tennis platforms, a few critical issues have not been fully addressed. Firstly, the **hardware requirements and costs**. Most state-of-the-art approaches employ advanced and costly commercial robot products (e.g., Kuka [10] and WAM Barrett [7]) and vision systems (e.g., capable of 149 fps [10] and 250 fps [8]) that are not directed to the general public.

All authors are with Huawei Noah's Ark Lab, China.
Corresponding author: Shuang Wu
Email: wushuang.noah@huawei.com

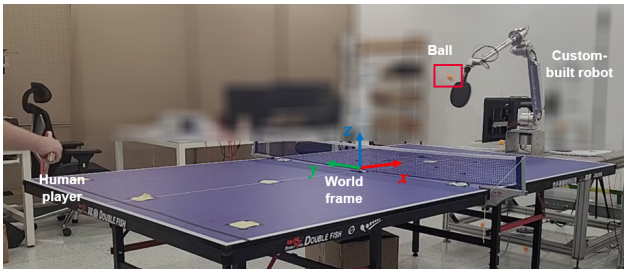


Fig. 1: The custom-built robot in the table-tennis scenario playing with a human.

Secondly, especially for real-world deployment, the **long training time** of RL agents. A typical approach is to train an agent end-to-end by outputting sequences of joint actions as in [11], [15]. Ding et al. [16] propose using human demonstrations to improve sample efficiency, yet, the approach requires hours of human data collection. Tebbe et al. [10], instead, propose a modified version of an actor-critic network and a parametrization of the stroke movement in order to improve sample efficiency over end-to-end learning. The hitting state is however defined only by two orientations and a single component for the hitting velocity, limiting the learned trajectories.

Additionally, none of the works has directly addressed the **safety issues and physical limitations** that robots have (e.g., joint position and velocity limits, avoiding contact with the table, etc.). Only a recent work [14] included safety in policy learning for robot table tennis, yet their approach requires a concurrent simulator to run in real-time and brute-force velocity limit saturation. The safety problem becomes more prominent if one uses a handmade robot, which does not have all the necessary safety measures as commercial robots.

In this work, we build a robot table tennis platform to address the above issues. Our platform is based on a low-cost custom-built robotic system consisting of a 6-DoF robot arm and two RGB monocular cameras. We propose a unified planning pipeline for generating safe swing strokes for robotic table-tennis, including safety bounds (e.g., hitting the table, reachable workspace) and robot's joint velocity limitations. We parameterize swing strokes with a set of parameters and develop a refinement procedure to plan the actual swing trajectory. Our parameterization scheme allows free swing motion while our refinement step ensures that all robot's physical and safety constraints are always satisfied. Our planning scheme is applicable to both model-based and RL-based agents. For RL agents, we do not learn sequences of actions in an end-to-end fashion since it is

not sample efficient and more challenging to satisfy the safety constraints. Additionally, the parametrization and the proposed motion planning are not robot-specific and can be adapted to any other robotic system.

From a planning perspective, [8] is closely related to our work. The authors use a quintic polynomial planner to reach the hitting state, but their approach is fully model-based, using ball physics to determine the hitting positions and heuristically setting the hitting velocity. Our parametrization in terms of hitting pose, velocity, and time allows for less heuristics and restricted motions, while also enabling us to inherently compensate for system latencies [14].

After conducting extensive experiments using our platform and planning algorithm, we find that a hybrid approach that combines both model-based and learning-based strategies yields the highest performance in terms of hit rate, return rate, precision, and training efficiency.

II. Preliminaries

This section gives a brief overview of general RL formalism and the physics model for ball trajectory prediction used to improve the RL training efficiency.

A. Reinforcement Learning

An RL agent is generally defined within the standard formalism of Markov Decision Process (MDP), which consists of a state space $\mathcal{S}$, action space $\mathcal{A}$, and a reward function r[1]. The agent, modelled by an artificial neural network, learns optimal action policies by interacting with the environment and receiving a reward for each proposed action.

This iterative process can be very time-consuming and inefficient if no *a priori* knowledge of the environment is given. In a table-tennis scenario, the agent should be able to learn optimal swing strokes to hit the ball to a goal position on the opponent's side in a continuous rally. For this reason, we propose to use a simplified ball trajectory prediction module to improve sample efficiency.

B. Ball Trajectory Prediction

The ball trajectory is crucial to determine where and when to hit the ball. Ball trajectories are defined by two phases and described by two corresponding models, *free-flight* and *bouncing* [7]. We ignore spin estimation which requires a high-speed camera running at high resolution [17]. Nevertheless, ignoring spin motions can lead to large ball prediction errors in the presence of high-speed spinning motions.

