Learning to Play Air Hockey with Model-Based Deep Reinforcement Learning

Andrej Orsula
Space Robotics Research Group (SpaceR)
Interdisciplinary Centre for Security, Reliability and Trust (SnT)
University of Luxembourg
`andrej.orsula@uni.lu`

Abstract

In the context of addressing the Robot Air Hockey Challenge 2023, we investigate the application of model-based deep reinforcement learning to acquire a policy capable of autonomously playing air hockey. Our agents learn solely from sparse rewards while incorporating self-play to iteratively refine their behaviour over time. The robotic manipulator is interfaced using continuous high-level actions for position-based control in the Cartesian plane while having partial observability for the state of the environment with stochastic transitions. We employ DreamerV3 as the underlying model-based reinforcement learning algorithm to demonstrate its versatility for solving complex robot manipulation tasks. The source code and pre-trained models are available at https://github.com/AndrejOrsula/drl_air_hockey.

1 Introduction

The Robot Air Hockey Challenge 2023 [1] offers an engaging opportunity for robot learning researchers and practitioners to develop and evaluate their solutions for a realistic robot manipulation task while being able to compare their outcomes with other teams in a competitive setting. The challenge is organized by the Intelligent Autonomous Systems Group of the Computer Science Department at the Technische Universitaet Darmstadt (TU Darmstadt) and consists primarily of a virtual phase followed by real-world experiments among the finalists. The organizers provide the simulation environment for the virtual phase of the challenge together with the accompanying evaluation framework and a baseline agent that can be used by the participants as a starting point for their own solutions. Furthermore, the organizers facilitate and conduct real-world experiments on their physical analogue of the air hockey table with two robotic manipulators. For each participating team, the challenge involves developing a controller capable of autonomously playing air hockey against controllers of other teams while employing learning-based algorithms for at least one component of their agent.

There are several possibilities for combining various learning-based algorithms applied to distinct components of a performant controller. However, our initial interest revolved around techniques that can be used to directly optimize a single policy capable of playing the full game of air hockey without relying on hand-crafted components or heuristics within the controller itself. From the popular approaches, the paradigms of imitation learning and reinforcement learning were initially considered. Despite the potential suitability of imitation learning, e.g. behaviour cloning in its simplest form, the need to collect expert demonstrations would demand additional engineering effort and could introduce undesired bias into the learned policy. Therefore, we decided to focus on reinforcement learning as it was deemed more suitable for the task of playing air hockey, which can be characterized as a two-player zero-sum game with simultaneously executed actions. This nature of the game further motivates the use of self-play that reinforcement learning agents can exploit to discover potential weaknesses in their own behaviour and develop better strategies to overcome them.

We believe that both model-free and model-based reinforcement learning approaches are applicable to the task of playing air hockey. In this work, we purposefully delimit our investigation to the model-based approach exemplified by DreamerV3 [2] due to recent advancements and its demonstrated capacity for solving various tasks in diverse domains. Although not directly analyzed in this work, model-based reinforcement learning might provide additional benefits for agents in competitive settings, such as its perceived sample efficiency being beneficial for enabling continual learning where agents could adapt their models over time to the changing behaviour of the current opponent.

2 Challenge Setup

The tournament phase of the challenge utilizes an air hockey table equipped with two *KUKA iiwa14* 7-DoF robotic manipulators, each controlled by one of the participating teams. This setup emulates the challenging nature of real air hockey with its fast-paced dynamics for robot-enabled gameplay. Each robot is equipped with a mallet attached to its end-effector that enables effective interaction with the puck. The simulation environment provided by the organizers mirrors the real-world setup inside *MuJoCo* [3] robotics simulator to provide a virtual testbed for development and evaluation. The virtual setup used throughout the challenge is shown in Figure. 1.

