1 Supplementary Information

Supplementary videos to this article can be found under https://sonyresearch.github.io/ace_public/.

1.1 Player profiles

Table S1 presents the profiles of the elite and professional players who played with *Ace* in April 2025.

Table S1: Player profiles. All participating players were right-handed.

Player	Gender	Experience (years)	Avg. Training Frequency (week^{-1})	Avg. Hours Per Training Session (session^{-1})	Record
Elite A	F	15	6	4	Won Rookie Tournament, 2019. Participated in Singles competitions of Inter Highschool National Championships, 2021.
Elite B	F	15	6	4	Best 32 in All Japan University Doubles, 2024.
Elite C	F	12	6	3	71st Kanto High School Table Tennis Tournament, 2021.
Elite D	M	10	6	4-5	Participated in Team and Singles competitions of Inter Highschool National Championships, 2023.
Elite E	M	15	5	2	Won the Division 2 Singles Championships at the National Selection Tournament, 2024.
Pro 1	F	24	6	5	Mix doubles champion in All Japan Championships 2025. Highest world ranking 29, 2019.
Pro 2	M	17	6	5	Singles quarter-finalist in All Japan Championships Men's 2025. Highest world ranking 205, 2019.

1.2 Comparison of visuomotor latency

Table S2 shows the latency of each component in our robot. The overall latency is less than 1/10 of the elite players'.

Table S2: Comparison of visuomotor latency: human players vs *Ace*

Module	Subtask	Human player (ms)	*Ace* (ms)
Perception	Image Acquisition	–	8.0
	Ball Detection	–	1.9
	Triangulation	–	0.3
	Spin Estimation	–	1.4[1]
	Subtotal: perception	–	**10.2**
Control policy	NN inference (state-to-action)	–	0.5[2]
	FAOC	–	0.01[3]
	Reset planner	–	1.0[4]
	Collision check	–	1.5[5]
	Subtotal: control policy	–	**5.0**[6]
Robot actuation	EtherCAT communication	–	2.0
	Servo latency	–	3.0
	Subtotal: robot	–	**5.0**
Total latency		**232**[7]	**20.2**

Notes

1. Spin is assumed to be constant during flight and computed asynchronously. Therefore the spin estimation latency is excluded from the perception latency, and thus from total latency.
2. Approximate value; depends on the neural network control policy used.
3. FAOC maps an action to a waypoint and computes a 32 ms-long trajectory segment.
4. Reset plan is computed from the end of the trajectory segment to bring the robot to a rest state. Computational time may vary depending on segment length (typically 0.5–1.0 ms).
5. Collision checks are applied to both the reset plan and trajectory segment. Duration varies with path complexity (around 0.7–1.5 ms).
6. Conservative upper bound to account for variability (e.g. collision check taking longer due to a longer reset trajectory).
7. Human visuomotor reaction latency (232 ms) for elite table tennis players is extracted from published data [1].

1.3 Comparison of performance

Table S3 presents a comparison of performance statistics in various table tennis matches between *Ace* and human players.

Table S3: Comparison of performances observed during the matches

Metric		**Human player**		***Ace***	
Maximum racket linear velocity[1] [m/s]	All motions	>20[2]		19.8[3]	
	At ball impact	>20[2]		11.5	
Maximum ball linear velocity[4] [m/s]		Serve	Return	Serve	Return
		10.1	25.9	9.1	16.4
Maximum ball angular velocity[4] [rad/s]		Serve	Return	Serve	Return
	Magnitude	740	867	376	600
	Top spin[5]	354	854	296	596
	Back spin[5]	439	366	353	486
	Side spin[6]	667	484	292	464
	Cork spin[7]	421	256	280	158
Working envelope[1] (X [m] × Y [m])	Observed	4.3 × 2.4		2.3 × 2.4	
	Theoretical	7.0 × 7.0		3.6 × 3.6	

Notes

1. Values for human players were estimated using perception data. Values for *Ace* were measured using encoders.
2. Estimated value due to difficulty in accurate measurement with unaltered markerless rackets.
3. Value observed while executing a reset trajectory.
4. Values measured immediately after ball impact using perception data.
5. Component values of angular velocity in the horizontal axis perpendicular to the flight direction (Top Spin: positive rotation, Back Spin: negative rotation).
6. Component value of angular velocity in the axis perpendicular to the other two.
7. Component value of angular velocity in the axis parallel to the flight direction.

1.4 Rally - RL policies

Here, we present additional details and evaluations with respect to the RL policies used by *Ace* during a rally.

1.4.1 Full state definitions

Table S4 provides a more detailed breakdown of the variables used to construct the state ($\boldsymbol{s}_t = [\boldsymbol{s}_t^{\text{ball}}, \boldsymbol{s}_t^{\text{robot}}, s_t^{\text{skill}}]$) for the RL policies.

1.4.2 Action computation time

Extended Data Fig. 1 shows how the state ($\boldsymbol{s}_t$) is constructed every 32 ms for the policies used in the rally phase of the game. As the computation of the segment trajectory ($\boldsymbol{Q}_t^{*[1:T]}$) also requires time-consuming operations such as reset plan generation

Table S4: Full State Definitions

State Component	Variable	Dimension	Description
$\boldsymbol{s}_t^{\text{ball}}$	$\boldsymbol{P}_{\text{ball}}^{[1:N]}$	3×15	The past N=15 ball position (3D) estimates.
	$\boldsymbol{\Omega}_{\text{ball}}^{[1:N]}$	3×15	The past N=15 ball spin (3D) estimates.
	$\boldsymbol{t}_{p_{\text{ball}}}^{[1:N]}$	15	The past N=15 ball time-stamps for the position estimates. The timestamps are represented as normalized delays between 9 and 16 ms based on the current timestamp. If no sensor measurement is available than a value of -1 is used.
	$\boldsymbol{t}_{\omega_{\text{ball}}}^{[1:N]}$	15	The past N=15 ball time-stamps for the spin estimates. The timestamps are represented as normalized delays between 9 and 16 ms based on the current timestamp. If no sensor measurement is available than a value of -1 is used.
$\boldsymbol{s}_t^{\text{robot}}$	$\boldsymbol{x}_{t-1}^{Q*[T]}$	24	The terminal joint states (position (8D), velocity (8D) and acceleration (8D)) of the trajectory segment computed in the previous RL step.
	$\boldsymbol{x}_{t-1}^{EE*[T]}$	10	The terminal end effector state (position (3D), rotation (2D), velocity (3D) and angular velocity (2D)) of the trajectory segment computed on the previous RL step.
$\boldsymbol{s}_t^{\text{skill}}$	$y_{\text{desired}} \in [-0.7, 0.7]$	1	The desired y-landing position for the current shot.
	$w_p \in [0, 1]$	1	The position reward weight for the current shot.
	$w_s \in [-1, 1]$	1	The spin reward weight for the current shot

and collision checking, we use a time budget of 5 ms for action computation. $\boldsymbol{s}_t$ is constructed using the current ball state ($\boldsymbol{P}_{\text{ball}}^{[1:N]}$, $\boldsymbol{\Omega}_{\text{ball}}^{[1:N]}$, $\boldsymbol{t}_{p_{\text{ball}}}^{[1:N]}$, $\boldsymbol{t}_{\omega_{\text{ball}}}^{[1:N]}$) and the desired y-landing position (y_{desired}) and reward weights ($\boldsymbol{w}_{\text{reward}}$) for the current shot. Also, $\boldsymbol{s}_t$ contains the terminal joint ($\boldsymbol{x}_{t-1}^{Q*[T]}$) and end effector states ($\boldsymbol{x}_{t-1}^{EE*[T]}$) from the previously computed trajectory segment that is currently being executed (i.e. 5 ms into the future). Conceptually, this means that when the action is computed, after the 5 ms

Table S5: Sampling probabilities of initial ball states across all KDE models for initial game states of serves and rallies.