1) Prediction Model: In the free-flight phase, the ball's position and velocity $\boldsymbol{b}, \dot{\boldsymbol{b}} \in \mathbb{R}^3$ are expressed as a set of ordinary differential equations (ODE) derived from

$$\text{(free-flight)} \quad \ddot{\boldsymbol{b}} = \boldsymbol{g} - C_d \|\dot{\boldsymbol{b}}\| \dot{\boldsymbol{b}}, \tag{1}$$

where the C_d is the air drag coefficient and $\boldsymbol{g}$ is the gravity. Given an initial state $(\boldsymbol{b}_0, \dot{\boldsymbol{b}}_0)$, the ball's future states are predicted by solving an initial value problem $\phi_{\text{ivp}} : \boldsymbol{b}_0, \dot{\boldsymbol{b}}_0, T \rightarrow \boldsymbol{b}_t, \dot{\boldsymbol{b}}_t$ for $t \in [0, T_{\text{pred}}]$. The ball's velocity after bouncing on an object (table or racket) is obtained by the bouncing model

$$\text{(bouncing)} \quad \dot{\boldsymbol{b}}_o = (\boldsymbol{I} - \boldsymbol{R}\boldsymbol{K}\boldsymbol{R}^\top)\boldsymbol{v} + \boldsymbol{R}\boldsymbol{K}\boldsymbol{R}^\top \dot{\boldsymbol{b}}_i \ , \tag{2}$$

where $\boldsymbol{K} = \text{diag}(k_x, k_y, k_z)$ is the matrix of the object's restitution coefficients, $\boldsymbol{R} \in SO(3)$ the object orientation, and $\boldsymbol{v} \in \mathbb{R}^3$ the object linear velocity. In the case of contact with the table, the ball velocity after contact becomes $\dot{\boldsymbol{b}}_o = \boldsymbol{K}_{\text{table}} \dot{\boldsymbol{b}}_i$, with the table being stationary and its orientation frame aligned with the world frame.

2) Initial Ball State Estimation: The ball's initial state is required to predict the ball trajectory. Given a set of noisy measurements of the ball's positions $\{\boldsymbol{b}_0, \dots, \boldsymbol{b}_M\}$, a polynomial interpolation along x, y, z is performed to estimate the ball's initial position and velocity. We use first-order polynomials for x, y and a second-order polynomial for z. The polynomial coefficients are determined by solving the corresponding least squares regression problem from seven ball measurements. The latest interpolated point is then chosen as the initial ball's state for the trajectory prediction.

3) Parameter Estimation: We use offline ball trajectories $\tau_n, n = 1, \dots, N$ to estimate the parameters for the trajectory prediction models. Each trajectory is a sequence of ball positions $\tau_n = \{\boldsymbol{b}_{0,n}, \boldsymbol{b}_{1,n}, \dots, \boldsymbol{b}_{T,n}\}$ of different lengths due to the different durations. We estimate the bouncing coefficients $\boldsymbol{K}$ in (2) by solving a least squares problem from stacked velocities before and after bouncing, where the velocities are computed by finite difference with Gaussian smoothing. The drag coefficient C_d is computed directly by solving the nonlinear optimization to minimize the errors between the predicted ball positions, computed from the free-flight model initial value problem, and the measured ones for each trajectory τ_n.

III. Methodology

This section presents our unified pipeline considering the different agents to control the robot in the table tennis scenario, discussing the model-based and learning-based agents, and the motion planning approach.

A. Overview

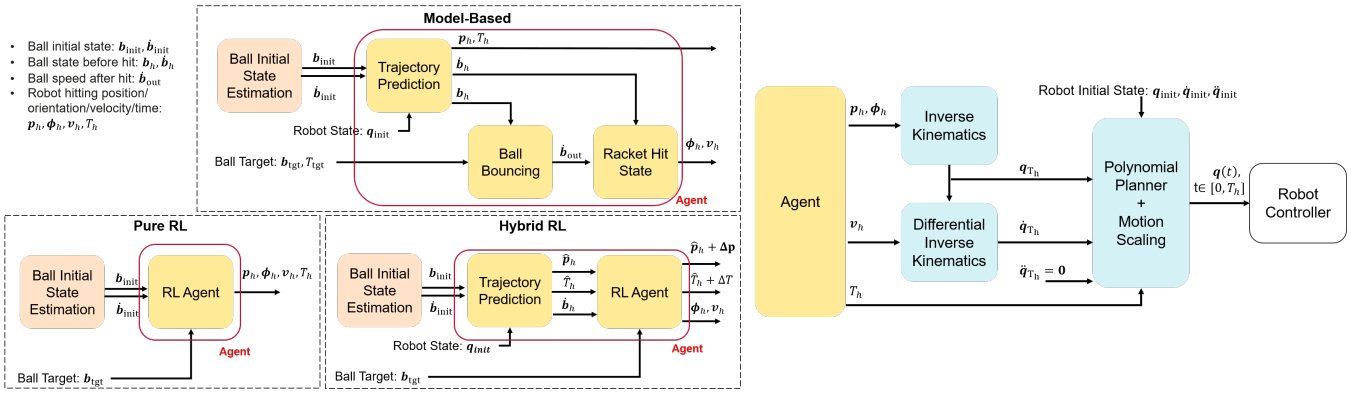
We consider three types of control agents: Model-based, model-free Pure RL, and Hybrid RL.[2] All three agents output a parameterized action to characterize the swing stroke trajectory. A swing action consists of ten real numbers:

$$\boldsymbol{p}_h \in \mathbb{R}^3, \ \boldsymbol{\phi}_h \in \mathbb{R}^3, \ \boldsymbol{v}_h \in \mathbb{R}^3, \ T_h \in \mathbb{R},$$

where $\boldsymbol{p}_h$ stands for the hitting position, $\boldsymbol{\phi}_h$ for the hitting orientation parameterized by the roll-pitch-yaw angle, $\boldsymbol{v}_h$ for the hitting velocity, and T_h for the hitting time. Fig. 2(a) and 2(b) depict the schematics of the three agents and the motion planning pipeline, respectively. It is worth mentioning that learning the hitting time helps to compensate