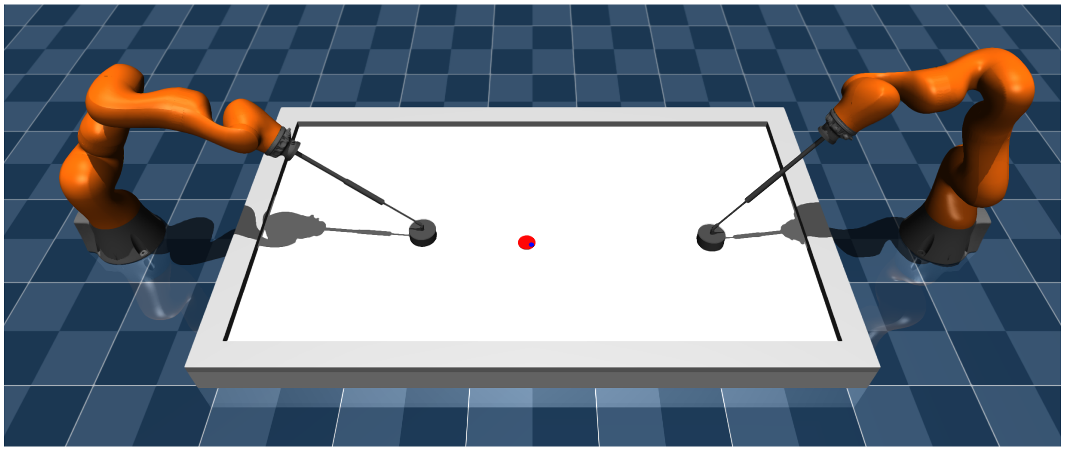

Figure 1: The simulation environment used in the Robot Air Hockey Challenge 2023 featuring a table, puck, and two robotic manipulators controlled by participating teams.

Both robots execute interpolated joint trajectories based on commands supplied by agents of the participating teams. These agents are required to provide joint-level commands at 50 Hz, imposing a computational time constraint of 20 ms per control cycle. At each step, the environment provides agents with the current state of the puck, the state of their joints, and the position of the opponent's mallet. The primary aim of agents is to score goals while preventing their opponent from doing so within the 15-minute duration of every match, i.e. 45000 simulation step. Agents can also receive a fault every time the puck remains on their side of the table for more than 15 s, with every three faults resulting in a subtracted point. Lastly, in cases where the puck gets stuck in the middle of the table while neither of the robots can reach it, it is randomly reset to one side.

In order to encourage the safety of sim-to-real deployment, the organizers impose limits on the behaviour of agents during the evaluation. These limits encompass the maximum joint positions and velocities, the Cartesian pose of the end-effector, and the aforementioned computational constraint. If agents exceed any of these limits, they accumulate penalty points and risk disqualification. Additionally, environmental stochasticity is introduced during the evaluation to examine the robustness of agents. Such modifications are undisclosed to participants but include environment disturbances, observation noise, and simulated loss of puck tracking.

3 Approach

Our approach revolves around leveraging DreamerV3 for training reinforcement learning agents to play the full game of robot air hockey. We focus on guiding agents towards optimal behaviour purely through sparse rewards corresponding to relevant score-affecting events that are not biased by a potentially over-engineered reward shaping. To acquire policies capable of competing against various approaches, we employ a form of fictitious self-play to support agents in discovering more robust strategies by exploiting and patching any weaknesses in their prior behaviour.

The observability of the environment is facilitated for agents at each discrete time step through a set of low-dimensional environment states. Observation stacking is employed within such representation to provide additional temporal information about environment dynamics. Based on these observations, our agents interface with the robotic manipulators via continuous high-level actions for position-based control in the Cartesian plane. Inverse Jacobian method is then used to map these high-level actions into lower-level joint space commands that can be used to actuate the arm.

