LATTE-MV: Learning to Anticipate Table Tennis Hits from Monocular Videos

Supplementary Material

Joint	Min angle (in °)	Max angle (in °)
A1	-170	170
A2	-120	120
A3	-170	170
A4	-120	120
A5	-170	170
A6	-120	120
A7	-175	175

Table 4. Kuka IIWA R820 joint angle ranges

A. Design of the controller to track any target pose of the end-effector

We use reinforcement learning to train a policy that can achieve a target pose within a given distribution in minimum time. The robot consists of manipulator with 7 joints and movement of the base in XY plane resulting into a 9DoF systems as it has 9 control variables to move the system. The racket is attached at the end-effector as shown in Figure 8. Reason for choice of a distribution for the target during training is to encourage taking solutions of joint angles and gantry position such that it can easily move to any pose within the distribution and also change within the same distribution very quickly. The exact joint angle range of the Kuka iiwa R820 are given in Table 4. We now describe the RL algorithm in detail

A.1. Reward design

We reward the RL agent for moving towards the target pose. Given a target goal T. The distance cost from the goal, C is defined as follows:-

$$C = (ee_{\text{position}} - T_{\text{position}})^2 + w_{\text{orientation}} \cosh\left((ee_{\text{quat}} T_{\text{quat}}^{-1}).w\right) \tag{17}$$

where $w_{\text{orientation}}$ is the weight factor for the orientation error in radians and is a hyperparameter set to 0.4 for the experiments in this paper, T_{position} is the goal position and T_{quat} is the goal orientation in quaternion, ee_{position} and ee_{quat} are the racket position and quaternion that is attached at the end-effector. The reward, R_t at each time step is given as follows:-

$$\begin{aligned} R_t &= -(C_t - C_{t-1}) + w_{\text{ac}}|a_t|^2 \text{ for } t > 0 \\ R_0 &= |a_0|^2 \end{aligned} \tag{18}$$

where a_t is the action command at time step t defined as the change in joint angle and gantry position wrt the current configuration. At every time step t, the target pose T_t is changed if the robot is able to match the pose of the racket on it's end-effector with some tolerance tol_t. The tolerance $\text{tol}_t = \text{tol}_{\text{final}} + (\text{tol}_{\text{init}} - \text{tol}_{\text{final}}) \exp^{-t/t_{\text{half}}}$ is changed in a curriculum from tol_{init} to $\text{tol}_{\text{final}}$. Mathematically, change in target pose T_t is defined as follows:-

$$\begin{aligned} T_t &= \begin{cases} T_{t-1} & \text{if } C_t \le \text{tol}_t, \\ T_{\text{new}} \sim T_{\text{dis}} & \text{otherwise} \end{cases} \quad \text{for } t > 0 \\ T_0 &\sim T_{\text{dis}} \end{aligned} \tag{19}$$

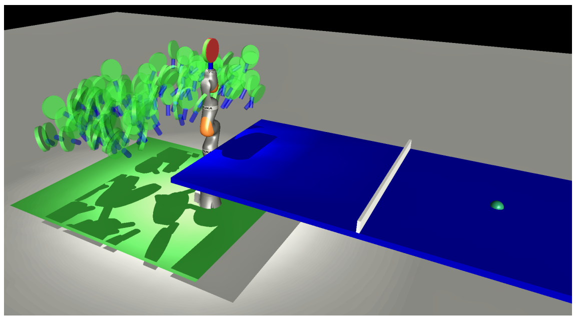


Figure 9. 100 poses sampled from T_{dis}

A.2. Training target pose distribution

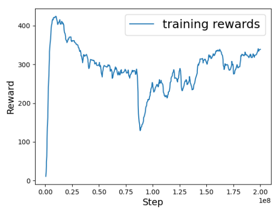
The target pose distribution T_{dis} is defined as follows:-

$$\begin{aligned} T_{\text{dis}} = \{ & \text{pos} \sim \mathcal{U}(\text{pos}_{\text{min}}, \text{pos}_{\text{max}}) \\ & \text{euler} = \text{euler}_{\text{facing}} + e \sim \mathcal{U}(-\text{euler}_{\text{range}}, \text{euler}_{\text{range}})\} \end{aligned} \tag{20}$$

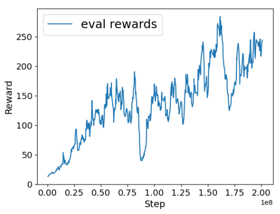
where $\text{euler}_{\text{facing}}$ is the orientation of racket face that extends to the target point in Figure 7 with handle facing in -Z direction; $\text{pos}_{\text{min}}, \text{pos}_{\text{max}}, \text{euler}_{\text{range}}$ are chosen accordingly. This enables learning to achieve poses only within the distribution of where players usually hold their racket to return the ball facing the table. 100 randomly chosen poses in this distribution are given in Figure 9

A.3. Policy training

We use Proximal Policy Optimization (PPO) RL algorithm to learn the optimal policy to maximize the reward in the environment described above. The episode sixe is chosen to be 1000 time steps with a time step $dt = 0.01s$. The training curve for rewards and the evaluation performance is given in Figure 10. For evaluation performance, we fix $\text{tol} = \text{tol}_{\text{final}}$ and choose the policy with best performance within $200M$ environment steps



(a) Train



(b) Eval

Figure 10. Rewards

B. Contact modeling

It is important to specify how the ball will bounce when collided with the table and the racket so as to recreate the ball exchanges collected from videos as described in Section 3

B.1. Racket to ball contact

For the racket to ball contact we assume lossless bounce i.e. given the racket normal n_{racket} and the ball velocity before collision v_{before}. The velocity after collision, $v_{\text{after}} = v_{\text{before}} + 2(v_{\text{before}}.n_{\text{racket}})n_{\text{racket}}$. This conserves the speed ($|v_{\text{before}}| = |v_{\text{after}}|$) and as the racket is assumed to be static, it does not impart any extra speed to the ball.