[1] In robotic table tennis, we consider a single step setup and thus ignores the state transition probability.

[2] To distinguish the model-based and learning-based strategies, we slightly abuse RL terminology by referring to model-free and model-based RL agents as Pure RL and Hybrid RL, respectively.



(a) The three control agents for action selection.

(b) Motion planning.

Fig. 2: The full control pipeline. (a) Three hitting strategies: (1) **Model-based**, predict hitting action with physics models (2) **Pure RL**, prediction without using physics models (3) **Hybrid RL**, modifies the Model-based prediction using RL mechanism. (b) The motion planning and robot control.

for latencies in the real-world setting. Additionally, the parametrization allows employing standard motion planning techniques that are guaranteed to ensure safety.

B. Model-based Agent

In a model-based setup, the hitting parameters are obtained according to the pipeline shown in Fig. 2(a).

Ball initial state estimation. Given the ball positions captured by the cameras, the initial ball position $\boldsymbol{b}_{\text{init}}$ and velocity $\dot{\boldsymbol{b}}_{\text{init}}$ are estimated as in II-B.

Trajectory prediction. With the ball's initial state, we predict the ball trajectory for $T_{pred} = 1.5$s by combining the free-flight model (1) and the bouncing model (2) for the contact with the table. From the predicted ball's trajectory, the closest point to the initial robot state $\boldsymbol{q}_{\text{init}}$ is chosen as the hitting point. This is selected among all points after the bounce that satisfies the safety conditions. This step determines the racket's hitting position $\boldsymbol{p}_h$, hitting time T_h, the ball's position $\boldsymbol{b}_h$ and velocity $\dot{\boldsymbol{b}}_h$ before hitting.

Get ball bouncing speed. This stage estimates the velocity the ball needs to reach after being hit by the racket ($\dot{\boldsymbol{b}}_{\text{out}}$) in order for it to reach a target goal $\boldsymbol{b}_{\text{tgt}}$ in a given time T_{tgt}, starting from the hitting position $\boldsymbol{b}_h$. Given the free-flight model ODE, a boundary value problem is solved $\phi_{\text{bvp}} : \boldsymbol{b}_h, \boldsymbol{b}_{\text{tgt}}, T_{\text{tgt}} \rightarrow \dot{\boldsymbol{b}}_{\text{out}}$.

Get racket hitting state. Once the desired ball's velocity after being hit is computed, the racket's velocity $\boldsymbol{v}_h$ and orientation $\boldsymbol{R}(\phi_h)$ in order for the ball to achieve $\dot{\boldsymbol{b}}_{\text{out}}$ need to be determined. This step adopts the racket's bouncing model (2) to solve a nonlinear optimization problem for the racket's velocity $\boldsymbol{v}_h$ and orientation $\boldsymbol{\phi}_h$:

$$\begin{aligned} \boldsymbol{v}_h, \boldsymbol{\phi}_h = \underset{\boldsymbol{v},\boldsymbol{\phi}}{\operatorname{argmin}} \, & \|\dot{\boldsymbol{b}}_{\text{out}} - (\boldsymbol{I} - \bar{\boldsymbol{K}}_r)\boldsymbol{v} - \bar{\boldsymbol{K}}_r \dot{\boldsymbol{b}}_h\|^2 \\ \text{s.t.} \quad & \bar{\boldsymbol{K}}_r = \boldsymbol{R}(\boldsymbol{\phi}) \cdot \boldsymbol{K}_r \cdot \boldsymbol{R}^\top(\boldsymbol{\phi}), \\ & \boldsymbol{v}_m \leq \boldsymbol{v} \leq \boldsymbol{v}_M, \;\; \boldsymbol{\phi}_m \leq \boldsymbol{\phi} \leq \boldsymbol{\phi}_M, \end{aligned} \tag{3}$$

where $\boldsymbol{v}_m, \boldsymbol{v}_M, \boldsymbol{\phi}_m, \boldsymbol{\phi}_M$ are the bounds on the velocity and Euler angles, and $\boldsymbol{K}_r$ the racket's bouncing matrix.

The model-based agent thus returns the hitting parameters $\boldsymbol{p}_h, \boldsymbol{R}_h, \boldsymbol{v}_h, T_h$ which are then used for the motion planning.