In the scope of this challenge, we train three distinct agents that optimize their behaviour towards different strategies. All of these agents use the same algorithm, network architecture, observations and action spaces but different reward functions that reinforce their behaviour towards a specific strategy. We use the concept of multiple strategies during the self-play training process and within the multi-strategy ensemble described at the end of this section. These strategies are referred to as *balanced*, *aggressive* and *defensive*.

3.1 Algorithm

At its core, DreamerV3 [2] employs three components, i.e. world model, actor and critic, that are optimized concurrently from replayed experience collected via interaction with the environment. The world model, implemented as a Recurrent State-Space Model, learns a compact latent representation of the observations using an autoencoder. Throughout the training, this model is used to predict abstract future sequences alongside rewards associated with potential actions. The actor and critic leverage the generative capabilities of the world model to optimize their networks purely from imagined sequences, while the actor maximizes the expected return, and the critic predicts the return given the current actor's behaviour.

We believe that this process results in sample-efficient training that would enable agents learning to compete in complex games to experience various environment transitions at a low computational cost, with the premise of acquiring a robust policy that generalizes well to novel strategies of unseen opponents. However, the soundness of this hypothesis and the full potential of this approach within the competitive setting of zero-sum games, such as robot air hockey, remains to be investigated.

3.2 Observation Space

The observation space of our agents comprises a set of low-dimensional environment states that the organizers made accessible for the participants. To capture information about the state of the controlled robot, its current joint positions and the planar position of the mallet attached as its end effector are included. Not only do these observations make an agent aware of its current state and manipulability, but it is also essential for the world model to learn the consequences of the agent's actions on the state of the controlled robot. The state of the opponent robot is available in the form of the planar position of their mallet, which allows agents to react to the opponent's actions even before they strike the puck. Similarly, the puck is observable through its planar position and orientation, providing the most crucial piece of information for agents to act upon and the world model to learn how to predict. Lastly, the duration of the puck spent on either side of the table is also encoded in the observation space to provide agents with time awareness with the aim of avoiding faults.

All observations are normalized to the range $[-1, 1]$. The joint positions are normalized based on their limits, while the planar positions of the mallets and the puck are normalized based on the dimensions of the table. We encode the orientation of the puck as sine and cosine of the angle to preserve its continuity, while the duration of the puck spent on either side of the table is normalized based on the time limit until a fault would be accumulated with the sign corresponding to the side of the table.

Although the aforementioned states might provide a sufficient amount of information for agents to act upon, they lack temporal information about the environment dynamics required to fulfil the Markov assumption. Derivative states, such as the linear and angular velocity of the puck, are directly available to the participants and could also be estimated manually from two consecutive observations, however, their normalization would either require artificial limits or a more complex normalization scheme. Moreover, the stochasticity introduced during the evaluation in the form of simulated loss of puck tracking would make the derivative states unreliable. Therefore, we employ observation stacking [4] as a simple method of providing additional temporal information about environment dynamics. In this way, the last n observations of the selected states are stacked together and used as a single observation. At the beginning of each episode, the first observation is duplicated to fill the stack. Furthermore, as the position of the puck is considered especially important for agents while suffering from the loss of tracking, the observation stacking is performed asymmetrically with the aim of keeping an even longer history about this particular state. The final observation is then formed through the concatenation of all its components listed in Table 1, resulting in a total dimensionality of 40.

Table 1: The observation space of our agents, indicating the dimension and stack history of each state.

State	Dimension	Stack (n)
Participant — Joint positions	7 (DoF)	1
Participant — Mallet position	2 (x, y)	2
Opponent — Mallet position	2 (x, y)	2
Puck position	2 (x, y)	10
Puck velocity	2 (x, y)	2
Fault timer	1 (t)	1

3.3 Action Space

Playing air hockey incorporates continuous control over a planar surface. Therefore, we employ continuous high-level actions for position-based control within the Cartesian plane. At each discrete time step, agents are required to provide the absolute target position of their mallet as a 2D vector, with its coordinates normalized to the range $[-1, 1]$ within the intersection of the table and the reachable workspace of the robot.