B.2. Ball to table contact

For the ball to table contact, we aim to exactly recreate the trajectory parabola obtained in Section 3. For this, we fit the ball velocities for each segment divided by when ball contacts the table or the opponent's racket. Whenever ball makes a collision with the table or the opponent's racket, the new velocity is set to the values fit for the new segment as fit in Section 3. This very closely enables recreating the trajectories in Section 3

C. Setting target pose G

The target pose G is calculated as the pose to reflect the ball based on Section B.1 such that it hits the table as close as possible to the target point b_{target}. Mathematically, given the table height z_{table}, velocity of the ball at the hitting point v_{before}, the hitting point on the racket b_{hit}, and the function f_{norm} that extracts the racket hitting plane normal from pose, this can be formulated as:-

$$
\begin{aligned}
G = \underset{G}{arg\,min} |b_{\text{target}} - p|^2 \\
\text{where } p_x &= b_{\text{hit},x} + v_{\text{reflected},x} t_c \\
p_y &= b_{\text{hit},y} + v_{\text{reflected},y} t_c \\
v_{\text{reflected}} &= v_{\text{before}} + (v_{\text{before}} . f_{\text{norm}}(G)) f_{\text{norm}}(G) \\
t_c &= \frac{-\sqrt{v^2_{\text{reflected},z} - 2g(b_{\text{hit},z} - z_{\text{table}})} - v_{\text{before},z}}{g}
\end{aligned}
\tag{21}
$$

where $g = -9.81 m/s^2$ is the acceleration due to gravity constant

D. Additional results

D.1. Varying central pose C

The results presented in Table 3 are with the central position, C at the center. However, we also set pose C not to be in the center of the table but biased towards one side. We set C as the mean of all the positions in the training dataset from where the player hits the ball. This is to show that the transformer does not just learn the mean of all returns made by the opponent but rather a correlation with the current pose, history of poses of the opponent. The updated results are given in Table 5. The return rates improve with the updated starting pose C for all the 3 cases, however the trend remains the same, i.e. the return rates with using the ground truth is the highest, followed by using our prediction for pre-positioning and the return rate if not pre-positioning is the lowest. This shows that the trained transformer indeed learns a correlation of the human poses with the anticipated return trajectory of the ball.

Pre-Pos. Strategy	Return Rate	Return Accuracy	Pose Accuracy
Baseline	52.9%	0.506 m	0.23 m / 12.68°
Anticipatory	**62.5%**	**0.469 m**	**0.17 m / 9.43°**
Oracle	*66.2%*	*0.489 m*	*0.14 m / 5.93°*

Table 5. Return rates for the robot under different pre-positioning strategies with C as the mean position of all the hit points in the training dataset and orientation facing the table. Return accuracy is the mean deviation of the return bounce point from the target point on the opponent's side of the table. Pose accuracy is the mean difference in pose (position error/orientation error) of the racket achieved vs target pose at the time when the ball hits or passes the racket.

D.2. Varying λ

Next, we study the effect on results by changing the hyperparameter λ which dictates how much to trust the prediction made by the anticipatory algorithm. $\lambda = 0$ means trusting the prediction completely, while $\lambda = 1$ means the anticipatory algorithm is completely trusted in setting the target pose. Table 6 shows results with different values of λ. As can be seen, we get the best return rate at $\lambda = 0.1$

Pre-Pos. Strategy	$\lambda = 0$	$\lambda = 0.1$	$\lambda = 0.5$
Baseline	49.9%	49.9%	49.9%
Anticipatory	55.6%	**59.0 %**	54.4%
Oracle	*64.5%*	*64.5%*	*64.5%*

Table 6. Return rates for the robot under different values of λ and using our anticipatory pre-positioning strategy

D.3. Varying T_h

Next, we study the effect on results by changing the hyper-parameter T_h, the length of history that dictates the time to anticipate before the opponent hits the ball. There is a trade-off between accuracy and the available time for the anticipatory algorithm to pre-position. With larger T_d, the anticipatory algorithm will have more response time, but the anticipation will be more inaccurate, as it will be hard to tell how the opponent will hit more ahead of time. Vice-versa for shorter T_h. Hence, the value of T_h is chosen accordingly. Table 7 shows results with different values of T_h. As can be seen, we get the best return rate at $T_h = 0.2s$.

Pre-Pos. Strategy	$T_h = 0.1s$	$T_h = 0.2s$	$T_h = 0.4s$
Baseline	49.9%	49.9%	49.9%
Anticipatory	57.3%	**59.0 %**	58.4%
Oracle	*62.1%*	*64.5%*	*65.2%*

Table 7. Return rates for the robot under different values of λ and using our anticipatory pre-positioning strategy