KDE Models	**Serve** $p_{\text{serve}}(.)$	**Rally** $p_{\text{rally}}(.)$
Human - Beginner	0.018	0.042
Human - Coaches	0.054	0.126
Human - Elites	0.108	0.252
Synthetic	0.120	0.280
Total	0.3	0.7

time budget, the observation of the ball state is 5 ms delayed whereas the joint and end effector states represent the current open-loop state.

1.4.3 Ball distributions

At the start of each episode during policy training, the initial ball state is sampled from a collection of KDE models according to Table S5. These KDE models are prefitted to human datasets (derived from real-world gameplay and mirrored with respect to the XZ plane) and a synthetic dataset (generated in simulation), which are split depending on the game state into serve and rally datasets. By adjusting the bandwidth parameter of the KDE models, the rate of out-of-distribution (OOD) samples can be controlled, where a higher rate of OOD samples promotes robustness of the resulting policy.

Since sampling from the fitted KDE models does not inherently guarantee valid initial ball states, the sampling process is repeated until a valid sample is drawn. Fig. S1 (**a**, **b**) illustrates that the synthetic distribution covers the entire human distribution, while the human distribution is much more concentrated in regions corresponding to realistic hit states. Furthermore, the t-SNE plots in Fig. S1 (**c**, **d**) reveal distinct regions in the embedding of the 9-dimensional initial ball states, even when restricted to the subset of human datasets, highlighting the difference in playing styles across the different groups.

1.4.4 Policy reward details

Table S6 shows the reward structure for all rally policies used in *Ace*. The different reward terms can be broadly classified into the mutually exclusive terms (R_{miss}) missing the ball, ($R_{\text{hit}}^{\text{return}}$) hitting the ball and successfully returning it or ($R_{\text{hit}}^{\neg\text{return}}$) hitting the ball but failing to return it. The final reward is computed as a weighted sum of the terms and their corresponding coefficients. A detailed description of each of the terms in Table S6 can be found in Table S7.

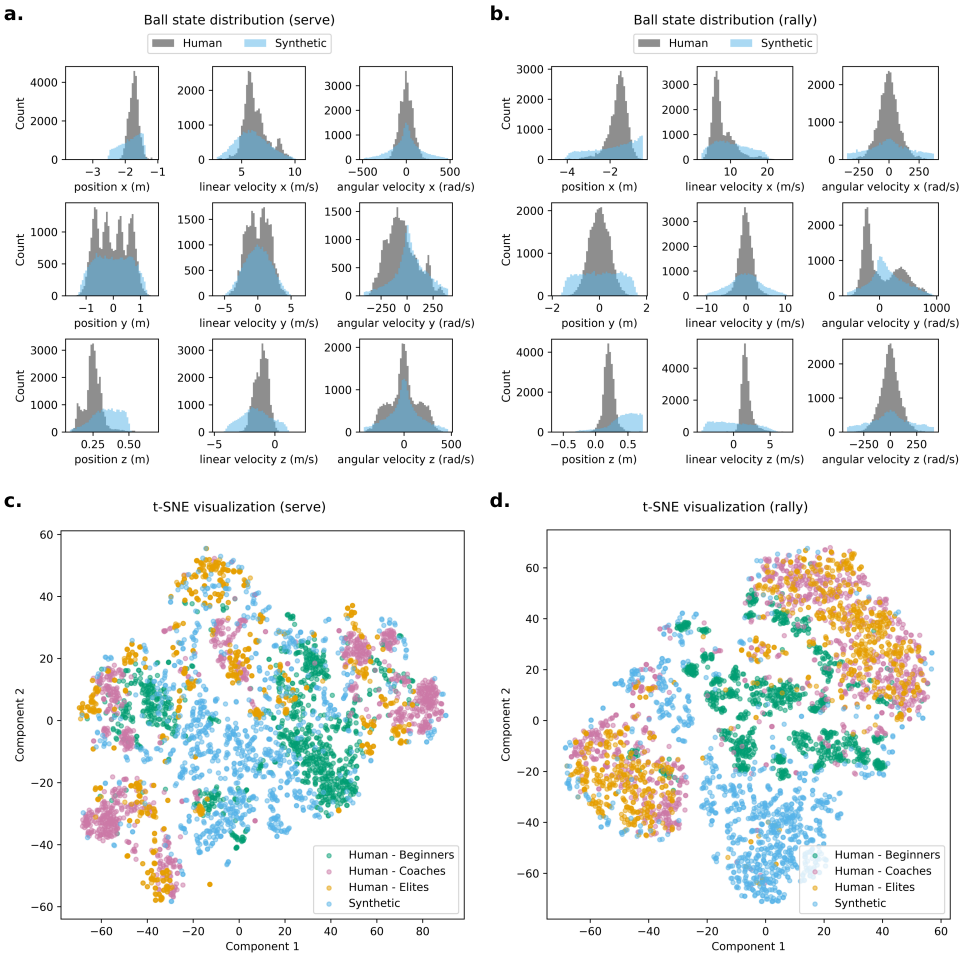

Figure S1: **Distributions of initial ball states used during policy training.** **a**, **b.** Histograms of the initial ball states for serve and rallies respectively. **c**, **d.** Two-dimensional t-SNE embeddings of 1000 sampled initial states for each KDE model.

$\eta(w_s)$ and $\eta'(w_s)$ are coefficients for $R_{\text{vel}}^{\text{ball}}$ and $R_{\text{height}}^{\text{ball}}$ respectively (Table S7) and are a function of the spin weight w_s:

$$\begin{aligned} \eta(w_s) &= \sigma\left(T(w_s + \frac{1}{2})\right) \\ \eta'(w_s) &= 1 - \eta(w_s) \end{aligned} \tag{1}$$

where T is a free temperature parameter and σ is the sigmoid function. While w_s is positive $\eta(w_s)$ remains at 1.0 and $\eta'(w_s)$ remains at 0.0. Conceptually this encourages the policy to play high velocity top-spin shots regardless of their height over the

Table S6: Policy Rewards. The reward function for a given policy (π_i) is defined as the sum of the terms each multiplied by their respective coefficient. y is the desired y-landing position, $w_p \in [0, 1]$ is the position reward weight and $w_s \in [-1, 1]$ is the spin reward weight. These are used to parameterize a subset of the reward functions and are observable to the policy. ω_y refers to the angular velocity of the ball around the y-axis upon landing.

Rewards		Coefficient for $\boldsymbol{\pi_i}$									
Type	**Term**	π_0	π_1	π_2	π_3	π_4	π_5	π_6	π_7	π_8	π_9
$\boldsymbol{R}_{\text{miss}}$	R_d^{racket}	1	1	1	1	1	1	1	1	1	1
$\boldsymbol{R}_{\text{hit}}^{\neg\text{return}}$	1	1	1	1	1	1	1	1	1	1	1
	R_ω^{racket}	1	1	1	1	1	1	1	1	1	-
	R_d^{table}	1	1	1	1	1	1	1	1	1	-
$\boldsymbol{R}_{\text{hit}}^{\text{return}}$	3	1	1	1	1	1	1	1	1	1	-
	R_ω^{racket}	1	1	1	1	1	1	1	1	1	-
	$R_{\text{vel}}^{\text{ball}}$	$\eta(w_s)$	$\eta(w_s)$	$\eta(w_s)$	1	1	1	1	-	1	-
	$R_{\text{height}}^{\text{ball}}$	$\eta'(w_s)$	$\eta'(w_s)$	$\eta'(w_s)$	-	-	-	-	1	-	-
	$R_{\text{pos}}^{\text{ball}}(y)$	w_p	w_p	w_p	-	-	-	-	-	-	-
	$R_{\text{spin}_y}^{\text{ball}}$	w_s	w_s	w_s	-	-	$\max_{(0,\omega_y)}$	$\max_{(0,\omega_y)}$	$\min_{(0,\omega_y)}$	-	$\min_{(0,\omega_y)}$
	$R_{\text{spin}_{\lvert z\rvert}}^{\text{ball}}$	-	-	-	-	-	-	-	-	1	-
	$R_{\text{bounce}}^{\text{ball}}$	-	-	-	-	-	-	-	-	-	1
	$R_{\text{robust}}^{\text{racket}}$	-	-	-	-	-	-	-	-	-	1

net. However, as w_s becomes increasingly negative $\eta(w_s)$ decreases to 0.0 and $\eta'(w_s)$ increases to 1.0. Therefore as the policy is encouraged to play back-spin shots the shot velocity becomes less important and the height over the net becomes more important. This allows the policy to learn slower shots with high back spin that have a low trajectory over the net.

