Air Hockey Challenge 2023: Air-HocKIT Team Report

Mustafa Enes Batur[3] **Vincent de Bakker**[3] **Atalay Donat**[3] **Ömer Erdinç Yagmurlu** [3]
Marcus Fiedler[3] **Zeqi Jin**[3] **Dongxu Yang**[3] **Hongyi Zhou**[2,3] **Xiaogang Jia**[1,2,3] **Onur Celik**[1,3,4]
Fabian Otto[1,3] **Rudolf Lioutikov**[2,3] **Gerhard Neumann**[1,3,4]
[1]Autonomous Learning Robots, KIT, Germany
[2]Intuitive Robots Lab, KIT, Germany
[3]Karlsruhe Institute of Technology, Germany
[4]FZI Research Center for Information Technology

Abstract

This work shows how we deploy deep reinforcement learning (RL) methods in a challenging air-hockey environment. We analyze the sub-tasks present in the environment and propose a composite agent that consists of five learning-based sub-agents governed by a hand-crafted state machine. Additionally, we analyze the physical constraints and suggest careful reward-shaping to overcome those. Finally, we show the techniques that we used to reduce the sim-to-real gap challenges modeled in the evaluation environment.

1 Introduction

The Robot Air Hockey Challenge 2023, a competition under the NeurIPS competition track, is primarily organized by TU Darmstadt with a focus on bridging the sim-to-real gap in learning-based robotic control. The challenge offers a unique platform to test and refine robotic systems in an air-hockey environment that closely replicates real-world perturbations, including observation noise, intermittent puck tracking loss, and imperfect controller responses, all of which add layers of complexity akin to real-life scenarios. Our participation in this challenge centers around the deployment of advanced reinforcement learning (RL) methods to develop a proficient air-hockey playing agent. This paper outlines our journey through the challenge, detailing the strategies we employed and the obstacles we overcame. Our approach, which hinges on the principles of reinforcement learning, showcases the potential of RL in navigating and mastering the dynamic and unpredictable nature of the air-hockey game. We delve into the specifics of our methodology, the design of our RL agents, the design of an agent-selecting state machine, and the techniques used to reduce the sim-to-real gap.

2 Reinforcement Learning Methods for Continuous Control

In addressing the challenges presented by the challenge, we focused on the application of the deep RL algorithm. Central to our approach is the Proximal Policy Optimization (1). PPO is renowned for its simplicity and effectiveness in on-policy learning of continuous control tasks. In implementing PPO, we leveraged the robust and widely-used framework provided by *Stable-Baselines 3* (2).

2.1 State-Action Space Modifications

State Space. The provided observations of the air-hockey environment include joint angles q and velocities $\dot{q}$, puck positions $p_p = [x_{\text{puck}}, y_{\text{puck}}]$ and velocities $\dot{p}_p = [\dot{x}_{\text{puck}}, \dot{y}_{\text{puck}}]$ in the x-y plane, and the yaw angle ψ and radial velocity $\dot{\psi}$ of the puck, as well as the opponent agent's end-effector position. We extend the observations by end effector Cartesian positions $p_{ee} = [x_{ee}, y_{ee}, z_{ee}] = f(q)$

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

where f is the forward kinematics, such that the agent does not have to learn the mapping by itself. Furthermore, we include the joint acceleration $\ddot{q}$ in the observation, which affects the interpolated torque at each step because of the third-order interpolation (which we will discuss later). Lastly, we remove the yaw angle ψ, radial velocity $\dot{\psi}$ of the puck, and the opponent end-effector position from the observation.

Action Space. The challenge setup involves differing control and simulation frequencies. While the high-level control actions provided by the agent are limited to 50Hz, the lower-level control operates at 1000Hz. Consequently, interpolation of 20 sub-actions between successive control actions is necessary. We interpolate the agent with cubic splines to match the frequency discrepancy.

As a natural first step, we initially experimented with directly predicting both the desired joint positions and velocities. However, our empirical results suggest that it is difficult for the PPO agent to correctly capture the correlation between position and velocity predictions, even after 100 million environment interactions, and this inconsistency leads to compromised performance and severe constraints violation. Subsequently, we restrict the agent to predict a scalar value per joint and then use numerical integration to obtain the desired joint positions and velocities. We tried using the RL agents to predict the joint velocity, acceleration, and jerk. Our experiments have revealed that predicting the desired joint acceleration results in the best performance among other representations that we tested. Another worth-noting observation is that through our experiment, we did not observe an advantage in controlling the last joint of the robot, which is responsible for the rotation of the end-effector. Therefore we fixed this joint to reduce the search space for RL policy.