C. Learning-based Agents

The model-based agent is highly dependent on the accuracy of the ball predictions, on the estimated ball parameters, and on heuristics to determine the hitting velocity. Learning-based agents instead can compensate for such errors and possibly generate better swing strokes. We adopt a single-stage RL to find a solution through exploration and exploitation.

1) Model-free Pure RL Agent: According to the formalism in II-A, a state $s \in \mathcal{S} \subset \mathbb{R}^8$ consists of the initial ball position $\boldsymbol{b}_{\text{init}} \in \mathbb{R}^3$, the estimated initial ball velocity $\dot{\boldsymbol{b}}_{\text{init}} \in \mathbb{R}^3$, and the target position on the table $\boldsymbol{b}_{\text{tgt}} \in \mathbb{R}^2$. For the action space $\mathcal{A} \subset \mathbb{R}^{10}$, we learn the action $a = [\boldsymbol{p}_h, \boldsymbol{\phi}_h, \boldsymbol{v}_h, T_h]^\top$. We reduce the action space by only learning the magnitude of the Euler angles, while forcing the signs of the Euler angles to orient the racket towards the target location.

2) Hybrid RL Agent: The hybrid agent combines both the model-based prediction and the learning mechanism. Given the predicted hitting position $\hat{\boldsymbol{p}}_h$, hitting time $\hat{T}_h$, and ball velocity before hit $\dot{\boldsymbol{b}}_h$ from trajectory prediction as state space, the agent learns offset values for the hit position $\boldsymbol{\Delta p}_h$ and timing ΔT_h for the predictions and the actual hitting linear velocity and orientation from scratch like the model-free agent. The offset range is kept small ($\pm$10cm for $\boldsymbol{\Delta p}_h$ and $[50, 150]$ms for ΔT_h) to boost exploration efficiency.

3) Reward and Learning Algorithm: The Pure RL and Hybrid RL agents share the same reward function, which is a sum of five terms. We reward the agents for 1) hitting the ball, 2) the ball passing the net, 3) passing the net with a suitable height (too low or high are penalized), 4) the ball landing on the opponent's side of the table, and 5) distance to the target position using an inverse distance reward measure.

Sample efficiency is critical for real-world RL training. In particular, we would like to deploy a real-world table tennis robot with as little training as possible. We adopt Twin Delayed DDPG (TD3) [18] to train the learning-based agents. TD3 is a variant of the DDPG algorithm [19], which is an extension of the Q-learning algorithm for the continuous action space. TD3 mitigates the overestimation issue of

the Q-factor by using two Q-value networks. TD3 is an off-policy algorithm, which takes advantage of the replay mechanism and is thus more sample-efficient training than on-policy algorithms such as PPO [20][3].

Remark 1: Hybrid Reinforcement Learning (RL) integrates the strengths of model-based and learning-based approaches. Unlike model-based agents, it uses learning to correct prediction errors, enabling more diverse hitting motions. Compared to solely learning-based RL agents, it employs trajectory predictions to narrow down the exploration space.

D. Motion Planning with Physical Limitations

We consider the physical limitations of the robotic arm in the motion planning to ensure safe and reliable motions. With the given racket hitting state defined by $\boldsymbol{p}_h, \boldsymbol{\phi}_h, \boldsymbol{v}_h, T_h$, a feasible motion plan needs to be generated to move the robot from its initial state. Directly using a Cartesian motion plan requires solving inverse kinematics at each time step, which increases the computational overhead. The computation overhead further increases if we include the robot's physical limitations. Therefore, we formulate motion planning at the joint level (Fig. 2(b)).

We employ two different safety checks for the racket hitting position $\boldsymbol{p}_h$: a minimum height above the table empirically set to 15cm; hitting point within the robot workspace, which considers the additional deceleration motion after hit and restricts the hitting point to be within a radius of 95cm from the robot base.

1) Trajectory Planning: We use inverse kinematics to solve the robot's joint state at hitting time from the computed hitting position $\boldsymbol{p}_h$ and orientation $\boldsymbol{\phi}_h$ as

$$\boldsymbol{q}_{T_h} = \mathrm{IK}(\boldsymbol{p}_h, \boldsymbol{\phi}_h). \tag{4}$$

Then we solve the joint velocities at hitting while including joint velocity limits:

$$\begin{aligned} \dot{\boldsymbol{q}}_{T_h} = \underset{\dot{\boldsymbol{q}}}{\operatorname{argmin}} \frac{1}{2} \| \boldsymbol{W}(\boldsymbol{J}\dot{\boldsymbol{q}} - [\boldsymbol{v}_h^\top, \boldsymbol{0}^\top]^\top) \|^2 \\ \text{s.t.} \quad \dot{\boldsymbol{q}}_m \leq \dot{\boldsymbol{q}} \leq \dot{\boldsymbol{q}}_M, \end{aligned} \tag{5}$$

where $\boldsymbol{W} \in \mathbb{R}^{6\times6}$ is a diagonal weighting matrix, $\boldsymbol{J} \in \mathbb{R}^{6\times6}$ is the manipulator's Jacobian matrix, and $\dot{\boldsymbol{q}}_m, \dot{\boldsymbol{q}}_M$ are the minimum and maximum joint velocity limits. In this way, the robot motion does not exceed the physical limits but sacrifices the accuracy in achieving the desired linear and angular velocities at the hitting time.