The high-level actions are then mapped into lower-level joint space commands using the Inverse Jacobian method. Initially, the relative displacement of the mallet $(\Delta x, \Delta y, \Delta z)$ is determined from its current position to the target. The position along the Z-axis is always constrained based on the relative height of the table. After computing the Jacobian matrix J in the current joint configuration and obtaining its pseudo-inverse J^+, the target joint displacement can be calculated as $(\Delta\theta_1, \Delta\theta_2, ..., \Delta\theta_n) = J^+ \cdot (\Delta x, \Delta y, \Delta z)$. To emphasize the displacement along the Z-axis and maintain consistent contact of the mallet with the table, a weighted product with weights $(0.25, 0.25, 0.5)$ is applied. We ensure that the motion is dynamically feasible by clipping the joint displacements based on the joint velocity limits. Finally, the resulting joint positions and velocities are derived and used as low-level joint space commands that are linearly interpolated over the duration of the control cycle.

3.4 Reward Function

To guide our agents towards optimal behaviour for playing air hockey, we adopt a sparse reward function designed to provide clear signals for specific events without introducing potential biases from intricate reward shaping. In the case of air hockey, the primary events coincide with terminal states that occur either when a goal is scored or a fault is accumulated by one of the agents. Within the duration of a match, episode termination can also occur when the puck gets stuck in the middle of the table, however, we do not consider this terminal state as part of the reward. In general, our agents receive a positive reward for scoring the goal, while a negative reward is attributed to receiving a goal or causing a fault. Although a positive reward could also be attributed when the opponent receives a fault, we do not consider this event as it is dependent on the opponent's behaviour and could lead to ambiguous credit assignment.

We experiment with three different configurations of the reward function that reinforce agents towards different strategies, i.e. *balanced*, *aggressive* and *defensive*. The reward function is summarized in Table 2. The *balanced* strategy is used as the default configuration and is designed to encourage agents to play a *balanced* game, i.e. to score goals while defending their own side of the table. Empirically, it was found that weighting the reward for scoring a goal equally with the reward for receiving a goal would result in risky behaviour where agents would keep their mallet close towards the centre of the table while eagerly trying to intercept the puck. This would often result in diminishing returns at the cost of receiving a goal, especially against novel opponents. Therefore, the *balanced* strategy is incentivized to be slightly more defensive. On the other hand, the *aggressive* strategy weights both rewards equally, resulting in more risky behaviour. Lastly, the *defensive* strategy is designed to solely defend the goal against the opponent without any incentive to score goals. At the same time, all of these strategies receive negative rewards for causing a fault, which is designed to discourage agents from accumulating them.

Table 2: The components of the reward function used to reinforce agents towards different strategies.

Strategy	Score a goal	Receive a goal	Cause a fault
Balanced (default)	$+\frac{2}{3}$	-1	$-\frac{1}{3}$
Aggressive	$+1$	-1	$-\frac{1}{3}$
Defensive	0	-1	$-\frac{1}{3}$

3.5 Self-Play

Although a baseline agent is provided by the organizers, its policy covers only a limited subset of potential strategies in competitive robot air hockey. Training solely against this baseline agent would result in a policy effective only against this specific opponent, as demonstrated in our experimental section. To overcome this limitation, we employ a form of fictitious self-play [5] to iteratively discover more robust strategies during training. In this way, our agents are trained against a pool of opponent agents with frozen weights. At the beginning of each episode, a new opponent is uniformly sampled from the pool and used until the episode terminates. The pool is gradually expanded with a new model every 1000 episodes while being limited to a total of 25 opponents, with a uniformly random opponent being replaced once the pool is full.