2.2 Composite Agent

The composite agent was constructed by integrating five independent PPO models, each specialized in handling specific scenarios: hit, fast defend, slow defend, close prepare, and far prepare. Governed by a hand-crafted state machine (see Appendix A), the selection of an agent at each time step was dictated by the current observation. An issue surfaced during model transitions: switching from one model to another after the robot had moved resulted in notably decreased performance. We attribute this to the shift of the initial state distribution. To mitigate the initial state distribution gap, we implemented a Jacobian-based reset agent to recover the robot to its original joint configuration before switching to a new agent. This adjustment alleviated the performance degradation observed during transitions. Each sub-agent was trained separately with a customized reward function. In the following, the main ideas behind each reward function are outlined.

Hit Task. The reward function for the hitting model consists of three separate stages. Before the robot hits the puck, moving the end effector towards the puck's position is rewarded. Following contact between the end effector and the puck, a larger reward, linearly scaled with the puck's x-velocity, is provided. Lastly, a substantial reward is granted for scoring a goal, multiplied by the puck's velocity and an additional base reward. The scoring reward is several magnitudes larger than all the previous step rewards, which becomes the main objective of the model after sufficient training time. The reward for the hitting agent is defined as

$$r_{hit} = \begin{cases} \max(0, \frac{p_p - p_{ee}}{|p_p - p_{ee}|} \cdot v_{ee}) & \text{if } |v_p| < 0.25 \text{ and } p_{p,x} < 0, \\ 10 \cdot |v_p| & \text{if hit,} \\ 2000 + 5000 \cdot |v_p| & \text{if scored,} \end{cases} \tag{1}$$

where p_p and v_p represent the puck position and velocity, respectively, the p_{ee} and v_{ee} states the end-effector position and velocity.

Defend Task. The two defend scenarios are separated by the incoming puck's velocity. The reward calculation for the slow defend strategy involves two parts. In the first part of the reward in eq. (2), a modest positive reward is granted when the end-effector contacts the puck for the first time. The value consists of a constant term and an exponential bonus term to encourage smaller puck velocity after contact. The reward for the slow defend agent is defined as

$$r_{\text{defend\_slow}} = \begin{cases} 30 + 100^{1-0.25|v_p|} & \text{if } v_p > -0.2 \text{ and ee touches puck for the first time,} \\ \cdots + 70 & \text{if } |v_p| < 0.1 \text{ and } -0.7 < p_{p,x} < -0.2 \text{ and } t = T, \\ 0.01 & \text{otherwise.} \end{cases} \tag{2}$$

We used a binary reward for the fast-defend agent. The algorithm receives a negative reward with a value of -100 for being scored by the opponent. The reward remains 0 for all the other cases.

Prepare Task. The two preparation scenarios are distinguished by the puck's proximity to our goal along the x-direction. For the close preparation, the agent is rewarded for moving the puck to a pre-defined target position, e.g., $(-0.5, 0)$ in our case, plus a small reward for moving the end-effector towards the puck. Additionally, a large reward of 2000 is granted when the success criterion is fulfilled. The reward for close preparation is defined as

$$r_{\text{proximity}} = \begin{cases} \max(0, \frac{p_p - p_{ee}}{|p_p - p_{ee}|} \cdot v_{ee}) & \text{if } |v_p| < 0.25 \text{ and } p_{p,x} < 0, \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

$$r_{\text{bonus}} = \begin{cases} 2000 & \text{if } |v_p| < 0.5, -0.65 < p_{p,x} < -0.35 \text{ and } -0.4 < p_{p,y} < 0.4, \\ 0 & \text{otherwise,} \end{cases} \tag{4}$$

$$r_{\text{close\_prepare}} = r_{\text{proximity}} + r_{\text{bonus}} + 10 \max(0, \min(0.5, \frac{(-0.5, 0)^T - p_p}{|(-0.5, 0)^T - p_p|} \cdot v_p)). \tag{5}$$

In contrast, the far prepare agent aims solely to return the puck to the opponent while avoiding faults. This approach stems from our observation that the PPO algorithm encounters challenges when exploring action sequences requiring the end effector to first maneuver to the puck's back-side for preparation. As a result, the reward function eq. (6) is identical to the hit reward with the caveat that the large reward is bound to getting the puck far enough into opponent territory without any puck velocity bonus, upon which the episode ends. The reward function for far preparation is given as

$$r_{\text{far\_prepare}} = \begin{cases} \max(0, \frac{p_p - p_{ee}}{|p_p - p_{ee}|} \cdot v_{ee}) & \text{if } |v_p| < 0.25 \text{ and } p_{p,x} < 0, \\ 3000 & \text{if } p_{p,x} > 0.2, \\ 10 \cdot |v_p| & \text{otherwise.} \end{cases} \tag{6}$$

