High-Level AirHockey Control via Deep and Rule-based Reinforcement Learning

RL3 Polimi
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano
Piazza Leonardo da Vinci, 32, Milan, Italy

1 Introduction

In this document, we present the applied approach in applying Reinforcement Learning (RL) for the development of the robotic AirHockeyAgent. In particular, we address the problem by employing Deep RL techniques and RL Optimized Rule-Based Controllers. Indeed, we train several low-level agents to perform some basic tasks in the game, and combine them via a rule-based controller to play the whole game.

In what follows, we present how we map the AirHockey environment to a Markov Decision Process (MDP) the employed Deep RL techniques, the developed Rule-Based controllers optimized via RL, and how we treat the noise in the evaluation environment.

2 AirHockey as an MDP

In this section, we present how we model the AirHockey environment as an MDP by defining the state-action space and reward functions employed.

State Space: We slightly modify the observation provided by the original environment. Indeed, we decided to discard the rotation-axis elements of the original observation, add the x and y component of the end effector computed with forward kinematics and add a flag indicating whether a collision happened between the end-effector and the puck. The resulting state space is then scaled between -1 and 1.

$$s = [\mathbf{p}_{\text{puck}}\ \dot{\mathbf{p}}_{\text{puck}}\ \mathbf{q}\ \dot{\mathbf{q}}\ \mathbf{p}_{\text{ee}}\ \dot{\mathbf{p}}_{\text{ee}}\ collision\_flag]$$

To smooth out the noise in the puck position and velocity observation, we employed a simplified Kalman Filter [Welch, 2020]. In order to deal noise in the joint positions and velocities, we directly use the agents actions in the previous step and ensure the requested robot poses are valid.

Action Space: We employ a high-level control setting where the agent controls either acceleration of the joints or directly the end-effector position which are then converted to joint position and velocities.

Reward function:: The designed *reward* is task-dependent, given that we train low level policies for each low level task. Moreover, we shape the reward function instead of providing a sparse reward for the completion of these tasks. Being that this reward shaping greatly affects the performance of the learning algorithm, we devise different reward functions for the DeepRL and Rule-based agents. Nevertheless, in each case, the reward function follows the following general structure:

$$r(s, a) = r_t(s, a) + r_c(s, a),$$

where $r_t(s, a)$ is the task depended reward and $r_c(s, a)$ is the penalty for violating the constraints. We describe each specific form of r_t in Section 3 and Section 4

3 AirHockey via Deep RL

We employ a DeepRL approach to train agents for the *defend* and *counter-attack* tasks. Both agents were trained employing the Soft Actor-Critic [SAC, Haarnoja et al., 2018] algorithm in combination

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

with ATACOM Liu et al. [2022]. This agent controls desired joint accelerations, which are then converted via ATACOM to joint position and velocities. In this setting, we opted for the control of acceleration as it was difficult to train an agent that directly controls the joint positions while respecting the constraints, especially when it was transferred to the evaluation environment.

In order to comply with the constraints, the single integration mode of ATACOM proved insufficient as it consistently violated joint velocity limits. Despite encountering challenges in implementing the double integration setting of ATACOM, we achieved success in ensuring that the agent adhered to the specified constraints. This was achieved by constraining joint accelerations within the range of -1 to 1.

3.1 Reward Structure

Defend and Counter-attack We train two agents for the defend and counter-attack tasks. In these tasks the episode starts with the puck moving towards our goal. In the both cases, the agent receives a high penalty (-100) when the puck breaches our goal. When the agent successfully interacts with the puck, a reward (+10) is granted, coupled with reward proportional to the norm of the puck's velocity, with a negative sign while defending and positive sign while counter-attacking. This incentivizes the agent to interact with the puck and lower its velocity while also protecting the goal in the defend task, while pushing back the puck in the counter-attack. Moreover, in the counter-attack, the agent can also score in the opponent's goal, for which he receiver an additional large bonus of 1000.

Return Home We define an additional task to train an agent to return to the initial position of the robot. At each step of the training process, the agent incurs a penalty based on the distance to the home configuration, $r = -\|\mathbf{q} - \mathbf{q}_{home}\|$.

3.2 Training via Curriculum Learning

To train each agent for the aforementioned task, we applied curriculum Learning [CL, Narvekar et al., 2020]. Indeed, we devise manual curriculums across three levels of complexity—easy, medium, and hard tasks. These levels differ in the initial configuration of the puck, considering its position and velocity. In the easy task, we constrained the velocity along the y-axis and kept the velocity in the x-axis low. Additionally, we positioned the puck at the end of the table, allowing the agent more reaction time. For the hard task, both x-axis and y-axis velocities were increased, and the puck's initial position was set in the middle of the table. The medium task served as an intermediary between the hard and easy tasks.

