Learning the Low-level Policy for Robot Air Hockey

Lernen der Low-level Policy beim Roboter-Air-Hockey
Bachelor thesis by Jonas Günster
Date of submission: 13th April 2022

1. Review: Prof. Ph.D. Jan Peters
2. Review: Dr. Davide Tateo
3. Review: M.Sc. Puze Liu
Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Erklärung zur Abschlussarbeit gemäß §22 Abs. 7 und §23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Jonas Günster, die vorliegende Bachelorarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß §23 Abs. 7 APB überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 13. April 2022

Jonas Günster

Abstract

The current approaches for solving robotics with reinforcement learning often lack safety guarantees, which are required for real-world usage. A solution to this is safe exploration algorithms that respect a set of constraints during the whole training process. In this work, we are going to evaluate the performance of a specific safe exploration algorithm introduced by Liu et al. [18]. To do that we are going to extend the experiments of the original paper to more general tasks in the air hockey domain. We chose this air hockey domain because it is a high speed, dynamic environment, in which it is important to fulfil constraints. Furthermore, air hockey has a small state space which results in fast convergence of RL algorithms.

We will evaluate the performance of a simple planar robot and an IIWA industrial robot. Both of these agents respect the constraints at all times when trained with the safe exploration algorithm. Additionally, their performance with and without the safe exploration algorithm is comparable. Thus we conclude that the safe exploration algorithm can fulfil constraints while not negatively impacting performance.

Furthermore, we analyse the learned behaviours of the planar robot to show that we designed sufficiently informative reward functions. Moreover, we show that it is possible to combine the tasks into a general air hockey player, that can play against itself.

Zusammenfassung

Den derzeitigen Ansätzen zur Lösung von Robotikproblemen mit Hilfe von Reinforcement Learning fehlt es oft an Sicherheitsgarantien, die für den Einsatz in der realen Welt erforderlich sind. Eine Lösung für dieses Problem sind sichere Explorations Algorithmen, die während des gesamten Trainingsprozesses eine Reihe von Beschränkungen einhalten. In dieser Arbeit werden wir die Leistung eines speziellen sicheren Explorationsalgorithmus bewerten der von Liu et al. [18] vorgestellt wurde. Zu diesem Zweck werden wir die Experimente der Originalarbeit auf allgemeinere Aufgaben in der Air-Hockey-Domäne erweitern. Wir haben diese Airhockey Domäne gewählt, weil es sich um eine dynamische Hochgeschwindigkeitsumgebung handelt, in der es wichtig ist, Beschränkungen zu erfüllen. Außerdem hat Airhockey einen kleinen Zustandsraum, was zu einer schnellen Konvergenz der RL-Algorithmen führt.

Wir werden die Leistung eines einfachen planaren Roboters und eines IIWA Industrieroboters bewerten. Beide Agenten hielten sich beim Training mit dem sicheren Explorationsalgorithmus jederzeit an die Beschränkungen. Außerdem ist ihre Leistung mit und ohne den sicheren Explorationsalgorithmus tionsalgorithmus vergleichbar. Daraus schließen wir, dass der sichere Explorationsalgorithmus die Bedingungen erfüllen kann, ohne sich negativ auf die Leistung auszuwirken.

Außerdem analysieren wir das gelernte Verhalten des planaren Roboters, um zu zeigen, dass wir ausreichend informative Belohnungsfunktionen entwickelt haben. Zudem zeigen wir, dass es möglich ist die Aufgaben zu einem allgemeinen Air-Hockey-Spieler zu kombinieren, der gegen sich selbst spielen kann.

Contents

List of Tables

List of Figures

1 Introduction

1.1 Motivation

Safety is a property that is highly valued by humans. Hence it is in almost every aspect of our lives. From transportation to housing or food, everything has strict safety regulations. For example, a car has to meet a certain standard to be deemed roadworthy and buildings have to be designed with fire hazards and emergency exits in mind. Perishable foods have to be handled in a certain way to be deemed safe for consumption. All these are things that increase safety in our society. With that in mind, the only way robots can see widespread adaptation in day-to-day life is with strict safety regulations.

1.1.1 Robot Safety

Robots are currently deployed in the production of goods, as well as in the construction, healthcare and service industries. In recent years robots have shown an increase in performance, which makes it viable for them to directly assist users in their work. When robots and users work hand in hand, safety violations can cause severe consequences for people and equipment [20]. Thus there have to be strict constraints on the movement of the robot, which cannot be violated at any time.

Traditionally these standards are enforced via a trajectory planner that, given two configurations, can produce a trajectory between them that satisfies the constraints [11]. The problem is that the intended behaviour has to be encoded into the trajectory planning algorithm, which is challenging in complex tasks.

To achieve good performance in these complex tasks deep reinforcement learning (RL) has become a prominent tool. It outperforms traditional methods in simulated continuous control tasks frequently [16, 8]. Unfortunately transferring this performance to the real

world remains challenging. Indeed maximizing the cumulative reward does not prevent occasional terrible performance which means there are no guarantees in RL. With this lack of guarantees, deploying an RL agent in a real-world environment is not feasible.

Liu et al. [18] address this problem by proposing a safe exploration algorithm called Robot Reinforcement Learning on the Constraint Manifold (ATACOM). Given an arbitrary number of constraints, the algorithm will ensure that no violations occur during the whole learning process. These constraints impose restrictions on the behaviour of the robot. They can for example limit the area a robot can move in and therefore create safety guarantees, which make the transition to real-world environments possible. Liu et al. further show that it is possible to learn an air hockey hit agent with multiple constraints. The agent can hit a puck in such a way that it will score a goal, while simultaneously not violating any constraints. Additionally, they were able to train a defending agent, which can stop an incoming puck. A limitation is that both these environments are completely static in their initialisation, giving the puck the same initial position and velocity every time. This is of course not representative of a real air hockey game.