1.4.5 Reward weight sampling

Algorithm 1 outlines the procedure used to sample reward weights during the training and execution of policies for the rally phase of the game. The reward weights $w_p \in [0, 1]$ and $w_s \in [-1, 1]$ are subject to this sampling procedure. The procedure is designed to bias the reward weight vectors towards sparse solutions and non-zero values close to $|1|$. During training this allows the policy to focus on strong independent skills such as top spin or aiming only, which provide clear learning signals. During execution this biases skills towards extreme behaviors, due to the non-zero values close to $|1|$, which are assumed to be the most competitive shots.

1.4.6 Real-world skill comparisons

Fig. S2 shows the ability of policy 0, conditioned on a y-landing position (y_{desired}) and a set of reward weights ($\boldsymbol{w}_{\text{reward}} = [w_p, w_s]$), to perform different skills against

Table S7: Description of Reward Terms.

Term	Description
R_d^{racket}	Positive reward inversely proportional to the distance between the center of the racket and the ball when the ball passes the racket due to a miss.
R_ω^{racket}	Negative reward proportional to the angular velocity of the racket at hitting time.
R_d^{table}	Positive reward inversely proportional to the distance between the ball and the closest edge of the table as it passes the table xy-plane due to a missed return.
$R_{\text{vel}}^{\text{ball}}$	Positive reward proportional to the velocity of the ball upon landing.
$R_{\text{height}}^{\text{ball}}$	Positive reward inversely proportional to the Euclidean distance between the height of the ball as it crosses the net and a desired height over the net.
$R_{\text{pos}}^{\text{ball}}(y)$	Positive reward inversely proportional to the Euclidean distance between the desired and achieved y-landing position.
$R_{\text{spin}_y}^{\text{ball}}$	Reward (can be positive or negative) proportional to the signed angular velocity of the ball in the y-axis upon landing.
$R_{\text{spin}_{\lvert z\rvert}}^{\text{ball}}$	Positive reward proportional to the absolute angular velocity of the ball in the z-axis upon landing
$R_{\text{bounce}}^{\text{ball}}$	Positive reward negatively proportional to the maximum height of the ball after bounce.
$R_{\text{robust}}^{\text{racket}}$	Positive reward inversely proportional to (1) the magnitude of the change in ball pitch required to make it touch the net or miss the table and (2) the angular velocity of the racket at hitting time.

Algorithm 1 Sampling Procedure for Reward Weights

Require: β_N temperature for the number of reward weights to be changed (N_{ch}). β_x temperature for the reward weight value (x). $w_{\text{ch}}^{\min}$ minimum value for chosen reward weight (w_{ch}). $w_{\text{ch}}^{\max}$ maximum value for chosen reward weight (w_{ch}).

$w_p \leftarrow 0$
$w_s \leftarrow 0$
$N_{\text{ch}} \sim \frac{e^{-\beta_N i}}{\sum_{j=0}^{2} e^{-\beta_N j}}, i = 0, 1, 2$

for N_{ch} **do**
  $w_{\text{ch}} \sim \text{Unif}\{w_p, w_s\}$ (without replacement)
  $p_{\text{accept}} \leftarrow 0$
  while $\mathcal{U}(0,1) > p_{\text{accept}}$ **do**
    $x \sim \mathcal{U}(w_{\text{ch}}^{\min}, w_{\text{ch}}^{\max})$
    $p_{\text{accept}} \leftarrow \frac{e^{\beta_x |x|}}{e^{\beta_x}}$
  end while
  $w_{\text{ch}} \leftarrow x$
end for

return (w_p, w_s)

elite human opponents. The y-axis represents the y-component of the ball state upon landing on the opponent's side of the table. We define a top spin skill as any shot where the spin reward weight was greater or equal than 0.75 ($w_s \geqslant 0.75$) and a back spin skill as any shot where the spin reward weight was lower or equal than -0.75 ($w_s \leqslant -0.75$). Based on this definition, one can see that the returned ball angular y-velocity (where positive x is the direction of the ball) was predominantly positive when requesting a top spin skill and predominantly negative when requesting a back spin skill. Similarly, we define the aiming skill as any shot where the position reward weight was greater or equal than 0.75 ($w_p \geqslant 0.75$) and no aiming as any shot where the position reward weight was lower or equal than 0.25 ($w_p \leqslant 0.25$). In this case, the Euclidean distance between the desired (y_{desired}) and achieved ($\tilde{y}$) y-landing positions was significantly lower when the policy was asked to perform the aiming skill.

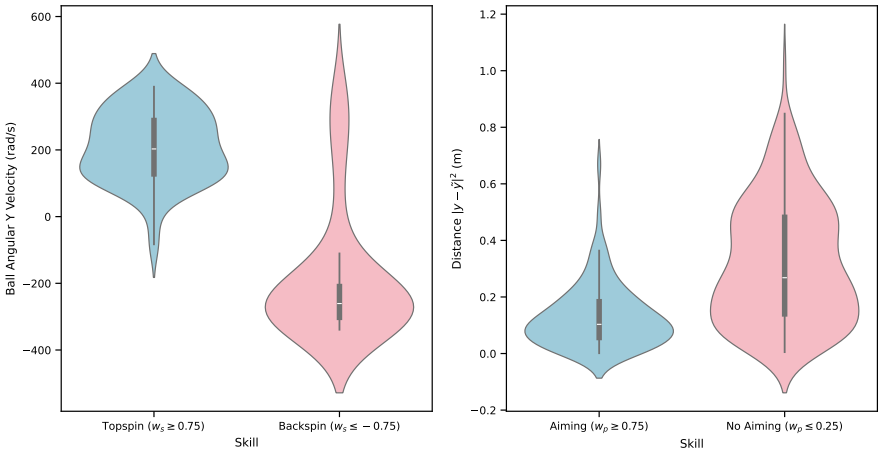

Figure S2: State of the ball upon being returned by the skill-conditioned policy for matches against elite players. **Left:** Comparison of ball angular y velocity (rad/s) for shots with a spin reward weight (w_s) greater or equal than 0.75 and lower or equal than -0.75. This corresponds to asking the policy to either produce high top spin or high back spin. **Right:** Comparison of the Euclidean distance between the desired and achieved y-landing position for shots with an aiming reward weight (w_p) greater or equal than 0.75 and lower or equal than 0.25. This corresponds to asking the policy to either aim to the target position or to ignore the target position.

1.4.7 Soft actor-critic hyper-parameters

Table S8 details the hyper-parameters used for Soft Actor Critic (SAC) in order to train policies for the rally phase of the game. For both the actor and critic networks we use a ResNet architecture [2] and for the processing of the ball position and spin histories in the actor we use 1D convolutions. We collect the first 200,000 transitions using an initial controller that is different from the policy. Both in the initial controller

Table S8: SAC Hyper-Parameters.