2.3 Dealing with the Challenge Constraints

The challenge imposes several constraints on the robot to ensure that the agents operate within the limitations of the real hardwares. The agent needs to respect limits on the joint angles and velocities (7), and the robot should not intersect the table (8). Additionally, the end-effector should stay inside the table's boundaries (9) and close to the table surface (10).

$$q_l < q < q_r, \quad \dot{q}_l < \dot{q} < \dot{q}_r \tag{7}$$

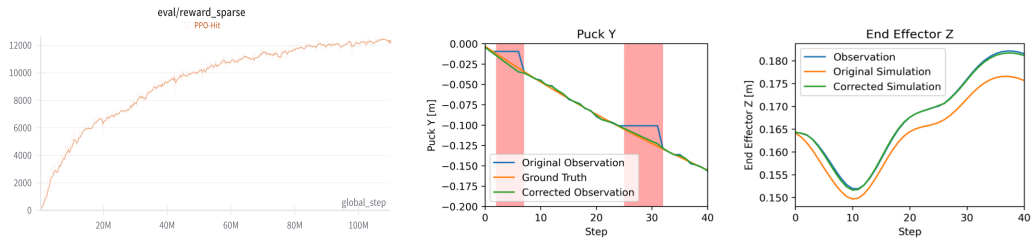
$$z_{\text{elbow}} > 0.25, \quad z_{\text{wrist}} > 0.25 \tag{8}$$

$$l_x < x_{ee}, \quad l_y < y_{ee} < u_y \tag{9}$$

$$h_{\text{table}} - h_{\text{tolerance}} < z_{ee} < h_{\text{table}} + h_{\text{tolerance}} \tag{10}$$

The evaluation criteria heavily weigh the extent to which agents comply with the established constraints. Given this, strict adherence to these constraints becomes important, even at the cost of potential performance reductions. In light of these considerations, we dedicated considerable time to fine-tuning the model to ensure that it consistently produces violation-free actions. Our initial approach involved straightforwardly penalizing the agent with a substantial negative reward for any steps that violated the constraints. However, this approach revealed a significant flaw during experiments. We observed that once the agent entered a state of constraint violation, it became exceedingly difficult for the agent to recover, leading to a series of negative rewards for the remainder of the episode. As a result, the agents tended to learn a counter-productive policy of always staying idle, as remaining stationary was more rewarding than attempting to act.

To tackle the challenge of constraint violations while keeping the policy from being "lazy," our approach involves terminating an episode immediately if the agent commits any violation. This method effectively addresses the issue of accumulating penalties due to the agent's inability to recover from a violating state. Furthermore, this approach does more than prevent the accrual of penalties; it actively promotes the generation of actions that adhere to the constraints. This is achieved by assigning positive rewards to each step where the agent successfully operates within the set constraints, encouraging the agent to engage in violation-free behavior consistently.



(a) (b) (c)

Figure 1: **(a) Leraning curve** of the PPO agent. **Reduction** of **(b) observation noise** and **(c) model and controller mismatch**. (b) shows the observed (blue), true (orange), and corrected (green) puck y position of one episode during evaluation. The red bands indicate loss of tracking events. Similarly, (c) shows the observed (blue), simulated (orange), and corrected (green) end effector z position.

2.4 Dealing with the Sim-to-Real Gap

The challenge models a sim-to-real gap by modifying the observations, model dynamics, and controller characteristics during evaluation. The evaluation environment is not accessible to the users. We investigate these modifications by analyzing the observations during the evaluation and adjusting our training environments to improve the agent's robustness against unseen changes.

Observation Noise. The observation noise during evaluation consists of additive noise on the puck positions and velocities and loss of puck tracking. The latter means that the observed puck position stays constant and hence, the observed puck velocity is zero. This is a major change for the agent as this case does not appear during training at all. Both effects are shown in Figure 1b. We replicate both the additive noise and loss of tracking in our training environment. We approximate the additive noise component with a Gaussian distribution that is estimated with maximum likelihood from observed and smoothed puck trajectories. Additionally, we apply the Kalman filter to the observations (3) and provide the estimated mean position as observation to the agent. Additionally, we discard the observed puck position and provide the estimated positions in case tracking of the puck is lost. We estimate the puck's velocity with finite differences of the estimated puck position. Our resulting correction in Figure 1b is shown in green.