1.1.2 Why Air Hockey

Air hockey is a game in which the objective is to push a puck into the opponent's goal. Whoever scored the most goals wins the game. We use a mallet to interact with the puck, which is a circular striker that propels the puck when a collision occurs. It is not allowed to lift the mallet from the table and place it over the puck to control it.

Because the puck only moves on the surface of the table, air hockey can be modelled as a 2D constrained environment. Therefore the state space will have a moderate size, resulting in a reasonable training time for RL algorithms. Additionally, many constraints have to be respected in order to prevent damage to the environment. For example, the mallet should not move beyond the bounds of the table, as it would destroy the sides of the air hockey table.

Solving air hockey with traditional methods is challenging because of fast puck movements and high modelling uncertainty. This uncertainty has two causes: firstly, air flows through small holes on the surface of the table to reduce the friction between the puck and the table. This creates an uneven airflow distribution, resulting in high uncertainty and fast movements. Secondly, the collision behaviour between the cylindrical puck and mallet or the table's borders produces highly variable trajectories, as it is sensitive to small differences in the system state [17].

In conclusion, the air hockey model is hard to solve and complex. The many constraints make it a challenging benchmark for the safe exploration algorithm. Due to the limited state space, we can solve the problem using RL techniques in a reasonable amount of time. This is perfect for evaluating the performance of ATACOM.

1.2 Goal of the Thesis

The goal of this thesis is to show that ATACOM can solve more challenging tasks than proposed by the original paper. We will stick with the air hockey domain for these harder tasks because of the reasons stated in section 1.1.2.

The first objective is to increase the difficulty of the hit and defend task of the paper. This is done by randomizing the initial position and velocity, such that it's akin to a real opponent. That makes the task much harder because the optimal behaviour is now based on the puck initialisation and differs for every episode. In order to achieve maximal performance, we hand tune the reward functions and hyperparameters. We show that it is possible to achieve good performance in these more general tasks while not violating constraints.

The second objective is to introduce new tasks that challenge ATACOM. To increase the realism, these new tasks together with the established tasks should be combinable into a general air hockey agent, which could play against a human opponent. In order to motivate the chosen tasks, we first introduce some air hockey strategies:

The goal of an air hockey player is of course to score in the opponent's goal while not conceding the own goal. To maximize the chance of scoring, it's desired to stop the puck in your half of the table and then get a precise shot. However, it is not always possible to stop the puck. If it is deemed too risky to stop the puck the goal is to repel the puck back to the opponent with such momentum that the opponent is also not able to take control of the puck. Thus the opponent cannot maximize his chances of scoring a goal and also has to repel. In short, it's desired to take control of the puck and deny control of the puck to the opponent.

Given this strategy, one of the new tasks is *repel*. The challenge for ATACOM is fulfiling constraints in high-velocity scenarios, where precision is needed. The other task is *prepare*, which improves the position of the puck that is already under control. This task is fundamental because the defence task can leave the puck in an undesirable location, for example, close to the bounds of the table. The *prepare* task then moves the puck closer to the centre, from where the hit task can perform better. The challenges are operating

precisely while being very close to violating constraints. We show that with hand-tuned reward functions and hyperparameters, ATACOM delivers good performance for the new tasks. Therefore it can solve a broad spectrum of tasks, which require different behaviours.

Finally, we prove that it is possible to build a system that exploits this learning framework in order to play air hockey. As aforementioned, this system combines the tasks in order to become a general air hockey agent.

2 Foundations

2.1 Reinforcement Learning

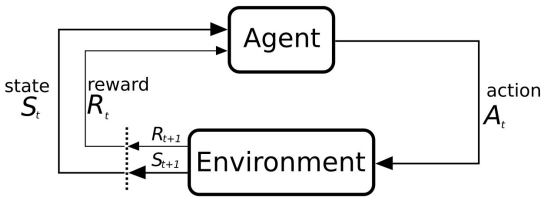


Figure 2.1: Diagram of the agent environment interaction. The agent interacts with the enviroment based on the state s with an action a

Reinforcement learning (RL) is an area of machine learning, where an agent learns to interact with an environment by exploring it. Instead of the labelled data utilized in supervised learning, reinforcement learning collects data via an agent. As shown in Figure 2.1 the agent observes that the environment is currently in state s_t. Based on this observation he chooses an action a_t. After taking the action in the state s_t, the agent reach a new state s_{t+1}. Additionally, there is a reward function, which provides a local measure of the agent's performance. The resulting reward can be seen as a rating for the action a_t given the state s_t.

The goal of reinforcement learning is to maximize the cumulative reward: In other words, reinforcement learning is the search for the optimal agent. This agent will choose actions in every state which result in the highest cumulative reward [14].

Before we can explore algorithms that search the best agent, we first have to define a formal model of the environment.

2.1.1 Markov Decision Process

A Markov Decision Process (MDP) is the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is the state space and $\mathcal{A}$ is the action space. The enviorment is modeled with a transitional probability distribution denoted by $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$. The reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ defines the reward obtained during the last transition. $\gamma \in [0, 1]$ is the discount factor, which weights each reward in a trajectory with a notion of time. A lower discount factor puts more weight on immediate reward, if it is 1 every reward is weighted equaly.

The agent is controlled by a policy, which is denoted as π. In general this policy maps states to a probability distribution over actions: $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$.

With these definitions we can now formalize the transitions in figure 2.1. The action $a_t \sim \pi(\cdot|s_t)$ is drawn from the policy given the state. With this action, the next state $s_{t+1} \sim \mathcal{P}(\cdot|s_t, a_t)$ is sampeled from the transition model. Furthermore the reward $r_t = \mathcal{R}(s_t, a_t, s_{t+1})$ is calculated. Then the loop starts again with s_{t+1}. If there is a fixed length T, after which the state is reset, the MDP is called episodic. The sequence of states, actions and rewards in an episode is called trajectory and denoted as τ. Every trajectory has a cumulative return $J(\tau) = \sum_{t=0}^{T-1} \gamma^t r_{t+1}$. When the length $T = \infty$ the MDP is called non-episodic. In this case a $\gamma < 1$ prevents the cumulative reward from becoming infinite.