We determine the trajectory using polynomial planners from the computed hitting states in joint space $\boldsymbol{q}_{T_h}, \dot{\boldsymbol{q}}_{T_h}$, the hit time T_h, and the current robot state $\boldsymbol{q}_{\text{init}}$. Our method is similar to [8]. For each joint $i = 1, 2, \ldots, 6$, we split the trajectory into a forward swing phase and a backward return phase and adopt separate quintic polynomials $q^{(i)}(t) = \sum_{n=0}^{5} c_n^{(i)} t^n,\ c_n^{(i)} \in \mathbb{R}$ to compute the robot state for any given time $t \in [0, T]$. The coefficients $c_n^{(i)}$ are computed by solving the linear system of equations system

[3]Similar results also observed in [21].

TABLE I: Duration T and boundary conditions for motion planning with quintic polynomials for each joint i.

phase	T	$q_0^{(i)}$	$q_T^{(i)}$	$\dot{q}_0^{(i)}$	$\dot{q}_T^{(i)}$	$\ddot{q}_0^{(i)}$	$\ddot{q}_T^{(i)}$
forward	T_h	$q_{\text{init}}^{(i)}$	$q_{T_h}^{(i)}$	0	$\dot{q}_{T_h}^{(i)}$	0	0
backward	T_h	$q_{T_h}^{(i)}$	$q_{\text{init}}^{(i)}$	$\dot{q}_{T_h}^{(i)}$	0	0	0

with boundary conditions on joint positions, velocities, and eventually accelerations as:

$$\begin{aligned} q^{(i)}(0) = q_0^{(i)},\ \dot{q}^{(i)}(0) = \dot{q}_0^{(i)},\ \ddot{q}^{(i)}(0) = \ddot{q}_0^{(i)}, \\ q^{(i)}(T) = q_T^{(i)},\ \dot{q}^{(i)}(T) = \dot{q}_T^{(i)},\ \ddot{q}^{(i)}(T) = \ddot{q}_T^{(i)}. \end{aligned} \tag{6}$$

The boundary conditions and duration T for the two phases are listed in Table I. We concatenate the forward and backward phases to form a complete motion trajectory.

2) Motion Scaling: The trajectory planner generates a set of joint positions and velocities for the robot to reach throughout the whole motion trajectory. The joint positions and velocity limits are guaranteed to be satisfied only at the beginning and at the end of the motion. For this reason, we scale down the computed trajectory if velocity limits are exceeded. In other words, the overall motion duration is enlarged. We therefore refine the planner in (6) in terms of the normalized time variable $\eta = t/T_h \in [0, 1]$. It can be shown [22, Section 7.7] that the joint velocities scale linearly with the motion duration as $\dot{\boldsymbol{q}}_t = \frac{\partial \boldsymbol{q}(\eta)}{\partial \eta}\frac{\partial \eta}{\partial t} = \frac{\partial \boldsymbol{q}(\eta)}{\partial \eta}\frac{1}{T_h}$. Given the initial planned trajectory, the time scaling factor is computed as $\sigma = \max\{1, \frac{\dot{\boldsymbol{q}}_{\text{lim}}}{\dot{\boldsymbol{q}}_M}\}$, where $\dot{\boldsymbol{q}}_M$ are the maximum allowable joint velocities and $\dot{\boldsymbol{q}}_{\text{lim}}$ the highest joint velocities achieved within the planned trajectory.

IV. System Deployment

We deploy our motion planning strategy on a real robotic platform. This section describes the vision system and the custom-made robot.

A. Vision System

We employ two monocular RGB cameras (Hikvision MV-CB013-A0UC) to capture the images. The cameras operate at a resolution of 1280×800 pixels at 83 FPS and an exposure time of 6.5ms. The cameras are placed on the ceiling to ensure that their field of view covers the entire table.

To detect the ball, we combine image segmentation and color filtering inspired by [23], [24]. We convert the raw images to the hue-saturation-value (HSV) color space and use manually tuned thresholds to obtain a binary mask. The background image is subtracted from the mask and size and shape features are adopted to extract the ball contour from any extraneous contours. To speed up computation, we restrict the ball detection to a small region of interest as the ball only moves short distances from one frame to the next. The 3D ball position is obtained by triangulating the 2D results from the two cameras. The raw images are obtained from the cameras in 12ms, while the detection procedure takes 3.5ms, resulting in 63 ball positions per second.

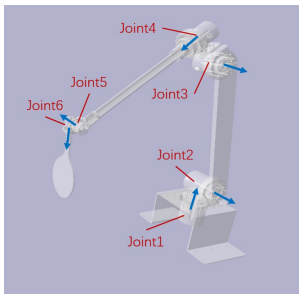


Fig. 3: The schematics of the custom-built robot. The blue arrows represent the joint axis of rotation.