We also incorporate the strategies described in the previous section into self-play using a two-step procedure. First, three agents following the three strategies are trained using self-play, with the opponent pool initialized using the baseline agent. Once these agents begin to exhibit stable behaviour, their training is terminated while keeping a history of checkpoints for their models. Subsequently, a new agent is trained from scratch following the *balanced* strategy. This time, the opponent pool is pre-filled with not only the baseline agent but also eight checkpoints for each strategy from the previously trained agents (total of 25 agents). In this way, the new agent is immediately exposed to a diverse pool of advanced opponents that have expertise in different aspects of the game. The *aggressive* opponents immediately incentivize the new agent to defend against their risky behaviour, while the *defensive* opponents provide challenging agents to score goals against. At the same time, the *balanced* opponents provide a tradeoff between the two extremes and further force the new agent to explore robust strategies.

Moreover, our custom version of the environment stochasticity is introduced during the training while incorporating environment disturbances, observation and action noise, and simulated loss of puck tracking. However, only the trained agent is affected by the added noise and simulated loss of puck tracking, while the opponents remain unaffected to further increase the difficulty of the training.

3.6 Multi-Strategy Ensemble

Recognizing the distinct strengths and weaknesses exhibited by agents trained with different strategies, we decided to combine them into a single ensemble for the final evaluation of the challenge. This ensemble dynamically selects one of three agents, i.e. *balanced*, *aggressive* and *defensive*, based on the current score of the match that is estimated by the ensemble through the observations available to the participants, i.e. the position of the puck and the duration of the puck spent on either side of the table. By default, the *balanced* agent is used for the majority of the game. If the opponent is in the lead, the ensemble switches to the *aggressive* agent to increase the likelihood of scoring a goal through a more risky playstyle. Conversely, if the opponent is trailing by a significant margin, the *defensive* agent is activated to provide a safer alternative in preventing the opponent from scoring a goal.

4 Experimental Results

Throughout this section, we present the preliminary results of our experimental evaluation of the proposed approach focused on selected aspects of the challenge and employed algorithm.

4.1 Training and Hyperparameters

Most default hyperparameters of DreamerV3 provided by the authors were used due to their demonstrated effectiveness. We primarily adjust the size of the model to match the computational requirements of the challenge while following the suggested S model size from the original paper [2], resulting in a total of $9.7M$ learnable parameters spread across the world model, actor and critic networks. With this model, the mean inference time of our agents was tested to be 5.2 ms on the evaluation server across four matches against the baseline agent. We set the imagination horizon to 50 $(1$ s) in order to provide agents with a sufficient amount of time in which they can learn the direct consequences of their actions. Furthermore, we adjusted the training process by updating the replay capacity to 10^7, the batch size to 32, and the training ratio to 512.

Our agents are trained on a single workstation with *AMD Ryzen 9 7950X* CPU and *NVIDIA GeForce RTX 4090* GPU for a total of 100 M simulation steps. To accelerate the training and provide diverse experience for agents, we employ ten parallel environment workers that collect transitions from the environment and store them in a shared replay buffer. Figure. 2 contains the learning curve of the episodic reward achieved by an agent following the *balanced* strategy throughout its training. It can be seen that the performance of the agent stagnates for most of its training, which can be largely attributed to the self-play mechanic, however, we note that the quality of the behaviour continues to improve. When tested against the baseline agent across 20 matches (5 hours of gameplay), our agent wins with an average score of $6.1 : 0.2$.