Value	Description
$2,000,000$	Replay buffer size
512	Minibatch size
30	The number of roll-out workers for data-collection
0.001	Target smoothing coefficient
0.0001	Learning rate
0.99	Discount factor
4	Policy: Number of residual blocks
2	Policy: Residual block depth
256	Policy: Number of units per layer
$[32, 64]$	Policy: Number of filters for convolution applied to ball history
$[5, 5]$	Policy: Kernel size for convolution applied to ball history
8	Policy: Embedding dimension after convolution applied to ball history
6	Critic: Number of residual blocks
2	Critic: Residual block depth
1024	Critic: Number of units per layer
ReLU	Non-Linearity
0.0024	Entropy temperature
$200,000$	Number of random actions before sampling from the policy
0.1	Rate of mean-reversion in the OU process
0.2	Magnitude of the noise in the OU process
1000	Temperature for scheduling the OU process based on the equation $e^{-E/1000}$ where E is the number of model updates

and on top of the policy we inject noise modeled as the Ornstein-Uhlenbeck (OU) stochastic process [3, 4] in order to aid exploration of time-correlated actions. In the initial controller the level of noise is constant, while in the policy the OU noise component is scheduled such that it decreases exponentially with the number of model updates.

1.4.8 Event tables

Event Tables are separate replay buffers containing transitions leading to a certain event $\mathcal{E}$ [5]. The events used in our training pipeline are defined based on heuristics by the following conditions:

- *close_hit*: the ball passes close to the center of the racket, within a threshold of 10 cm plus the racket radius, but without a successful hit.

- *ball_hit*: racket hits the ball.

- *ball_returned*: robot returns ball on the opponent side of the table.

- *smash_skill*: Let $v = \sqrt{v_x^2 + v_y^2 + \hat{v}_z^2}$ be the velocity of the ball when it lands on the opponent's side of the table, where $\hat{v}_z = \max(|v_z| - \frac{1}{2}|g_z|, 0)$ and g_z is the gravitational acceleration. The *smash_skill* event occurs when $v > 6$ m/s.

- *topspin_skill*: Let ω_y be the angular velocity of the ball around its y-axis. The *topspin_skill* event occurs when $\omega_y > 200$ rad/s and $w_s > 0.9$.

- *backspin_skill*: occurs when $\omega_y < -100$ rad/s and $w_s < -0.9$.

Additionally, we always maintain a *default* table where all transitions are stored.

1.5 Rally - FAOC

1.5.1 FAOC optimization problem formulation

The optimization problem of FAOC minimizes the sum of quadratic jerks to compute reference trajectories for each robot joint as N_l cubic splines, each of length τ_c. Let $\dddot{u}_l$ and x_l denote the jerk and joint state (position, velocity, acceleration) at the end of each spline respectively. The maximal control invariant set, denoted by $\mathbb{X}^\infty$, is the largest subset of the feasible state space such that for any state $\boldsymbol{x} \in \mathbb{X}^\infty$ there exists a feasible control input $\dddot{u}$ satisfying $\widetilde{\boldsymbol{A}}\boldsymbol{x} + \widetilde{\boldsymbol{b}}\dddot{u} \in \mathbb{X}^\infty$. The discrete-time dynamics described by the triple integrator:

$$\widetilde{\boldsymbol{A}} = \begin{bmatrix} 1 & \tau_c & \frac{\tau_c^2}{2} \\ 0 & 1 & \tau_c \\ 0 & 0 & 1 \end{bmatrix}, \quad \widetilde{\boldsymbol{b}} = \begin{bmatrix} \frac{\tau_c^3}{6} \\ \frac{\tau_c^2}{2} \\ \tau_c \end{bmatrix}. \tag{2}$$

Subsequently, the set $\mathbb{X}^\infty$ characterizes the largest set of initial states from which the following MPC problem remains recursively feasible [6]:

$$\begin{aligned} \min_{\dddot{u}, \boldsymbol{x}} \quad & \sum_{l=0}^{N_l - 1} \frac{1}{2} \dddot{u}_l^2 \\ \text{s.t. } \boldsymbol{x}_{l+1} &= \widetilde{\boldsymbol{A}}\boldsymbol{x}_l + \widetilde{\boldsymbol{b}}\dddot{u}_l \quad \forall l \in \{0, 1, \ldots, N_l - 1\} \\ \underline{\dddot{u}} &\leqslant \dddot{u}_l \leqslant \bar{\dddot{u}} \quad \forall l \in \{0, 1, \ldots, N_l - 1\} \\ \underline{\boldsymbol{x}} &\leqslant \boldsymbol{x}_l \leqslant \bar{\boldsymbol{x}} \quad \forall l \in \{1, \ldots, N_l - 1\} \\ \boldsymbol{x}_{N_l} &\in \mathbb{X}^\infty \\ \boldsymbol{x}_0 &= \boldsymbol{x}(0) \\ \boldsymbol{x}(t_{N_l}) - \Delta\boldsymbol{x} &\leqslant \boldsymbol{x}_{N_l} \leqslant \boldsymbol{x}(t_{N_l}) + \Delta\boldsymbol{x} \end{aligned} \tag{3}$$

where $\Delta \boldsymbol{x}$ is the deviation allowed in the terminal target state $\boldsymbol{x}(t_{N_l})$ and $[\underline{\ddot{u}}, \bar{\ddot{u}}]$ and $[\underline{\boldsymbol{x}}, \bar{\boldsymbol{x}}]$ are the jerk and state limits respectively. *Ace* uses $\tau_c = 8$ ms, $N_l = 4$, resulting in a time horizon of $T = \tau_c \cdot N_l = 32$ ms. The solution of the above optimization problem is computed using the solver DAQP [7] and the resulting cubic spline is sampled at 1 kHz to produce a 32 ms trajectory segment.

1.5.2 Near Time-optimal MPC

The near time-optimal MPC problem is phrased in the same way as the optimization problem of FAOC. However, N_l is computed with a separate procedure which guarantees near time optimality, and the final state is always static corresponding to $\boldsymbol{q}_0$ computed by the prepare policy. N_l is found for each joint separately through a bisection algorithm given a range of possible values $[N_{l_{\min}}, N_{l_{\max}}]$. $N_{l_{\max}}$ is determined offline and $N_{l_{\min}}$ is analytically computed at runtime as the solution to a continuous-time time-optimal control problem for a double integrator. Using the same time discretization and kinematic limits of FAOC's optimization results in the same maximal control invariant set for the two optimization problems, which guarantees that trajectory generation is recursively feasible.

1.6 Ablation experiments

In order to investigate the importance of the different design decisions made for *Ace* we performed a series of ablation experiments. To reliably quantify the effect of each ablation we designed a hybrid benchmark that combined real world data recorded from the tournament with our own simulated physics. The shots from human players during the games reported above were recorded at the sensor level, resulting in ball trajectory data from 139 serves and 484 strokes. The hybrid benchmark replays these sensor readings to each of the policies and the response of the robot is simulated. If either (1) the sensor readings are exhausted (i.e. the robot hit the ball in the recording) or (2) the simulated racket would hit the ball, then the benchmark switches to simulated physics. The ball state at the time of this switch is based on a ground truth estimate of the ball computed offline.

The design of this hybrid benchmark allows us to replay the same data distribution, in terms of incoming shots and real world sensor readings, to different policies, and with different sensor inputs, and produce a measure of how well it would perform assuming an accurate racket model. For all ablation experiments, we trained policies with 3 different random seeds for each setting and report the return rate on the hybrid benchmark. This included training with 3 seeds for the baseline, which used the same settings as π_0 in Table S6.