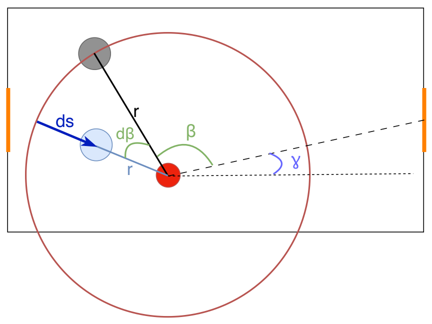
We trained the agents for a total of 8000 epochs, where each epoch consists in 30 episodes of experience. The first 3000 epochs the agent was trained with the easy defend tasks. After these initial epochs, we perform a *soft* switch to the medium task over the next 1000 epochs by increasing the proportion of the episodes drawn from the medium task. Between epoch 4000 to 6000, we exclusively train on the Medium task. From epoch 6000 to 7000 we progressively switch to the hard task, and then exclusively train on experience generated on the Hard task.

To train an agent to return to the home configuration from any arbitrary position, we curated a dataset of configurations obtained from various simulations of the other tasks. These configurations serve as initial states for each training episode. The episode terminates only after a certain number of steps in order to teach the agent to remain in the home configuration once it reaches it. For this particular task the state space includes only the joints positions and velocities.

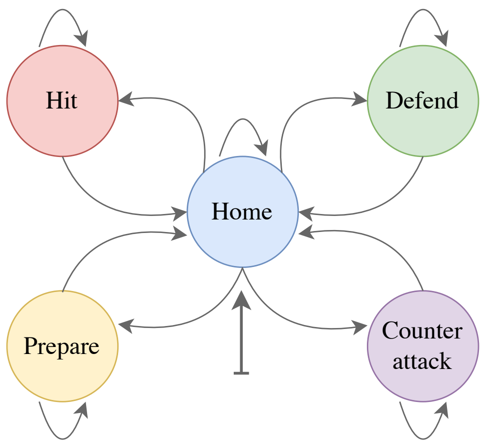
4 RL-Optimized Rule-based Controllers

For the *hit* and *prepare* tasks, we developed a Rule-Based Controllers to be optimized via Policy Gradient Parameter-based Exploration [PGPE, Sehnke et al., 2008] following the approach in Likmeta et al. [2020]. In this case, we directly control the position of the end-effector. We transformed the actions from the *task* space into the *joint* space, combining inverse kinematics and Anchored Quadratic Programming [AQP, Liu et al., 2021]. We divided the policy set into 2 main branches that resemble the challenge tasks: *hit* and *prepare*. Each policy is further divided into phases, common to both of them:

- *Adjustment*: move the end-effector on (i) the line linking the puck and the goal for the *hit*; (ii) the vertical line parallel to the short side, passing through the puck, for the *prepare*;



(a) Coordinates used to find the next desired position of the end-effector. the puck is the red circle, the grey circle is the mallet and the blue circle is the next desired position of the mallet.



(b) Finite state machine for controlling task changes

Figure 1: Components of the rule-based controllers: (a) definitions of $d\beta$ and ds, (b) representation of the Finite State Machine

- *Acceleration*: accelerate the end-effector and hit the puck. The *hit* policy hits the puck to score a goal, while the *prepare* one makes a more soft bump, in order to re-adjust the puck's position;
- *Final*: this phase only applies to the *hit* policy. Here, the agent keeps hitting the puck making it reach a desired acceleration, then slowly stops the end-effector following a curved trajectory.

In particular, the **hit policy** computes two quantities, $d\beta$ and ds in the following way:

$$d\beta = \begin{cases} (\theta_0 + \theta_1 \cdot t_{phase} \cdot dt) \cdot correction \\ \frac{correction}{2} \\ (\theta_0 + \theta_1 \cdot dt) \cdot correction \end{cases}, ds = \begin{cases} \theta_2 & \textit{adjustment} \text{ phase} \\ \frac{ds_{t-1} + \theta_3 \cdot t_{phase} \cdot dt}{radius + r_{mallet}} & \textit{acceleration} \text{ phase} \\ constant & \textit{slow-down} \text{ phase} \end{cases},$$

where $radius$ is the distance between the center of the puck and the end-effector, while $correction$ is always defined as:

$$correction = \begin{cases} 180 - \beta & y_{puck} \leq \frac{table\_width}{2} \\ \beta - 180 & y_{puck} > \frac{table\_width}{2} \end{cases}.$$

For what concern the **prepare policy**, the quantities $d\beta$ and ds are computed as:

$$d\beta = \begin{cases} \theta_0 \cdot t_{phase} + \theta_1 \cdot correction \\ correction \end{cases}, ds = \begin{cases} 5 \cdot 10^{-3} & \textit{adjustment} \text{ phase} \\ \theta_2 & \textit{acceleration} \text{ phase} \end{cases},$$

where $correction$ is defined as:

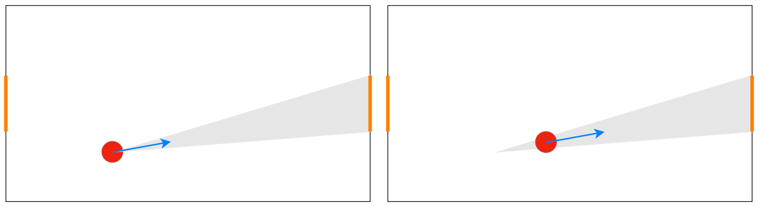
$$correction = \begin{cases} \beta - 90 & y_{puck} \leq 0 \\ 270 - \beta & y_{puck} > 0 \end{cases}.$$

All the quantities appearing in the computation of $d\beta$ and ds are explained graphically in Figure 1a.