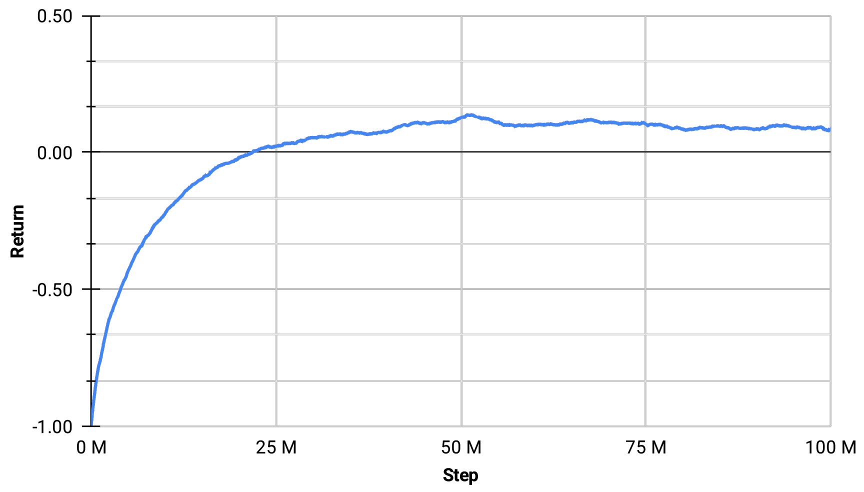

Figure 2: The learning curve of an agent following the *balanced* strategy.

4.2 Effect of Self-Play

We also analyze the impact of self-play on the robustness of our agents. In addition to the agent from Figure. 2, we train another agent using the same configuration but without self-play. This agent is trained only against the baseline agent for the entire duration of its training. When tested against the baseline agent across 20 matches, the agent without self-play wins with a much higher average score of $14.5 : 0.1$, which is indicative from its learning curve shown in Figure. 3. However, when matched against the agent trained with self-play, it loses with an average score of $0.7 : 14.3$. This indicates that the agent trained without self-play is overfitted to the baseline agent and is unable to generalize to novel opponents, which disproves our initial hypothesis that learning from the imagined sequences of the world model would enable agents to generalize to novel strategies of unseen opponents.

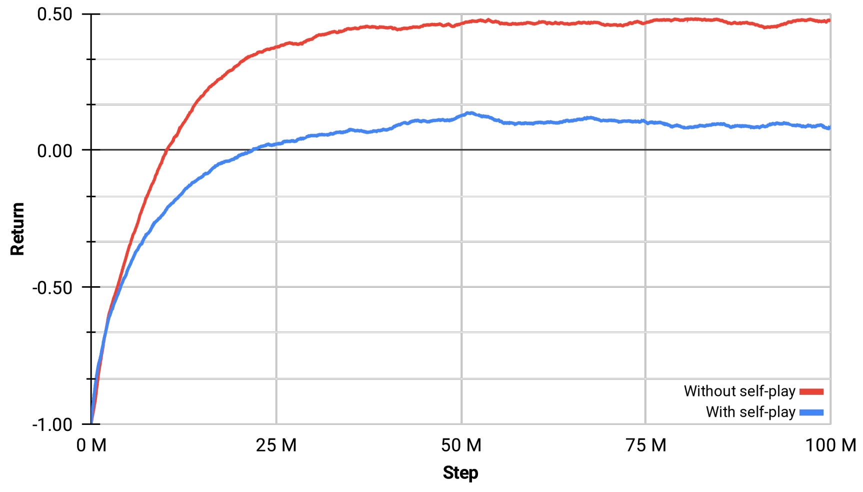

Figure 3: Learning curves of agents trained with and without self-play.

4.3 Impact of Imagination Horizon

The critic network enables DreamerV3 to consider rewards even beyond the imagination horizon. However, the direct impact of this hyperparameter on the performance of the agent has not been explored to a great extent. Therefore, we investigate its impact on the performance of our agents. Figure. 4 contains the learning curves of agents following the *balanced* strategy trained with different imagination horizon lengths. It can be seen that agents trained with longer imagination horizon lengths exhibit a more stable learning curve while also achieving a higher episodic reward. However, increasing the imagination horizon length also increases the memory requirements of the agent during the training, where the length of 50 fully saturates the memory of the utilized GPU. When evaluated across 20 matches, agent using imagination length of 50 wins both against agents using lengths of 25 and 10 with an average score of $3.9 : 2.0$ and $3.6 : 1.8$, respectively. Similarly, the agent using imagination length of 25 marginally wins against the agent using length of 10 with a average score of $4.3 : 3.7$.