The experiments are split into two groups, with the first group focusing on the importance of key features of *Ace* (Fig. S3a). The results for this group can be summarized as follows:

1. **Action Frequency:** In this experiment we tested the importance of the frequency of RL actions on performance. We therefore trained different policies with different multiples of the original RL control frequency (1x, 2x, 0.5x, 0.25x) and evaluated them on the hybrid benchmark. We found that the chosen frequency of $31.25Hz$

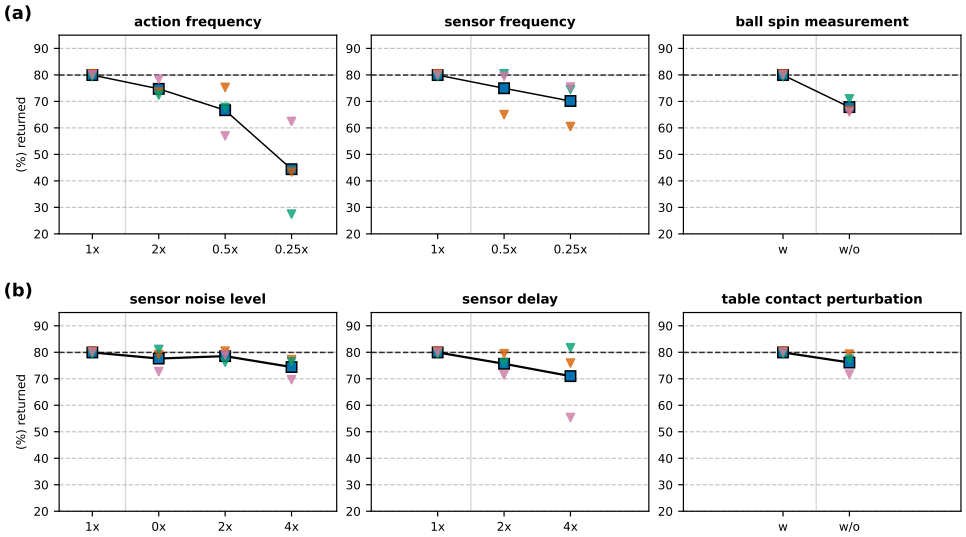

Figure S3: Results of ablations exploring (a) the importance of key features of the system and (b) the settings used to aid transfer from simulation to the real world. The x-axis represents the feature value and the y-axis represents the return rate (%) on the hybrid benchmark. For each feature value policies were trained with 3 random seeds, including for the baseline which used the same settings as π_0 in Table S6. The mean for each feature value is also reported and the dashed line indicates the mean for the baseline.

(i.e. 1x) performed better than other multiples of this frequency. Interestingly, this demonstrates that reacting faster from an RL perspective (i.e. 2x) does not guarantee better performance on the final problem. This is likely due to the increased horizon of the RL problem and the need for more effective exploration techniques.

2. **Sensor Frequency:** In this experiment we assessed the importance of high-speed sensing for the performance of *Ace*. To achieve this we decreased the sensor frequency by multiples (1x, 0.5x and 0.25x) both during training and on the hybrid benchmark. The results show that as the sensor frequency decreases the performance of the policies also steadily decreases. This highlights the importance of high-speed sensing for *Ace*.
3. **Ball Spin Measurement:** In this experiment we investigated the change in performance when a policy is trained without ball spin measurements and is evaluated on the hybrid benchmark. The result is a significant drop in performance. However the policy is still able to return $> 65\%$ of the tournament shots. This is likely due to the fact that the policy still observes histories of ball positions and is encouraged to reconstruct the spin using an auxiliary loss.

The second group of experiments focused on the settings used to aid transfer of the policies from simulation to the real world (Fig. S3b). The hybrid benchmark still allows

us to investigate such settings due to the use of replayed real world sensor readings up to the point of racket contact. The results for this group can be summarized as follows:

1. **Sensor Noise:** In this experiment we trained policies with different multiples of sensor noise (1x, 0x, 2x and 4x) and evaluated them on the hybrid benchmark. The findings show that the noise levels used for the tournament (i.e. 1x) perform best, while increasing noise levels slowly decreases the performance of the policy. Crucially, no noise (i.e. 0x) also led to worse average performance, indicating that sensor noise during training is important for handling real world sensor readings.
2. **Sensor Delay:** In this experiment we kept the sensor delays in the hybrid benchmark fixed and explored the effect of evaluating policies trained with different multiples of sensor delay (1x, 2x and 4x). As the mismatch in delays between the benchmark and the policy increased, the variance in performance increased and the average performance decreased. This highlights that in order to achieve robust and strong performance the policy should be trained with the correct sensor delays experienced in the real world.
3. **Table Contact Perturbation:** In this experiment we investigated the change in performance when a policy is trained without random table contact perturbations. When evaluated on the hybrid benchmark this resulted in a small drop in performance. This suggests that the perturbations help the policy deal with real world sensor readings and therefore maintain performance in the real world.

1.6.1 Transferability of robot dynamics model

In this experiment, we evaluate the importance of the robot actuation frequency on the transferability of the robot dynamics model from simulation to the real world. To this end, we trained two dynamics models operating at different control frequencies, 500Hz and 1000Hz, using a set of collected system identification excitation trajectories. As seen in Fig. S4, while both models exhibit low deviation between simulation and reality, the model operating at 1000Hz achieves substantially lower error than the 500Hz model. These results indicate that the higher actuation frequency significantly improves transferability, enabling a simple and fast model to achieve accurate performance.

1.7 Rally - strategy

1.7.1 Policy sampler

Heuristics.

The *heuristic policy sampler* observes the ball incoming spin along the y-axis, denoted ω_y and discretizes it into three categorical regimes based on threshold values -50 and 50 rad/s. Within each regime, a policy is selected probabilistically, guided by empirically observed return rates associated with that ω_y category in real-world deployments. The sampler further evaluates the predicted incoming post-bounce peak height and spin. If the spin and height exceed 50 rad/s and 0.6 m, respectively, a dedicated policy, specifically optimized for smashes, is triggered.

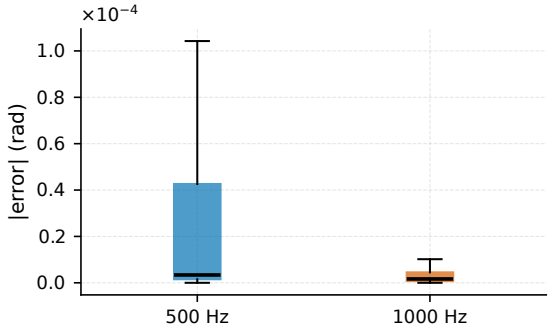

Figure S4: Absolute error in the transferability of the robot dynamics model from simulation to real world for one joint. Two dynamics models, operating at control frequencies of 500 Hz and 1000 Hz, were trained using system identification excitation trajectories. The vertical axis represents the absolute deviation between the simulated model response and the measured joint position on the physical robot over the validation set

.

Data-driven.

The *data driven sampler* operates analogously to the heuristic sampler, but is based on a supervised learning model trained on data from elite table tennis players. For a target ball landing position (y_{desired}) and a spin reward weight (w_s), the model predicts the win probability given the ball and racket state of the opponent prior to the current shot. This way, the desired pair of (y_{desired}, w_s), from a predefined set, with highest win rate is identified. The value of w_s determines the categorical regime. Then, policy sampling is performed probabilistically, guided by offline simulation of return rates associated with each candidate policy. Unlike the heuristic sampler, the data-driven approach does not deploy a specialized smash policy.

1.7.2 Prepare policy

In rallies, skilled players continuously recover towards an optimal position based on the placement of their own shot and the anticipated return of the opponent. Similarly, the robot needs to maximize its capability to arbitrarily position and orient its racket when striking the ball. For instance, the robot's ability to return the ball diminishes when the ball is too close to its base, whereas its capacity to execute an effective return increases when its arm is more extended, enabling greater reach and maneuverability.

To this end, the prepare policy is designed to set the target of the reset plan for higher dexterity in the subsequent shot by sampling from the optimal reset configuration dataset $\boldsymbol{q}_0^*$ that maximizes $\det\left(\boldsymbol{J}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})^T\right)$ given the kinematic constraints of the robot. $\boldsymbol{J}(\boldsymbol{q})$ denotes the Jacobian matrix at configuration $\boldsymbol{q}$. The determinant $\boldsymbol{J}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})^T$ is proportional to the volume of the velocity ellipsoid [8]. A larger ellipsoid volume indicates that the robot can generate higher velocities in a wider range of directions, reflecting greater dexterity at that configuration.