Formally, the goal of RL can be defined as finding the optimal policy π^*, under which the expected return from all states reaches its maximum. This can be formalized by defining a function $\mathcal{J}^\pi = \mathbb{E}[J|\pi]$ as the expexted return over all possible trajectories given policy π. Then the goal is to find:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathcal{J}^\pi$$

In the following, we will introduce algorithms, that try to learn the optimal policy.

2.1.2 Reinforcement Learning Algorithms

Reinforcement Learning Algorithms can be divided into two main approaches. One uses the value function method, where the goal is to learn an optimal policy through the value function. This is achieved by first learning a function that can predict the expected return given a state-action pair:

$$Q^\pi(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_\pi[J|\boldsymbol{s}, \boldsymbol{a}]$$

The expected return starts at state s with action a and subsequently, follows the policy π, collecting the reward at every step. In other words, it is a rating for how good of a choice a is, given the state s. A policy can be extracted from the Q^π function by taking the $\arg\max_a Q^\pi(s, a)$ for every state s. In order to achieve good performance with this approach, the learned Q function has to be a good estimate of the true one. This can be challenging, especially in large action and state spaces.

It is of that reason that directly optimizing the policy can be advantageous, which is the other approach called policy search. For this, a parametrized policy is used:

$$\boldsymbol{a} \sim \pi(\boldsymbol{a} \mid \boldsymbol{s}; \boldsymbol{\theta})$$

The policy has to be differentiable with respect to the parameters $\boldsymbol{\theta}$, so these policy parameters can be updated via gradient descent. The general idea is to weigh the gradient of a trajectory by the reward it accumulates. A problem that arises in this approach is that the variance of the gradient can be very high, leading to unstable training.

Additionally, there is a hybrid out of both called the actor-critic method. It tries to exploit the advantages of both approaches to achieve superior performance. Actor-Critic algorithms learn a Q function and a parametrized policy. This is done in an alternating manner, in which Q and π slowly converge to an optimum. The survey Arulkumaran et al. [2] explains these approaches in greater detail. One example of an actor-critic algorithm is Soft Actor Critic (SAC), which is the algorithm used throughout this thesis.

Soft Actor Critic

What makes SAC special is its inclusion of the entropy of the policy in its objective:

$$J(\pi) = \sum_{t=0}^{T-1} r_{t+1} + \alpha \mathcal{H}\left(\pi\left(\cdot \mid \mathbf{s}_t\right)\right)$$

Adding entropy to the objective encourages the policy to encode as much information as possible. It can also be seen as a regularization, which prevents overfitting.

SAC is an off-policy algorithm. Off-policy is a description of the manner in which data is collected. More specifically it means that the transitions don't have to be collected from the current best policy. Instead, older transitions can be reused, increasing the sample efficiency. In order to reuse the samples, they are stored in the replay buffer $\mathcal{D}$. While

training new transitions are added to the replay buffer. Given that the buffer has a fixed size, older transitions will eventually be replaced by new ones. In other words, the replay buffer represents the recent memory of observed transitions.

To learn the Q function, a batch of transitions is randomly sampled from the replay buffer. These transitions are used to refine the Q function approximation via the Bellman equation. For further details on learning the Q function refer to Haarnoja et al. [12].

The policy of SAC is a gaussian distribution, squashed by a $\tanh$ where we use a neural network with parameters θ to evaluate mean and variance in each state. Its update is computed by taking the gradient of the Q function with respect to the policy parameters:

$$\pi_\theta^* = \arg\max_\theta \mathop{\mathrm{E}}_{\substack{s\sim\mathcal{D} \\ a\sim\pi_\theta}} \left[Q^{\pi_\theta}(s,a) - \alpha \log \pi_\theta(a \mid s)\right] \tag{2.1}$$

The second term of equation 2.1 is just the entropy of the policy. It carried over from the modified objective of SAC. The problem is that the expectation in equation 2.1 can not be differentiated with respect to θ. To fix this issue, SAC employs the reparametrization trick. The goal is to divide a sample drawn from $\pi_\theta(\cdot \mid s)$ into a deterministic and therefore differentiable function. This function takes the state and policy parameters as well as some independent noise, which is responsible for the stochasticity:

$$\tilde{a}_\theta(s,\xi) = \tanh\left(\mu_\theta(s) + \sigma_\theta(s) \odot \xi\right), \quad \xi \sim \mathcal{N}(0, I)$$

Plugging this back into equation 2.1 results in the optimization of the policy:

$$\pi_\theta^* = \arg\max_\theta \mathop{\mathrm{E}}_{\substack{s\sim\mathcal{D} \\ \xi\sim\mathcal{N}}} \left[Q^{\pi_\theta}\left(s, \tilde{a}_\theta(s,\xi)\right) - \alpha \log \pi_\theta\left(\tilde{a}_\theta(s,\xi) \mid s\right)\right]$$

The training process of SAC is summarized by the algorithm 1.