TABLE II: Denavit-Hartenberg (DH) parameters.

joint i	d_i(m)	$\theta_i(^\circ)$	a_i(m)	$\alpha_i(^\circ)$
1	0.18	θ_1	0	-90
2	0	$\theta_2 - 90$	0.50	0
3	0	θ_3	0.07	-90
4	0.50	θ_4	0	90
5	0.08	$\theta_5 - 90$	0	-90
6	0.2	θ_6	0	0

B. Robot System

We employ a custom-built low-cost robot manipulator (Fig. 1). The robot schematic is shown in Fig. 3 and the corresponding DH parameters [25] in Table II. The robot is a 6-DoF manipulator with all revolute joints and a racket screwed on the end-effector. The links are made of aluminum and built in-house using standard CNC machining.

We adopt six Eyoubot motors [26] for actuating the revolute motions. Given the different sizes of the motors, the motor speed limits are $180, 180, 220, 220, 200,$ and $200°/s$. The low-level motor operates in a interpolation control mode using a cascaded proportional–integral–derivative (PID) algorithm with feed-forward terms to track the interpolated waypoints. The controller is set to run at 200 Hz.

Our custom-built robot costs less than 3,000 USD, which is more affordable than commercial robots. However, it lacks their extensive safety features, highlighting the need for a safe motion planner for swing strokes.

V. Results

This section shows results from using different agents in simulations and real-life games against a human.

A. Simulation Tests

The simulation setup is created in PyBullet [27] to simulate the physics and OpenAI gym [4] to build the reinforcement learning environment. For the simulation, the same pipeline as for the real case is used, except that the ball measurements are directly obtained from the simulator and not from the vision system. Seven ball positions are used to estimate the initial ball position as in section II. Additionally, the same robot limits as in the real case are set.

1) Training: We randomize the initial ball positions and velocities when training the learning-based agents. The initial velocity is determined such that the ball bounces on the robot's side of the table in a random time between 400 and 600 milliseconds. The agents then aim at a random target location on the opponent's side. An episode terminates if the ball falls outside of the table, if it hits the net, or if it bounces on the robot's side after being hit by the racket.

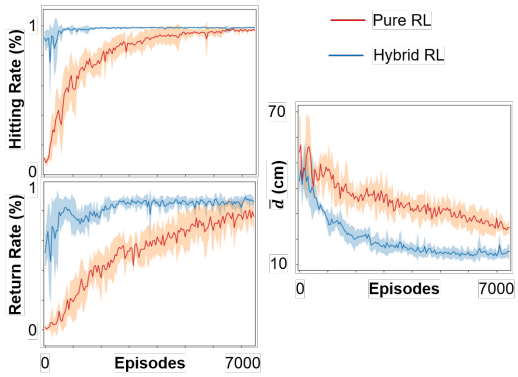


Fig. 4: Training performance over 5 runs for the Hybrid RL agent (blue) and the Pure RL agent (red) in simulation.

TABLE III: Simulation test results (average over 1,000 episodes). $\bar{d}$ is the average distance to the target.

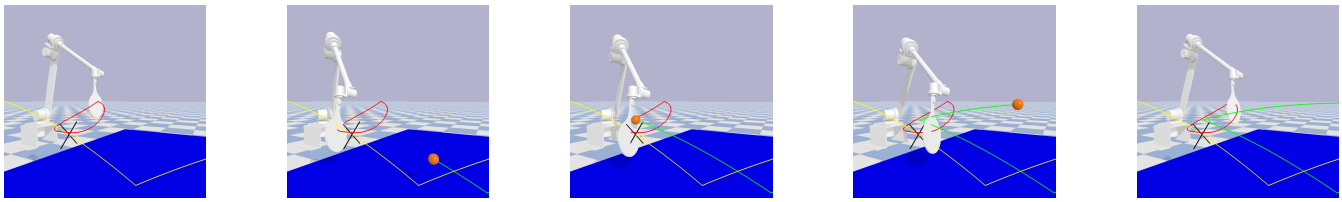
Agent	Training episodes	Hit rate (%)	Return rate (%)	$\bar{d}$ (cm)
Model-based	N.A.	85.8	71.8	24
Model-based (no safety)	N.A.	52.9	45.5	21
Pure RL	4,000	90.8	83.4	27
Pure RL	8,000	98.4	93.1	27
Hybrid RL	300	96.9	80.4	33
Hybrid RL	1,500	98.4	93.6	20

We train the agents for different numbers of epochs to evaluate the efficacy and efficiency of the two learning-based approaches. Fig. 4 shows the performance of each agent over the training episodes in terms of hit rate, successful return rate, and distance to the target for successful returns. As evident, the hybrid agent has better sample efficiency and can reach higher hitting rates, return rates, and accuracy faster than the model-free agent.

2) Evaluation: The same initial ball randomization as the training is used, yet, for easier comparison, the robot's target is fixed to be in the middle of the opponent's side of the table. The average velocity of the ball in simulation is 5.3 m/s with a range of $[4.0, 7.1]$ m/s. Table III reports the results for 1,000 episodes. Given the training results, we compare the model-based agent to the learning-based ones trained for different numbers of epochs. Both Pure RL and Hybrid RL largely improve over the model-based agent in terms of hitting and return rates. Only a slight deterioration occurs in the average distance to the target.