Table S9: Deployment of four different policy sampler strategies in games. $\pi_{\mathcal{S}}$ denotes a repertoire of policies indexed by $\mathcal{S} \subseteq \{0, \ldots, 9\}$ detailed in Table S6. Policies that are conditioned on the desired y-landing position could also afford the use of the prepare policy. Games won by *Ace* are indicated by boldface.

	Policy Sampler				**prepare**
	Fixed	Random	Heuristics	Data Driven	
Elite A	$\boldsymbol{\pi_0}$	-	-	-	**y**
	-	-	$\pi_{\{2,3,5,8,9\}}$	-	-
	$\boldsymbol{\pi_0}$	-	-	-	**y**
Elite B	$\boldsymbol{\pi_0}$	-	-	-	**y**
	-	-	$\boldsymbol{\pi_{\{2,3,5,8,9\}}}$	-	-
Elite C	$\boldsymbol{\pi_0}$	-	-	-	**y**
	π_1	-	-	-	-
	$\boldsymbol{\pi_0}$	-	-	-	**y**
Elite D†	$\boldsymbol{\pi_0}$	-	-	-	**y**
	-	-	$\pi_{\{2,3,5,8,9\}}$	-	-
	π_0	-	$\pi_{\{2,3,5,8,9\}}$	-	*
Elite E	π_0	-	-	-	y
	-	-	$\pi_{\{2,3,5,8,9\}}$	-	-
Pro 1†	$\boldsymbol{\pi_1}$	-	-	-	-
	-	-	-	$\pi_{\{4,6,7,8\}}$	-
	π_1	-	-	-	-
	-	$\pi_{\{4,6,7,8\}}$	-	$\pi_{\{4,6,7,8\}}$	-
Pro 2	π_1	-	-	-	-
	-	-	-	$\pi_{\{4,6,7,8\}}$	-
	-	$\pi_{\{4,6,8\}}$	-	-	-

*: prepare policy with π_0 was used in the first half of the game before human player requesting a time-out (point:6-5) after which the heuristic policy sampler was deployed (point: 11-7).
†: In two games (game 3 against elite player D and game 4 against pro player 1) two different policy samplers were deployed, which negatively impacted overall performance.

This approach contributed to enhance tactical performance and was deployed in 7 games, being present in 5 of the 7 total wins achieved (Table S9). As shown in Fig. S5, the distribution of ball bounce states on the robot side of the table under the prepare policy is significantly wider and more distant from the robot base. This provides greater spatial and temporal margins for the robot to respond regardless of the sampled skill. Also, the velocity distribution of the incoming balls is relatively more uniform with the prepare policy.

1.8 Serves

In this subsection we provide additional details on the serve results, followed by details on the MPC and the genetic algorithm used to create the serve library.

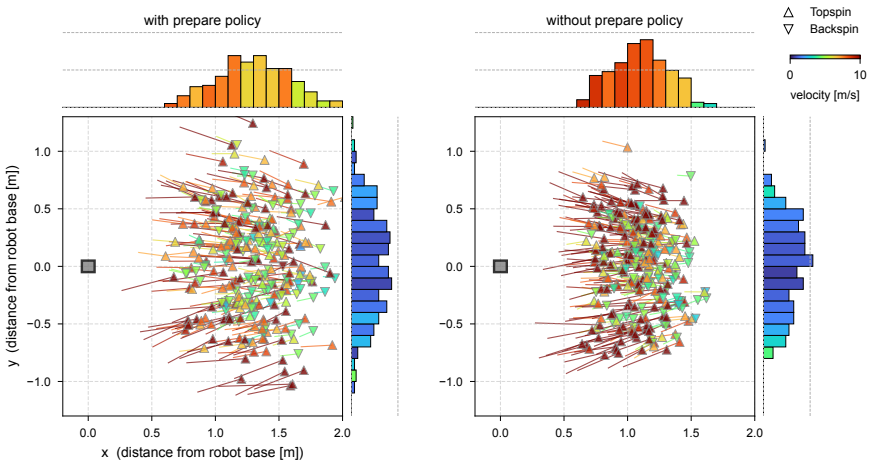

Figure S5: Distribution of ball bounce states on the table (in the robot base frame) with and without the prepare policy in games. The robot base is depicted at the origin. Upward- and downward-pointing triangles denote bounce events with topspin and backspin on the robot side of the table, respectively. Superimposed quivers indicate the horizontal ball velocity and are color-mapped with relative length according to velocity magnitude (refer to the colorbar). Marginal histograms along the top and right axes represent the marginal distribution of bounce locations, with bin colors reflecting the mean marginal velocity within each bin region.

1.8.1 Serves used during the match

Fig. S6 shows a sample of 100 ball trajectories during the matches. The serves in the library have been selected to cover a wide variety of landing points on the opponent's side of the table (indicated by the circle). On average, 13 different serves were executed per match, with the same serve being used an average of 1.88 times throughout a match.

For matches against pros, serves were selected from the library based on serve scores obtained initially in dedicated assessment sessions and subsequently updated throughout matches as a proxy to the serve's winning probability. In the assessment session, human experts scored the "difficulty" of each serve by repeatedly receiving the robot serve as many times as needed to study the serve strengths and weaknesses and to come up with a difficulty score from 1 to 100. Serves with a score above 50 were included in the library used in matches. During a match, the initial scores were normalized as prior weights and the next serve was selected via weighted random sampling, giving higher-probability selection to serves with higher scores. We then updated the score of each serve at the end of each point that started with it (increasing the score in the case of a winning rally, decreasing otherwise) and re-normalized as the winning probability for the next selection.

For elite matches, we adopted serve spin dissimilarity rather than subjective scoring as the governing strategy. The classification of serves by spin is achieved by recording,

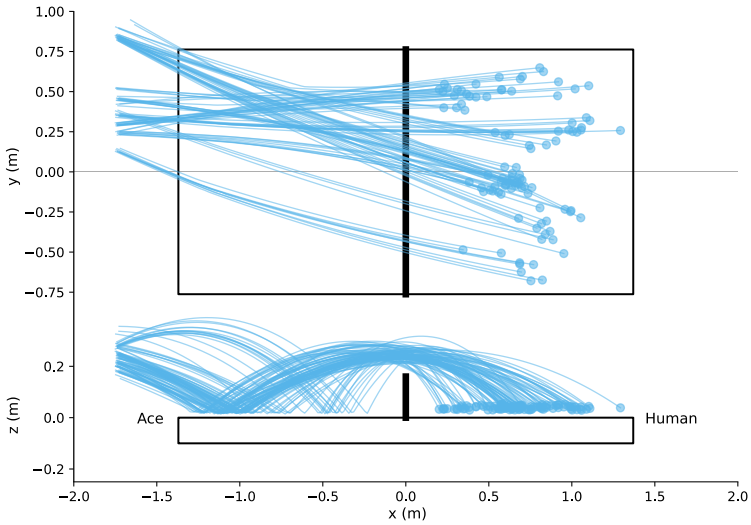

Figure S6: A sample of 100 ball trajectories resulting from *Ace* serves during the matches. The circles indicate the landing point of the ball on the opponent's side. The library consisted of 15 and 13 pre-selected serves against the elite and pro players, respectively.

for each of the serves, their spin directions and magnitudes when building the library of serves. At the start of a match, the strategy groups the serves based on their recorded spin values, which serve as the basis for defining dissimilarity and are qualitatively categorized in cork, top, back, side spin, and their combinations. This strategy does not have the concept of winning probability. Instead, by selecting the next serve as one that has not been used recently while also belonging to a different class of spin, it attempts to make it difficult for players to learn how to react to a particular serve, while also providing an element of surprise. Direct serve points significantly increased from four (pro players) to 16 (elite players) using the dissimilarity as strategy, although a definitive comparative conclusion is difficult due to the differing skill levels between the two categories of players.