The major issue we faced while developing this solution was the extreme sensitivity of the parameters w.r.t. the constraints as the return landscape for our rule-based policies was highly non-smooth.

In particular, for what concerns the *hit* task, the reward function is based on the construction of a triangle with the opponent's area and the puck as vertices (Figure 2). If the puck, after the hit, lies inside the triangle, a reward is assigned, proportional to the puck's velocity; if the puck is outside the triangle, a penalty is assigned. The more the puck is outside the polygon, the bigger the penalty.

$$reward\_before\_hit = \begin{cases} -\frac{||ee_{pos} - puck_{pos}||}{0.5 \cdot table\_diag} & \text{no hit} \\ A + B \cdot \frac{(1 - (2 \cdot \frac{\alpha}{\pi})^2) \cdot ||v_{ee}||}{max\_vel} & \text{hit the puck} \\ \frac{1}{1-\gamma} & \text{goal} \end{cases}, \tag{1}$$



(a) Triangle computation at step t. (b) Puck moving and reward assignment at step $t+1$.

Figure 2: Triangle construction for assigning reward after the mallet hit the puck.

Where A and B are constant values, respectively equal to 100 and 10, $table\_diag$ is the diagonal of the table, while $\alpha = \arctan 2(ee_{y\_vel,ee_{x\_vel}})$. Finally max_vel is a constant value equal to the maximum velocity observable in the environment.

If the agent hits the puck, it receives an instantaneous reward and, after that, the triangle approach is applied.

$$reward\_after\_hit = \begin{cases} B + ||v_{puck}|| & \text{puck inside the triangle} \\ -diff\_angle & \text{puck outside the triangle} \end{cases}, \tag{2}$$

where B is a constant equal to 10 and

$$diff\_angle = \arctan 2(v_{y_{puck}}, v_{x_{puck}}) - angle\_border, \tag{3}$$

where $angle\_border$ is the angle between the puck velocity vector and the closest triangle border.

5 Hierarchical agent

Finally, we combined the described solutions into a single agent which operates at a higher level, dynamically selecting the most appropriate task during the game. To select the task, this agent relied on two components: the *Switcher* and the *Finite State Machine*. The *switcher* is the component developed to select a new task when the current one is completed. It consists in a simple rule-based controller which decides the task to employ via geometric considerations. After its last action, each task sends a signal to the high-level agent, notifying its completion. The switcher will select another task, which can potentially be also the same as before. The *Finite State Machine* (FSM) is added as a second layer of safety to mitigate possible constraints violations while switching tasks. It forces the agent, at the end of each task, to go back to the Home and start selecting a new task from there. The structure of the FSM can be observed in Figure 1b. The agent always starts a match in the *Home* state, it is possible to remain in the same state but not switching from a task to another without passing from the *Home* state.

6 Conclusion

In this paper, we describe the approach developed by the RL3 Polimi team for the control of the AirHockey agent during the Robot Air Hockey Challenge 2023. We propose a two-level approach, where low-level policies are trained to perform base tasks in the game like hitting, defending and counter-attacking, while a high-level rule-based controller is employed to decide which low-level policy to employ in each step. The low-level policies were trained via DeepRL or Rule-based RL optimization according to the task. The final proposed agent achieved a 50% win-rate in the final phase of the competition, mostly respecting the robot-constraints even under noisy observations and displaying a satisfactory game-playing ability.

References

Gregory F Welch. Kalman filter. *Computer Vision: A Reference Guide*, pages 1–3, 2020.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018. URL `http://arxiv.org/abs/1801.01290`.

Puze Liu, Davide Tateo, Haitham Bou Ammar, and Jan Peters. Robot reinforcement learning on the constraint manifold. In Aleksandra Faust, David Hsu, and Gerhard Neumann, editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 1357–1366. PMLR, 2022.

Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *CoRR*, abs/2003.04960, 2020. URL `https://arxiv.org/abs/2003.04960`.

Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber. Policy gradients with parameter-based exploration for control. In *Artificial Neural Networks-ICANN 2008: 18th International Conference, Prague, Czech Republic, September 3-6, 2008, Proceedings, Part I 18*, pages 387–396. Springer, 2008.

Amarildo Likmeta, Alberto Maria Metelli, Andrea Tirinzoni, Riccardo Giol, Marcello Restelli, and Danilo Romano. Combining reinforcement learning with rule-based controllers for transparent and general decision-making in autonomous driving. *Robotics and Autonomous Systems*, 131:103568, 2020. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2020.103568. URL `https://www.sciencedirect.com/science/article/pii/S0921889020304085`.

Puze Liu, Davide Tateo, Haitham Bou-Ammar, and Jan Peters. Efficient and reactive planning for high speed robot air hockey. 2021.