2.2 Safe Reinforcement Learning

The goal of safe reinforcement learning is to fulfil state-dependent constraints at every timestep. Safe exploration methods are one way to achieve this goal. They require that the constraints are not violated during the entire training process [10]. One approach to

Algorithm 1 Summarized Soft Actor Critic

Initialize policy parameters θ and Q function parameters ϕ
repeat
    Observe state s and select action $a \sim \pi_\theta(\cdot|s)$ from the current policy
    Apply a to the environment
    Store the resulting transition in replay buffer $\mathcal{D}$
    if n transitions are sampeled **then**
        select random batch of transitions from $\mathcal{D}$
        Update the Q_ϕ function by one step of gradient descent using the batch of data
        Update the policy π_θ by one step of gradient ascent using the batch of data
    end if
until convergence

safe exploration is to define a safe policy, that tries to keep the agent within the safe region [1, 13]. Another approach is to exploit prior knowledge of the system (policies, value functions) to define a safe initial region. This safe area can then be gradually increased by collecting more information from the environment [9, 3]. A problem is that defining these policies requires a lot of work. Lastly, other approaches exploit the model information of constraints and combine that with model model-free RL algorithms [6, 4]. Here the agent computes a safe action using constrained optimization techniques at every time step. ATACOM, the safe exploration algorithm used in this thesis, also belongs to this category.

2.2.1 ATACOM

ATACOM is a method that incorporates the constraints in the learning process of the policy. The basic idea is to shrink the action space to only those actions, which will not result in a constraint violation. Therefore, they are never considered in the learning process and will not be used by the learned policy.

There are two types of constraints that are accommodated:

$$\boldsymbol{f}(\boldsymbol{q}) = \mathbf{0}, \quad \boldsymbol{g}(\boldsymbol{q}) \leq \mathbf{0} \tag{2.2}$$

$\boldsymbol{f}$ is called equality constraints and $\boldsymbol{g}$ is called inequality constraints. In order to combine these constraints into one function, the inequality constraints are turned into equality

constraints. This is done by adding a slack variable $\boldsymbol{\mu}$ to the inequality constraints. The result is the constraint function:

$$\boldsymbol{c}(\boldsymbol{q}, \boldsymbol{\mu}) = \begin{bmatrix} \boldsymbol{f}(\boldsymbol{q}) & \boldsymbol{g}(\boldsymbol{q}) + \frac{1}{2}\boldsymbol{\mu}^2 \end{bmatrix}^\top = \mathbf{0} \tag{2.3}$$

The goal is to limit the actions such that $\boldsymbol{c}(\boldsymbol{q}, \boldsymbol{\mu}) = \mathbf{0}$ for all $\boldsymbol{q}$ across every transition. This is achieved by taking the time derivative of equation 2.3.

$$\dot{\boldsymbol{c}}(\boldsymbol{q}, \boldsymbol{\mu}, \dot{\boldsymbol{q}}, \dot{\boldsymbol{\mu}}) = \begin{bmatrix} \boldsymbol{J}_f(\boldsymbol{q}) & \mathbf{0} \\ \boldsymbol{J}_g(\boldsymbol{q}) & \mathrm{diag}(\boldsymbol{\mu}) \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{q}} \\ \dot{\boldsymbol{\mu}} \end{bmatrix} = \boldsymbol{J}_c(\boldsymbol{q}, \boldsymbol{\mu}) \begin{bmatrix} \dot{\boldsymbol{q}} \\ \dot{\boldsymbol{\mu}} \end{bmatrix}$$

Any choice of $[\dot{\boldsymbol{q}}, \dot{\boldsymbol{\mu}}]^T$ for which $\dot{\boldsymbol{c}}(\boldsymbol{q}, \boldsymbol{\mu}, \dot{\boldsymbol{q}}, \dot{\boldsymbol{\mu}}) = \mathbf{0}$ does not violate the constraints. That means the Null space of $\boldsymbol{J}_c(\boldsymbol{q}, \boldsymbol{\mu})$ is the space of all velocities which do not violate the constraints. It can be defined as $\boldsymbol{J}_c(\boldsymbol{q}, \boldsymbol{\mu})\boldsymbol{N}_c(\boldsymbol{q}, \boldsymbol{\mu}) = \mathbf{0}$.

Using $\boldsymbol{\alpha} \sim \pi(\cdot|\boldsymbol{s})$ not as an action which is sent to the agent but as a coordinate of the Nullspace enables the exploration in the constrained tangent space:

$$[\dot{\boldsymbol{q}}, \dot{\boldsymbol{\mu}}]^T = \boldsymbol{N}_c(\boldsymbol{q}, \boldsymbol{\mu})\boldsymbol{\alpha}$$

The only problem is that the velocities need to be converted back to actions. For this ATACOM assumes that the function $\boldsymbol{a} = \Lambda(\dot{\boldsymbol{q}})$ exists. In the context of robotics that would be an inverse dynamics model. Algorithm 2 summarizes the idea.

Algorithm 2 Basic idea of ATACOM

1: Start in a safe state $(\boldsymbol{q}_0, \boldsymbol{\mu}_0)$, which do not violate constaints
2: **while** episode is not terminated **do**
3: Sample action from policy $\boldsymbol{\alpha}_t \sim \pi(\cdot|\boldsymbol{s}_t)$
4: Observe $\boldsymbol{q}_t$ from state $\boldsymbol{s}_t$
5: Compute $\boldsymbol{J}_c(\boldsymbol{q_t}, \boldsymbol{\mu_t})$
6: Compute $\boldsymbol{N}_c(\boldsymbol{q}, \boldsymbol{\mu})$
7: Compute tangent space velocities $[\dot{\boldsymbol{q}}_t, \dot{\boldsymbol{\mu}}_t]^T = \boldsymbol{N}_c(\boldsymbol{q}_t, \boldsymbol{\mu}_t)\boldsymbol{\alpha}_t$
8: Integrate the slack variable $\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t + \dot{\boldsymbol{\mu}}_t + \Delta T$
9: Apply control action $\boldsymbol{a_t} = \Lambda(\dot{\boldsymbol{q}}_t)$ to the environment
10: Provide obtained transition to RL algorithm
11: **end while**

Instead of the constraints from 2.2 ATACOM actually uses viability constraints. Viability constraints essentially limit the velocity when the constraints are close to their limits, which prevents overshooting. The derivations are analogous to the normal constraints but add complexity to the equations. Additionally, there is an error correction term. This term corrects the constrained violations which occur when discretizing time-continuous systems. These points are explained in greater detail by Liu et al. [18].