Fig. S7 shows the velocity magnitudes of the serves executed by *Ace* and human players. Across 194 serves executed by *Ace*, the average linear velocity of the ball was 6.94 $\pm$ 1.28 m/s (mean $\pm$ standard deviation) and the average angular velocity magnitude was 199.53 $\pm$ 68.64 rad/s. In comparison, for elite players (110 serves), these averages were 7.28 $\pm$ 1.35 m/s and 214.07 $\pm$ 87.94 rad/s, respectively. For professional players (68 serves), the corresponding average velocities were 7.30 $\pm$ 1.22 m/s and 230.27 $\pm$ 55.42 rad/s. From the figure it is observed that in both cases of human and robot serves, aces (shown by the large circles and triangles) and rally points (smaller markers with black edges) are distributed over the entire ranges of linear and angular velocities. Fast serves with high spin do not necessarily led to more points.

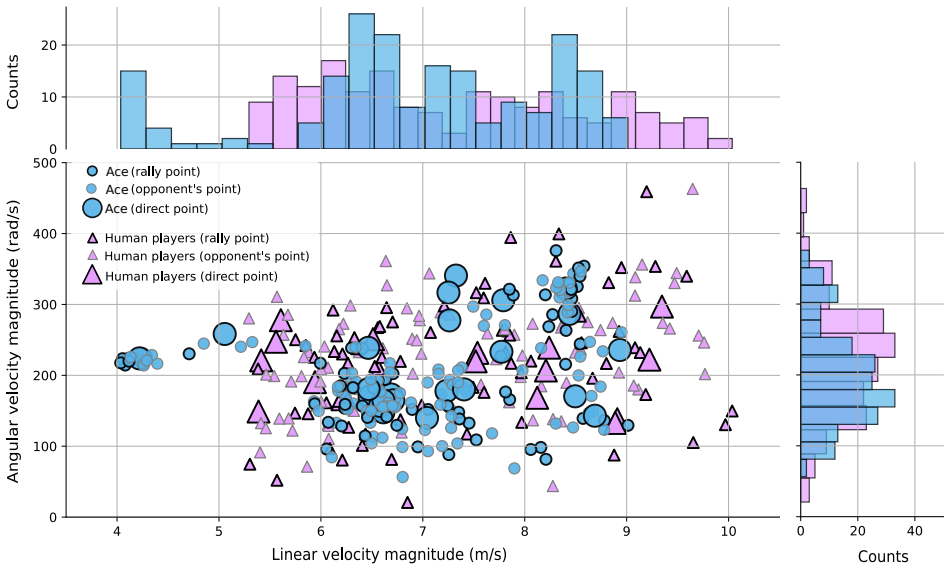

Figure S7: Magnitudes of the linear and angular velocities of *Ace* and human players during serves. The velocity magnitudes are largely comparable, with *Ace*'s velocity distribution being slightly slower by 1 m/s. Direct point indicates points obtained by the serving side when the opponent failed to return the ball. Rally point indicates that the serving side won during the rally.

1.8.2 MPC for serve execution

The motion planner used for serve strikes is a nonlinear MPC that solves the following parametric, nonconvex control problem:

$$
\begin{aligned}
f^*(\boldsymbol{\xi}_s) = f^*\big(\boldsymbol{u}_0, \dot{\boldsymbol{u}}_0, \ddot{\boldsymbol{u}}_0, \boldsymbol{x}_0, \tau, \boldsymbol{p}_\tau, \boldsymbol{n}_\tau, \boldsymbol{v}_\tau, \tau_f, \boldsymbol{u}_{\tau_f}\big) = \min\ & \tfrac{1}{2}\dddot{\boldsymbol{u}}^T\dddot{\boldsymbol{u}} + \hat{\lambda}_p\epsilon_p + \hat{\lambda}_n\epsilon_n + \hat{\lambda}_v\epsilon_v \\
\text{s.t. } & (\boldsymbol{z}, \boldsymbol{q}_\tau, \dot{\boldsymbol{q}}_\tau) \in \mathbb{P}(\boldsymbol{u}_0, \dot{\boldsymbol{u}}_0, \ddot{\boldsymbol{u}}_0, \boldsymbol{x}_0, \tau, \tau_f, \boldsymbol{u}_{\tau_f}) \\
& \Delta\boldsymbol{p} = \boldsymbol{p}(\boldsymbol{q}_\tau) - \boldsymbol{p}_\tau \\
& -\boldsymbol{n}_\tau^T\boldsymbol{n}(\boldsymbol{q}_\tau) \leqslant -\cos(\Delta\phi_\tau) + \epsilon_n \\
& \Delta\boldsymbol{v} = \boldsymbol{J}_v(\boldsymbol{q}_\tau)\dot{\boldsymbol{q}}_\tau - \boldsymbol{v}_\tau \\
& \|\Delta\boldsymbol{p}\|_2 \leqslant \Delta p_\tau^2 + \epsilon_p \\
& \|\Delta\boldsymbol{v}\|_2 \leqslant \max\left\{\Delta v_{a,\tau}^2, \Delta v_{r,\tau}^2\|\boldsymbol{v}_\tau\|_2\right\} + \epsilon_v \\
& \epsilon_p, \epsilon_n, \epsilon_v \geqslant 0
\end{aligned}
\tag{4}
$$

As a result of the optimization, we obtain a sequence of jerk values over a control horizon for each of the N_q joints. Through triple integration starting at the initial spline state $[\boldsymbol{u}_0, \dot{\boldsymbol{u}}_0, \ddot{\boldsymbol{u}}_0] \in \mathbb{R}^{3N_q}$, we generate cubic reference splines for both the *attack phase*, which ends at the hitting time $\tau > 0$, and the *reset phase*, which ends at the

terminal time $\tau_f > \tau$ at the stationary terminal state $\boldsymbol{u}_{\tau_f} \in \mathbb{R}^{N_q}$. Before supplying these reference trajectories to the joint controllers, we sample them at 1 kHz.

The control horizon, and thus the size of (4), is kept constant at 32 steps. This implies that the continuous-time dynamics must be newly discretized each time the terminal time τ_f changes. However, by appropriately scaling the states and inputs, we can avoid these computations and only need to update the left- and right-hand side vectors of the constraints.

The vectors $\boldsymbol{q}_\tau \in \mathbb{R}^{N_q}$ and $\dot{\boldsymbol{q}}_\tau \in \mathbb{R}^{N_q}$ denote the *actual* position and velocity of the joints at the hitting time τ according to the joint models. The decision variables are constrained by the abstract polytopic set $\mathbb{P}$, which results from the state update equations, the bounds on the states and inputs over the control horizon, and the effect of spline sampling. All of these constraints can be formulated as affine equality and inequality constraints on the decision variables. Physical limitations motivate the use of bounds on states (position, velocity and acceleration), whereas bounded jerk ensures smooth motion and reduces excessive strain on moving components and motors. These bounds also help to keep the joint control loops in their linear regimes, ensuring that the joint models remain accurate.

At hitting time τ, we require the center of the racket $\boldsymbol{p}(\boldsymbol{q}_\tau)$ to be "close" to the target position $\boldsymbol{p}_\tau \in \mathbb{R}^3$. However, we allow for a user-defined 2-norm error $\Delta p_\tau > 0$. Similarly, the racket normal $\boldsymbol{n}(\boldsymbol{q}_\tau)$ should be close to the target normal $\boldsymbol{n}_\tau$ with an acceptable angular error of $\Delta\phi_\tau > 0$. Due to the racket's rotational symmetry around its center, only the racket surface normal is used to specify racket orientation. Note that the functions $\boldsymbol{p}(\cdot)$ and $\boldsymbol{n}(\cdot)$ are obtained from the robot's forward kinematics. For translational racket speed, we aim for the target speed $\boldsymbol{v}_\tau \in \mathbb{R}^3$, but allow some absolute or relative 2-norm error, defined by the user-specified values $\Delta v_{a,\tau} > 0$ and $\Delta v_{r,\tau} > 0$.