Model dynamics and controller characteristics. For the same initial conditions, the observed robot movements during evaluation differ from the movement during the training environment (see Figure 1c). In our recorded data, this difference can be up to 1cm, which is critical for tight constraints such as staying within 2cm of the table surface. This mismatch can be explained by differences in the model and controller parameters, including masses, friction coefficients, damping coefficients, and proportional and differential gains of the controller. We aim to estimate the correct values of these entities by framing it as a black-box optimization problem. The objective is to minimize the differences between observed and simulated joint movements. We solve this problem using the Covariance Matrix Adaptation Evolution Strategy algorithm (4). Figure 1c (green curve) shows the movement after updating the model and controller parameters, which achieves a reduction of the root-mean-squared error of the end-effector position by 80.5% .

3 Conclusion

In this work, we have shown how we can train RL agents to solve the different tasks in the robot air hockey game. A key aspect of our approach was the selection of the state and action spaces, coupled with carefully crafting stage-specific reward functions. These adaptations proved essential for simultaneously accomplishing the game's targets and adhering to its inherent physical constraints. Additionally, we have presented how we dealt with the sim-to-real gap. Interesting future work would include training policies that can learn highly multi-modal policies such that the agent can choose between different strategies and therefore be less predictable to the opponent.

References

[1] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[2] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[3] R. E. Kalman, "A new approach to linear filtering and prediction problems," 1960.

[4] N. Hansen, "The cma evolution strategy: A tutorial," 2023.

A State Machine Definition

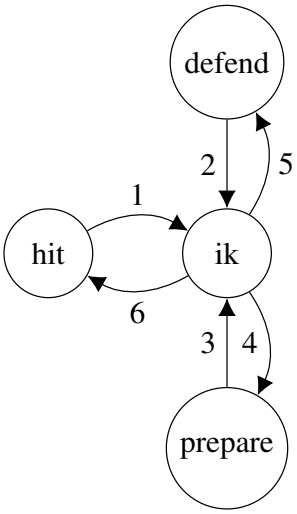


Figure 2: **State Machine of the composite agent.** The figure visualizes the different states of the the composite agent. In states *hit, defend, prepare* the hitting, defending and preparation agent is activated respectively. In the *ik* state the agent is supposed to go back to the initial joint position from which it can transition to the different states. The conditions for transitioning are shown in Table 1.

Condition	Expression
1	$\Delta p_{p,x} = p_{p,x} - 1.51$ $(\Delta p_{p,x} > -0.2) \vee (\Delta p_{p,x} + \frac{1}{2} v_{p,x} > -0.2) \vee (p_{p,x} \leq$ $p_{ee,x}) \vee (\lvert p_{p,y} \rvert > m) \vee \lvert p_{p,y} + 0.75 v_{p,y} \rvert > m$
2	$(v_{p,x} > -0.2) \vee (p_{p,x} < p_{ee,x})$
3	$\lvert p_{p,y} \rvert < 0.41 \vee p_{p,x} > -0.2$
4	$[(\Delta p_{p,x} < -0.2) \wedge (\max(\lvert v_{p,x} \rvert, \lvert v_{p,y} \rvert) < 0.05)] \wedge$ $[(\Delta p_{p,x} \leq -0.8) \vee \lvert p_{p,y} \rvert > m]$
5	$[((\Delta p_{p,x} < 0.3) \wedge (v_{p,x} < -0.5)) \vee (v_{p,x} < -1.5)] \wedge (p_{ee,x} < p_{p,x})$
6	$[(\Delta p_{p,x} < -0.2) \wedge (\Delta p_{p,x} + v_{p,x} < -0.2)] \wedge (v_{p,x} <$ $0.5) \wedge (\lvert v_{p,y} \rvert < 0.5)$ $\wedge \neg [(\lvert p_{p,y} \rvert > m) \vee (\lvert p_{p,y} + 0.75 v_{p,y} \rvert > m)] \wedge (\Delta p_{p,x} + 0.75 v_{p,x} >$ $-0.8)$

Table 1: **State transition conditions for the state machine in Fig. 2.**

B List of Hyperparameters

The hyperparameters that we have set during training are listed in table 2.

Hyperparameter	Value
Number of environments	40
Number of steps	512
Batch size	512
Learning rate	$5 \cdot 10^{-5}$
Gamma [Defend]	1
Gamma [Hit, Prepare]	0.99
Number of epochs	10
Network architecture	[64, 64]

Table 2: List of hyperparameters.