3 Methodology

In this chapter, we are going to define the setup of the conducted experiments. These experiments should evaluate ATACOMs' performance on more general tasks. We will first introduce a base environment upon which all tasks are built as well as the agents we deployed with their respective constraints. Then we will introduce each task, explain their setup and desired behaviour as well as motivate their reward function. Lastly, we will list all experiments with their respective hyperparameters.

3.1 Experimental Setting

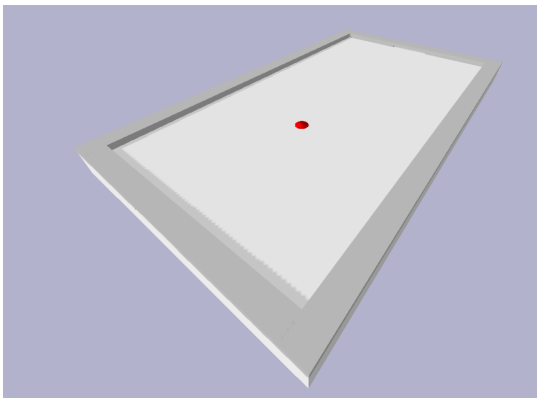
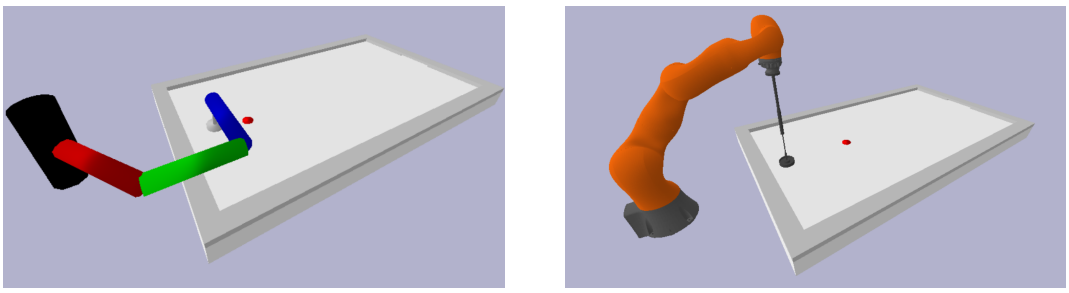


Figure 3.1: Base environment in PyBullet

As already explained in section 1.2 four tasks will be considered in the experiments. These tasks will build upon the same base environment which contains the air hockey table shown in figure 3.1. The table has a goal on each end and a puck that can move freely within the bounds of the table. The goal's width is one-fourth of the width of the table. The initial puck position and velocity depend on the task.

The system is simulated by PyBullet [5] with a model obtained by identification of a real air hockey table from Liu et al. [19]. We use an implementation of this model introduced by Liu et al. [18]. With this, we can accurately simulate the interaction between the puck and all surfaces on the table.



(a) Planar robot agent (b) Iiwa robot agent

Figure 3.2: Agents used in the environment

To play air hockey we also need to be able to interact with the puck. As a human player, this interaction will be via an air hockey mallet. The agent which will move the mallet across the table is a robot arm.

3.1.1 Agents

To test ATACOM in domains with arbitrary complexity we are going to use two different robot arms as agents. This increases the variety of constraints that are tested because each agent needs its own set of constraints to operate safely.

The simpler agent shown in figure 3.2a is a toy robot taken from Liu et al. [18], which only exists in the simulator. It consists of 3 links, which are connected via revoluting joints. The joints can only rotate around the z-axis. Thus the robot can only move on an XY plane over the table and has 3 degrees of freedom. At the end of the third link, the mallet is mounted in such a way that it always hovers slightly above the table. This robot has a small action space and simple kinematics, resulting in swiftly converging RL algorithms. We use this agent for searching and refining reward functions as well as analysing the performance in chapter 4.

In order to show that our results can be translated to a complex agent, we use the IIWA robot arm shown in Figure 3.2b. It is an industrial robot manufactured by Kuka which has 7 degrees of freedom. The mallet is mounted on an extension rod, which is reused from 2. It consists of an aluminium rod, gas spring as well as a universal joint, on which the mallet is connected. The purpose of the gas spring is to make the system more resilient to small errors, by adding a compressible component, which takes the force away from the

table. The universal joint's job is to keep the mallet parallel to the table at all times. We will show preliminary results of this agent, while we leave an in-depth analysis to future works.

Both agents have an initial position that is used in all tasks. The agents are controlled by torque, which is directly applied to each joint. Thus, there is no inherent safety in the controller. To prevent collisions with the environment, we define constraints that have to be fulfilled at all times.

3.1.2 Constraints

Because we want to improve the results of Liu et al. [18] we will use the constraints they defined for each agent. The simple planar robot uses a total of six inequality constraints. Three of them prohibit the mallet from leaving the table boundaries. The other three enforce the joint position and velocity limits. Formally they can be written as:

$$g_1 : -x_{ee} + x_{table,l} < 0, \qquad g_2 : -y_{ee} + y_{table,l} < 0, \qquad g_3 : y_{ee} - y_{table,u} < 0,$$
$$g_{4,5,6} : q_i^2 - q_{i,l}^2 < 0, \qquad |\dot{q}_i| - \dot{q}_{i,l} < 0, \qquad \forall i \in \{1, 2, 3\}$$

where (x_{ee}, y_{ee}) is the position of the robot end-effector, $x_{table,l}, y_{table,l}, y_{table,u}$ are the boundaries of the air-hockey table, $q_i, \dot{q}_i$ refers to position and velocity of the i-th joint, and $q_{i,l}, \dot{q}_{i,l}$ are the position and velocity limit for the joint i [18].