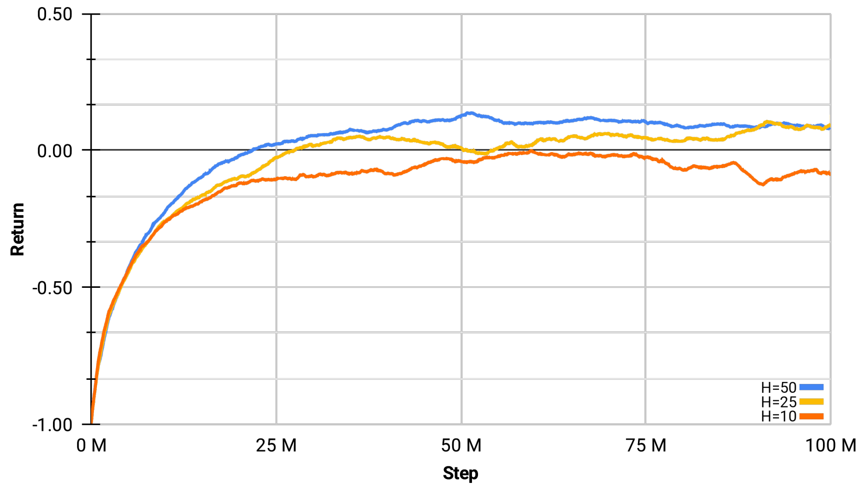

Figure 4: Learning curves of agents trained with different imagination horizon lengths (H).

5 Discussion and Conclusion

This work presented our approach employed in the Robot Air Hockey Challenge 2023. We demonstrated that model-based reinforcement learning exemplified by DreamerV3 can be applied to acquire a policy capable of playing air hockey. Our agents learn directly from sparse rewards while incorporating self-play to iteratively refine their behaviour over time. We demonstrated that agents are prone to overfitting to the baseline agent when trained solely against it, which indicates that self-play is essential for agents to generalize to novel opponents. Furthermore, we showed that the imagination horizon length has a direct impact on the performance of agents, with longer lengths resulting in a more stable learning curve and higher episodic reward.

The use of high-level actions for position-based control in the Cartesian plane simplifies the interface with the robotic manipulator while providing a sufficient amount of control for playing air hockey. However, directly using the policy to drive the motion of the robot without additional heuristics results in a perceivably chaotic behaviour that is not suitable for real-world operation. Before attempting to deploy the policy on the real robot, safety concerns need to be addressed. This could be achieved by incorporating additional objectives or constraints into the policy, smoothing the motion of the robot or changing the interface to lower-level actions, e.g. by using velocity-based control.

Based on the preliminary results presented in this work, we believe that model-based reinforcement learning is a promising approach for solving complex robot manipulation tasks such as playing air hockey. The versatility of DreamerV3 for solving various tasks without the need for extensive hyperparameter tuning makes it a suitable candidate for applications in diverse domains. However, further evaluation is required to better understand its benefits within the context of competitive zero-sum games, especially with respect to the use of self-play and its eventual incorporation into the inner process of learning from imagination. Furthermore, we have not thoroughly investigated the impact of observation stacking on the performance of our agents when combined with the actor that already benefits from the Markovian representation learned by the world model.

References

[1] P. Liu *et al.*, “Robot air hockey challenge 2023,” https://air-hockey-challenge.robot-learning.net.

[2] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap, “Mastering diverse domains through world models,” 2023.

[3] E. Todorov, T. Erez, and Y. Tassa, “Mujoco: A physics engine for model-based control,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 5026–5033.

[4] V. Mnih *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[5] J. Heinrich, M. Lanctot, and D. Silver, “Fictitious self-play in extensive-form games,” in *International Conference on Machine Learning*, 2015.