To avoid an empty feasible set and facilitate solver convergence, we adopt an exact penalty-type formulation of the MPC problem. This involves introducing non-negative slack variables ϵ_p, ϵ_n and ϵ_v and penalizing them linearly using positive penalty parameters $\hat{\lambda}_p$, $\hat{\lambda}_n$ and $\hat{\lambda}_v$. These parameters are chosen large enough to ensure exact penalty-type behavior. The quadratic term in the objective penalizes the sum of quadratic jerks over the control horizon.

Finally, as with the rally, we require the robot to avoid hitting any obstacles or itself. Collision avoidance constraints would unnecessarily complicate the formulation in (4), greatly increasing the computational cost. Instead, safety is ascertained *a posteriori* in a manner similar to the rally control. Since safe trajectories cannot be taken for granted, the serve GA (cf. Section 1.8.3) must learn to avoid collisions by carefully selecting the parameters of (4).

We formulate and solve (4) using Artelys KNITRO [9]. The following values for the tolerances are used: $\left(\Delta p_\tau, \Delta\phi_\tau, \Delta v_{a,\tau}, \Delta v_{r,\tau}\right) = (5\text{ mm}, 0.5\text{ deg}, 0.01\text{ m/s}, 0.01)$. For successful problems, when the optimal constraint violations satisfy $\epsilon_p^* \approx 0.0$ m, $\epsilon_n^* \approx 0.0$ deg, and $\epsilon_v^* \approx 0.0$ m/s, a solution is typically found in less than 15 ms on an Intel I9-13900KF CPU (single threaded).

1.8.3 Serve genetic algorithm

Algorithm 2, details the implementation of a GA to optimize the parameters of our MPC planner $\boldsymbol{\xi}_s$ (cf. Section 1.8.2) to learn how to serve in simulation. Running a serve $\boldsymbol{u}_s(t)$ in our simulator $\mathcal{S}_s$ produces a ball trajectory $\boldsymbol{x}_b(t) \triangleq \mathcal{S}_s(\boldsymbol{u}_s(t))$. The proficiency level of $\boldsymbol{u}_s(t)$ can be modeled as a scalar value f, calculated using a parameterized fitness function $\mathcal{F}_\theta(\boldsymbol{u}_s(t), \boldsymbol{x}_b(t))$. The function $Converged(\boldsymbol{F}_s, \epsilon)$ checks if the best fitness value meets the convergence criteria ϵ, which means that it has not increased significantly in the last few iterations. $Recombine(\boldsymbol{\Xi})$ uses the best parameterizations $\boldsymbol{\xi}_s^i$ to generate a new *population* using both genetic *cross-over* and *random mutations*. This is the standard procedure used in GA's, and we use PyGAD for the implementation [10].

Table S10: Compute hardware and robot power consumption

Category	**Device**	**Note**	**Power consumption(W)**
GCS PC	CPU	AMD Ryzen Threadripper PRO 5965WX	381
	GPU 1	NVIDIA GeForce RTX 4090	
	GPU 2	NVIDIA RTX 4000 Ada Generation	
Ball detection	FPGA	9 FPGAs in total	64
Perception and control PC	CPU	Intel Core i9-13900KF	83
Robot	Robot	Actuators	10'700 (rated)
Total power consumption			**11'228**

Algorithm 2 Generation of serves with GA

Input: $\boldsymbol{u}_{\text{toss}}(t)$, t_{lift}, $\mathcal{F}_\theta$, $\mathcal{S}_s$, N_ξ, ε
Output: Incumbent $\boldsymbol{u}_s^*(t)$ and its simulated ball trajectory $\boldsymbol{x}_b^*(t)$.

function GetServe($\boldsymbol{u}_{\text{toss}}(t), \boldsymbol{\xi}_s$)
  $\boldsymbol{u}_{\text{strike}}(t) \leftarrow \text{MPC}(\boldsymbol{u}_{\text{toss}}(t), \boldsymbol{\xi}_s)$
  if $\exists\, \boldsymbol{u}_{\text{strike}}(\text{t})$ **then**
    return $\left[\boldsymbol{u}_{\text{toss}}(t) \parallel \boldsymbol{u}_{\text{strike}}(t)\right]$
  else
    return [ ]
  end if
end function

function SimulateServe($\boldsymbol{u}_s(t), \mathcal{F}_\theta, \mathcal{S}_s$)
  $\boldsymbol{x}_b(t) \leftarrow \mathcal{S}_s(\boldsymbol{u}_s(t))$
  $f \leftarrow \mathcal{F}_\theta(\boldsymbol{u}_s(t), \boldsymbol{x}_b(t))$
  return $f, \boldsymbol{x}_b$
end function

procedure Evolve($\boldsymbol{u}_{\text{toss}}, t_{\text{lift}}, \mathcal{F}_\theta, \mathcal{S}_s, N_\xi, \varepsilon$)
  $\boldsymbol{\Xi} \leftarrow \mathit{RandomInit}()$
  $\boldsymbol{U}_s = [\,] \times N_\xi$
  $\boldsymbol{F}_s = [\,] \times N_\xi$
  while not $\mathit{Converged}(\boldsymbol{F}_s, \varepsilon)$ **do**
    for $i \in \{1, ..., N_\xi\}$ **do**
      $\boldsymbol{U}_s[i] = \text{GetServe}(\boldsymbol{u}_{\text{toss}}(t),\, \boldsymbol{\Xi}[i])$
      $\boldsymbol{F}_s[i] = \text{SimulateServe}(\boldsymbol{U}_s[i], \mathcal{F}_\theta, \mathcal{S}_s)[1]$
    end for
    $\boldsymbol{\Xi} \leftarrow \mathit{Recombine}(\boldsymbol{\Xi})$
  end while
  $\boldsymbol{u}_s^*(t) = \boldsymbol{U}_s[\mathit{Argmax}[\boldsymbol{F}_s]]$
  $\boldsymbol{x}_b^*(t) = \text{SimulateServe}(\boldsymbol{u}_s^*(t), \mathcal{F}_\theta, \mathcal{S}_s)[2]$
end procedure

Additional References

1. Hülsdünker, T., Ostermann, M. & Mierau, A. The Speed of Neural Visual Motion Perception and Processing Determines the Visuomotor Reaction Time of Young Elite Table Tennis Athletes. *Frontiers in Behavioral Neuroscience* **13.** https://www.frontiersin.org/journals/behavioral-neuroscience/articles/10.3389/fnbeh.2019.00165 (2019).
2. He, K., Zhang, X., Ren, S. & Sun, J. Deep residual learning for image recognition. in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), 770–778.
3. Uhlenbeck, G. E. & Ornstein, L. S. On the theory of the Brownian motion. *Physical review* **36,** 823 (1930).
4. Lillicrap, T. P. *et al.* Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).
5. Kompella, V., Walsh, T. J., Barrett, S., Wurman, P. & Stone, P. Event tables for efficient experience replay. *arXiv preprint arXiv:2211.00576* (2023).
6. Kerrigan, E. C. & Maciejowski, J. M. Invariant sets for constrained nonlinear discrete-time systems with application to feasibility in model predictive control. in *Proceedings of the 39th IEEE conference on decision and control* **5** (2000), 4951–4956.
7. Arnström, D., Bemporad, A. & Axehill, D. A Dual Active-Set Solver for Embedded Quadratic Programming Using Recursive LDL^T Updates. *IEEE Transactions on Automatic Control* **67,** 4362–4369 (2022).
8. Yoshikawa, T. Manipulability of robotic mechanisms. *The international journal of Robotics Research* **4,** 3–9 (1985).
9. Byrd, R. H., Nocedal, J. & Waltz, R. A. Knitro: An Integrated Package for Nonlinear Optimization (eds Di Pillo, G. & Roma, M.) 35–59 (Springer US, Boston, MA, 2006).
10. Gad, A. F. Pygad: An intuitive genetic algorithm python library. *Multimedia Tools and Applications,* 1–14 (2023).