The IIWA uses the same basic inequality constraints. It however needs some more to ensure safety. Because the kinematics are not constrained to moving on a plane, the z-direction has to be taken into consideration. To ensure the end-effector is moving on the table surface, an equality constraint is added. Furthermore, two inequality constraints prevent the 4th and 6th links of the robot to collide with the table. The resulting constraint set is:

$$f : \quad z_{ee} - z_{\text{table}} = 0, \quad g_1 : \quad -z_4 + \hat{z}_{4,l} < 0, \quad g_2 : -z_6 + \hat{z}_{6,l} < 0,$$
$$g_3 : \quad -x_{ee} + x_{\text{table},l} < 0, \quad g_4 : \quad -y_{ee} + y_{\text{table},l} < 0, \quad g_5 : y_{ee} - y_{\text{table},u} < 0,$$
$$g_{6,7,\cdots 11} : \quad q_i^2 - q_{i,l}^2 < 0, \qquad \forall i \in \{1, 2, \cdots 6\}$$

where f ensures that the end-effectors height z_{ee} is always on the table height z_{table}. The inequality constraints g_1, g_2 prevent the height of the 4th and 6th link to go below their limits $\hat{z}_{4,l}$ and $\hat{z}_{6,l}$ respectively. g_3, g_4, g_5 force the end-effector to be in the bounds of the table and $g_{6,7,\cdots 11}$ are the joint limit constraints.

In order to monitor the constraint violations accurately all collisions are disabled in the simulator. Hence all constraints have to be actively guarded and cannot just be fulfiled passively through collisions with the environment.

3.2 Tasks

After introducing the base environment, agents and constraints we can now focus on the task-specific environments. For that, we will define the purpose of each task, specify its properties and motivate the reward function. In the following $\boldsymbol{p}_{\text{ee}}, \boldsymbol{p}_{\text{puck}}, \boldsymbol{p}_{\text{goal}}$ are used for the 2D positions of the mallet, the puck and the goal. We access the x and y component of the mallet position with $\boldsymbol{p}_{\text{ee},x}$ and $\boldsymbol{p}_{\text{ee},y}$. This notion is used analogues for all other 2D positions and velocities. The x-axis always runs parallel to the long side of the table while the y-axis is parallel to the short side. The penalty of the actions is defined as $\lambda = 0.001 \cdot \|\boldsymbol{a}\|$ and $\boldsymbol{v}_{\text{puck}}$ denotes the 2D linear velocities of the puck. Additionally the function $sign(x)$ is defined as:

$$sign(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{else} \end{cases}$$

The reward functions in these tasks are convoluted. A simple binary reward will not suffice because the desired behaviours are too complex. Thus good performance can only be achieved with a sufficiently informative reward. Any flags that are used within the reward functions are added to the observation, such that the reward is markovian.

3.2.1 Hit

The Hit task is essential for playing air hockey. Its purpose is to hit a stationary puck to score a goal. The secondary objective is to prevent the opponent from taking control of the puck, so they can not maximize their chances of scoring a goal. This task will only hit

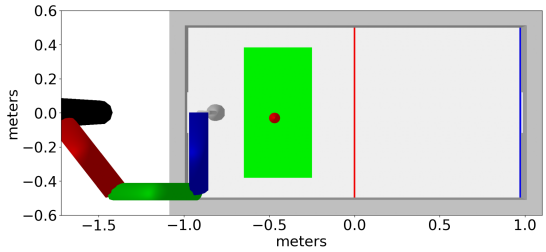


Figure 3.3: Hit environment with green puck spawn area and blue termination line

a puck that is not moving, thus it is used at the beginning of a match or after the robot has taken control of the puck.

To isolate just the hitting task we assume that the puck will be in a reasonable position at the beginning of the episode. This position is randomly initialized in the green area in figure 3.3 with a uniform distribution. The initial puck velocity is always zero. An episode terminates when the puck hits the opponent's goal or backboard. Visually this is the puck touching the blue line in figure 3.3. Additionally, the episode terminates after a fixed horizon.

In order to score, the puck has to have enough velocity to reach the goal. Additionally, when reaching the goal the puck has to be close to the middle. One tactic is to shoot as straight as possible towards the goal. Another one is to use the sides of the table to bounce the puck. While the puck can ricochet between the sides an arbitrary amount of times, it reduces the forward velocity and therefore makes it easier to defend. So only one or two bounces are feasible to preserve momentum. A higher velocity is better in general as it makes defending harder. It should however not come at the cost of accuracy.

With that in mind, our reward function is defined as:

$$r = \begin{cases} \exp[-8\|\boldsymbol{p}_{\text{ee}} - \boldsymbol{p}_{\text{puck}}\|] \cdot \text{clip}(\theta, 0, 1)^2 - \lambda & \text{if has not hit} \\ 0.5 + 2 \cdot \min(1, \frac{\boldsymbol{v}^4_{\text{puck},x}}{4}) - \lambda & \text{if has hit and } \boldsymbol{p}_{puck,x} \leq 0.7 \\ 0.5 + 2 \cdot \min(1, \frac{\boldsymbol{v}^4_{\text{puck},x}}{4}) + r_{\text{middle}} - \lambda & \text{if has hit and } \boldsymbol{p}_{\text{puck},x} > 0.7 \\ 200 & \text{if scored a goal} \end{cases}$$

where $r_{\text{middle}} = \frac{10}{0.1\sqrt{2\pi}} \exp[\frac{-\boldsymbol{p}^2_{\text{puck},y}}{0.02}]$ gives a reward when the pucks y-position is close to