Comparing the Hybrid and Pure RL agents, the hybrid approach is already able to achieve superior performance when only trained on 300 episodes. Training the agent for longer largely improves the return rate and its precision. For the Pure RL, instead, a longer training improves both the hitting rate and return rate, but still fails to outperform the hybrid agent.

As shown in Fig. 5 with the screenshots of the Hybrid RL agent while playing, the benefit of using the learning-based approach is that it is able to properly return the ball by correcting the inaccuracies in the ball prediction model. However, even with learning-based agents, the physical robot limitations still highly condition the robot's performance.



(a) Planning (b) Pre-hit (c) Hit (d) Post-hit (e) Reset

Fig. 5: Example of ball hitting stages for the Hybrid RL agent. The solid green line is the actual ball trajectory, the yellow line is the estimated trajectory, the black cross is the predicted hit position, and the red curve is the robot's planned motion.

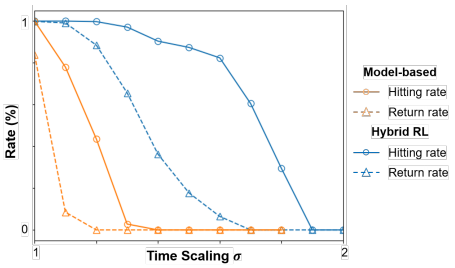


Fig. 6: The hitting and successful return rates as a function of the time scaling factor σ due to the joint velocity limits using the Hybrid RL (blue) and model-based (orange) agents.

Table III shows how the proposed motion planning with safety already improves the model-based agent. For further analysis, Fig. 6 shows how the time scaling factor from III-D affects the hitting and return rates. For the model-based agent the hitting rate and the return rate quickly deteriorate, even at low scaling factors: the return rate drops to below 10% when at least one of the joints exceeds the limit by only 10% ($\sigma = 1.1$). With the Hybrid RL agent, instead, similar drops occur at $\sigma = 1.6$, with the robot still having over 80% hit rate. The learning-based agent more effectively adjusts hitting parameters for improved swing strokes compared to the model-based agent. Yet, at higher scaling factors, decreased reachable speeds hinder successful returns.

B. Real-world Experiments

We evaluated the different agents in a real-world scenario with the robot playing cooperatively with a human user. The robot is required to return the ball at a specified and fixed location on the opponent's side. Because it is time-consuming and possibly dangerous to train the Pure RL agent, the same model as the one trained in simulation is used. For the Hybrid RL, instead, different models are used: trained purely in simulation (sim); pre-trained in simulation and fine-tuned in the real scenario for 100 or 200 episodes (sim+fine-tune); purely trained in the real scenario (from scratch). Table IV reports the comparative results for the different models when tested for 200 additional episodes[4]. The average initial ball velocity was estimated to be about 4m/s during training and testing. For simulation trained models, the best models reported in Table III are used.

Compared with the Model-based approach, the Pure RL have a worse performance, despite some improvements in simulation. The sim2real transfer of the Hybrid RL, instead, leads to slight improvements in terms of hitting rate, but to a lower precision and rally length. This is mainly due to discrepancies between the simulation and the real-world environments.

[4]We did not fine-tune Pure RL because its exploration space is too large.

TABLE IV: Real-world results from cooperative play between the agents and a human player (average over 200 episodes): (sim) agents are trained in simulation; (fine-tuned) are fine-tuned in the real setting; (from scratch) are entirely trained in real setting. $\bar{d}$ is the average distance to the target.

Agent	Episodes	Hit rate (%)	Return rate (%)	$\bar{d}(cm)$	Longest rally
Model-based	N.A.	84.5	82.0	**25.5**	18
Pure RL (sim)	0	77.5	58.0	33.3	9
Hybrid RL (sim)	0	89.5	81.0	33.1	16
Hybrid RL (sim + fine-tune)	100	91.0	85.0	29.2	18
Hybrid RL (sim + fine-tune)	200	**95.5**	**90.0**	26.3	15
Hybrid RL (from scratch)	400	94.0	87.5	32.2	**30**

The fine-tuning helps enhance the Hybrid RL agent. Our results show that with only 200 episodes it largely improves the hitting rate, return rate, and precision in reaching the desired target. Also, training from scratch improves the agent, but requires at least 400 episodes to achieve a performance similar to that of the fine-tuned one.

C. Further Discussion

Both simulations and real-world experiments show that adding prior knowledge to RL agents cuts down training time and outperforms purely model-based or learning-based methods. While all agents perform well in simulation, the difference is more noticeable in real-world settings. This is due to the limitations of our custom robot, vision system inaccuracies, and ball prediction errors. Notably, transferring model-free RL from simulation to reality is challenging, even though it's cost-effective to train purely in simulation.