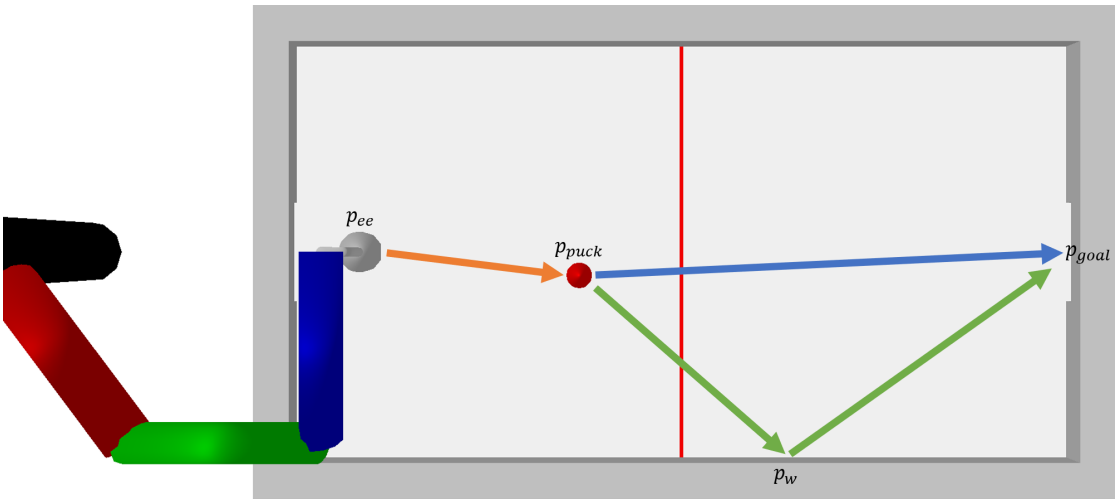


Figure 3.4: Illustation of the vectors in the pre-hit reward. The green and blue vectors each describe a desired trajectory the path should take.

the middle. $\theta = \max(\theta_{\text{side}}, \theta_{\text{straight}})$ is weight for the distance reward, where $\theta_{\text{straight}} = \left\langle \frac{\boldsymbol{p}_{\text{puck}} - \boldsymbol{p}_{\text{ee}}}{\|\boldsymbol{p}_{\text{puck}} - \boldsymbol{p}_{\text{ee}}\|}, \frac{\boldsymbol{p}_{\text{goal}} - \boldsymbol{p}_{\text{puck}}}{\|\boldsymbol{p}_{\text{goal}} - \boldsymbol{p}_{\text{puck}}\|} \right\rangle$ and $\boldsymbol{p}_{\text{goal}}$ is the 2D position of the middle of the goal. $\theta_{\text{side}} = \left\langle \frac{\boldsymbol{p}_{\text{puck}} - \boldsymbol{p}_{\text{ee}}}{\|\boldsymbol{p}_{\text{puck}} - \boldsymbol{p}_{\text{ee}}\|}, \frac{\boldsymbol{p}_{w} - \boldsymbol{p}_{\text{puck}}}{\|\boldsymbol{p}_{w} - \boldsymbol{p}_{\text{ppuck}}\|} \right\rangle$ with $\boldsymbol{p}_w = (w, \text{sign}(\boldsymbol{p}_{\text{puck},y}) \cdot y_{\text{table},l})$ and $w = \frac{|\boldsymbol{p}_{\text{puck},y}| \cdot \boldsymbol{p}_{\text{goal},x} + \boldsymbol{p}_{\text{puck},x} \cdot \boldsymbol{p}_{\text{goal},y} - y_{\text{table},l}(\boldsymbol{p}_{\text{puck},x} + \boldsymbol{p}_{\text{goal},x})}{|\boldsymbol{p}_{\text{puck},x}| + \boldsymbol{p}_{\text{goal},y} - 2y_{\text{table},l}}$.

The idea behind this reward is, that if the puck has not yet been hit, it encourages the end-effector to get close to the puck. However, the approach of the end-effector is limited by θ to two possible options, illustrated in figure 3.4. The orange vector between mallet and puck has to be roughly parallel to either the blue vector between puck and goal or the green vector between the puck and the point on the side of the table, which scores a goal after the bounce. The calculation for this bounce point assumes that the incoming angle is equal to the outcoming angle. That is a false model for air hockey as it does not consider the angular rotation of the puck. It is however a useful model for approximating the rough approach, that the agent should take. The post-hit reward is always bigger than the reward before the hit because of the constant. Hence hitting the puck is always desirable for the agent. The other factor of the post-hit reward is the clipped velocity of the pucks x-direction. The clipping is done so the velocity reward can not overpower everything

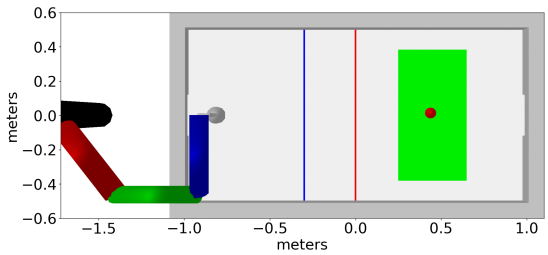


Figure 3.5: Defend environment with green puck spawn area and blue termination line. The episode only terminates if the blue line is crossed after the puck bounced of the mallet or wall.

else, preventing a fast but inaccurate shot. Additionally, there is a bonus reward when the puck is near the goal. It is high when the pucks' y-position is close to 0, which results in a trajectory hitting the goal. Lastly, scoring a goal results in a large one time bonus.

This reward is designed in a way to avoid the downfalls a delayed reward entails. If we would just reward scoring a goal, the reward has to be propagated back to the hit. This is very time consuming and usually requires special frameworks [15].

We consider an episode successful if the puck hits the goal.

3.2.2 Defend

The purpose of the defence task is to stop an incoming puck within the reach of the agent, in order to get an accurate shot afterwards. It is a risky manoeuvre because small errors can cause the puck to coast to the opponent, giving them a free shot. However, when the robot successfully stops the puck, it enables an accurate shot of a still puck that maximizes the chance of scoring a goal.