VI. Conclusion

We here proposed a unified planning pipeline to enable a low-cost custom-built robot to play against humans, while satisfying safety constraints. Our framework allows employing both model-based and learning-based agents. Our results show the advantage of using learning-based agents, and the additional benefits of integrating model-based predictions with RL. We also show how the robot's physical limitations affect the agents and their capabilities to effectively learn alternative swing strokes to overcome those limitations. Currently, our robotic system is still limited in design and future work is focusing on deploying the pipeline on a more athletic robot with a larger workspace and higher performance.

REFERENCES

[1] R. Andersson, "Aggressive trajectory generator for a robot ping-pong player," *IEEE Control Systems Magazine*, vol. 9, no. 2, pp. 15–21, 1989.

[2] ——, "Dynamic sensing in a ping-pong playing robot," *IEEE Transactions on Robotics and Automation*, vol. 5, no. 6, pp. 728–739, 1989.

[3] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, nov 2017.

[4] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," 2016.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[6] A. S. Polydoros and L. Nalpantidis, "Survey of model-based reinforcement learning: Applications on robotics," *Journal of Intelligent & Robotic Systems*, vol. 86, no. 2, pp. 153–173, 2017.

[7] O. Koç, G. Maeda, and J. Peters, "Online optimal trajectory generation for robot table tennis," *Robotics and Autonomous Systems*, vol. 105, pp. 121–137, 2018.

[8] Y. Ji, X. Hu, Y. Chen, Y. Mao, G. Wang, Q. Li, and J. Zhang, "Model-based trajectory prediction and hitting velocity control for a new table tennis robot," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 2728–2734.

[9] 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 *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 5556–5563.

[10] J. Tebbe, L. Krauch, Y. Gao, and A. Zell, "Sample-efficient reinforcement learning in robotic table tennis," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 4171–4178.

[11] S. W. Abeyruwan, L. Graesser, D. B. D'Ambrosio, A. Singh, A. Shankar, A. Bewley, D. Jain, K. M. Choromanski, and P. R. Sanketi, "i-Sim2Real: Reinforcement learning of robotic policies in tight human-robot interaction loops," in *Conference on Robot Learning (CoRL)*, 2022, pp. 212–224.

[12] Y. Gao, J. Tebbe, and A. Zell, "A model-free approach to stroke learning for robotic table tennis," in *2022 International Joint Conference on Neural Networks (IJCNN)*, 2022, pp. 1–8.

[13] D. Büchler, S. Guist, R. Calandra, V. Berenz, B. Schölkopf, and J. Peters, "Learning to play table tennis from scratch using muscular robots," *IEEE Transactions on Robotics*, vol. 38, no. 6, pp. 3850–3860, 2022.

[14] D. B. D'Ambrosio, N. Jaitly, V. Sindhwani, K. Oslund, P. Xu, N. Lazic, A. Shankar, T. Ding, J. Abelian, E. Coumans, G. Kouretas, T. Nguyen, J. Boyd, A. Iscen, R. Mahjourian, V. Vanhoucke, A. Bewley, Y. Kuang, M. Ahn, D. Jain, S. Kataoka, O. E. Cortes, P. Sermanet, C. Lynch, P. R. Sanketi, K. Choromanski, W. Gao, J. Kangaspunta, K. Reymann, G. Vesom, S. Q. Moore, A. Singh, S. W. Abeyruwan, and L. Graesser, "Robotic table tennis: A case study into a high speed learning system," in *Proceedings of Robotics: Science and Systems*, 2023.

[15] Y. Huang, D. Büchler, O. Koç, B. Schölkopf, and J. Peters, "Jointly learning trajectory generation and hitting point prediction in robot table tennis," in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, 2016, pp. 650–655.

[16] T. Ding, L. Graesser, S. Abeyruwan, D. B. D'Ambrosio, A. Shankar, P. Sermanet, P. R. Sanketi, and C. Lynch, "Learning high speed precision table tennis on a physical robot," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 10 780–10 787.

[17] J. Tebbe, L. Klamt, Y. Gao, and A. Zell, "Spin detection in robotic table tennis," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 9694–9700.

[18] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International Conference on Machine Learning*, 2018, pp. 1587–1596.

[19] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2016.

[20] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[21] 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, 2023.

[22] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. Springer, 2010.

[23] J. Tebbe, Y. Gao, M. Sastre-Rienietz, and A. Zell, "A table tennis robot system using an industrial Kuka robot arm," in *German Conference on Pattern Recognition*, 2019, pp. 33–45.

[24] J. Tebbe, "Adaptive robot systems in highly dynamic environments: A table tennis robot," Ph.D. dissertation, Universität Tübingen, 2022.

[25] J. Denavit and R. S. Hartenberg, "A kinematic notation for lower-pair mechanisms based on matrices," *Journal of Applied Mechanics*, vol. 22, no. 2, pp. 215–221, 1955.

[26] "Eyoubot motors," http://2112095099.p.make.dcloud.portal1.portal.thefastmake.com/products_1/930837391971934208.html.

[27] E. Coumans and Y. Bai, "PyBullet, a python module for physics simulation for games, robotics and machine learning," 2016.