We emulate the opponent by spawning the puck in the area, over which the opponent's hit task would operate. This is visualised in figure 3.5 by the green area. As with the hit task, the probability for every point is uniformly distributed. The initial velocity of the puck is also chosen at random. For that, we sample a linear velocity from the uniform distribution $U(1, 2.2)$ and an angle from $U(-0.5, 0.5)$, with which both the x and y linear velocities are computed. The initial puck rotation is sampled from $U(-1, 1)$. The episode

terminates when the puck crosses the blue line of figure 3.5, after bouncing off the wall or mallet. Additionally, the episode terminates after a fixed horizon.

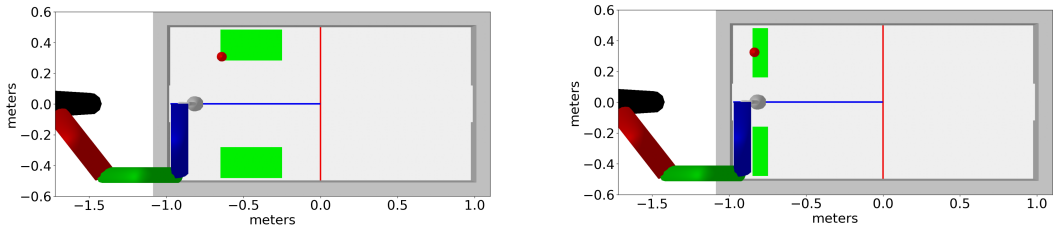
A tactic for defending is reducing the momentum of the puck by gently catching it. This is however very hard to accomplish because the agent has to match the speed of the puck accurately in order to prevent the puck from bouncing away. Another approach is to hit the puck at such an angle that it ricochets between the sides of the table until it lost all momentum. Considering these desired behaviours, the reward is:

$$r = \begin{cases} -50 & \text{if puck scores goal} \\ -1 - \lambda & \text{if puck touched the back wall} \\ 0.3\,\exp[-3\,|-0.6-\boldsymbol{p}_{\text{ee},x}|] + 0.7\,r_y - \lambda & \text{if has not hit} \\ 1 + 3\,\exp[-3\,|\boldsymbol{p}_{\text{puck},y}|] + r_x + 5\,r_{\text{vel}} - \lambda & \text{if has hit} \end{cases}$$

where $r_y = \frac{1}{0.4\sqrt{2\pi}}\exp[(\frac{|\boldsymbol{p}_{\text{ee},y}-\boldsymbol{p}_{\text{puck},y}|-0.08}{0.02})^2]$, $r_{\text{vel}} = \exp[-(5\,\|\boldsymbol{v}_{\text{puck}}\|)^2]$ and $r_x = \exp[-5\,|\boldsymbol{p}_{\text{puck},x}+0.6|]$.

If the puck has not yet been hit, the goal of the reward function is to reduce the distance between the puck and the end-effector on the y-axis. The x-axis is fixed to -0.6 and weighted less, so the agent is incentivised to move on this x-axis to reduce the momentum of the puck by catching it. The reward for the y-axis is off-centre by 0.08 to encourage deflecting the puck into the side walls. When the puck has been hit three puck properties accumulate reward: The first two are the y-position being close to zero and the x-position being close to -0.6. This is done to encourage the agent to stop the puck at the point, where the subsequent hit task performs best. The third property is velocity, which should be zero for the maximal reward. Additionally, the post-hit reward has a constant that ensures it always being bigger than the pre-hit reward. When the puck touched the back wall the reward is a constant -1 because otherwise, an agent behaviour is trapping the puck in a corner of the table. While this takes control of the puck, it is very hard to move the puck out of the corner. Thus this behaviour is prohibited by a negative reward. Lastly, there is a punishment when the agent fails to defend, letting the opponent score a goal.

An episode is considered a success if the puck has not touched the back wall or the termination line at the end of an episode.



(a) Side initialization

(b) Bottom initialization

Figure 3.6: Prepare environments with green initilization areas and blue termination line

3.2.3 Prepare

The goal of the *prepare* task is to improve the puck position for the subsequent hitting task. A position is considered bad when it is close to the side or back wall and good when it is more centred, allowing a better approach for the agent. Improving these positions is difficult because very precise and gentle movements are required. A small error can lose control of the puck and give the opponent a free shot.

In order to reduce the complexity, we split this task into two subtasks. The first one shown in figure 3.6a considers all puck positions which only need to move on the y-axis towards the middle. Figure 3.6b illustrates the second subtask, in which the puck needs to move closer to the centre on both the y and the x-axis. In both figures, the green area represents the possible initial puck positions, which are uniformly distributed. If the puck touches the blue line or the horizon is reached the episode terminates. There is an area close to the short wall of the table which is not considered in the tasks. We decided to exclude this area for now because it is very hard to find the desired behaviour that performs reliably in these circumstances.

Side initialization

The desired behaviour for the side initialization is a gentle straight shot into the wall, which results in the puck bouncing off the wall with little momentum. This momentum should carry the puck to the middle of the table, where it should stop. When the puck's velocity carries it further than desired or not far enough the task can be repeated. The reward function used to encapsulate the behaviour is:

$$r_s = \begin{cases} \exp[-8\,\|\boldsymbol{p}_{\text{ee}} - \boldsymbol{p}_{\text{puck}}\| \cdot \text{clip}(\cos\theta_s, 0, 1)^2] - \lambda & \text{if has not hit} \\ 1 + r_{\text{vel}} + 0.5 - \|\boldsymbol{p}_{\text{ee}} - \boldsymbol{p}_i\| - \lambda & \text{if has hit and } \boldsymbol{p}_{\text{puck},y} < 0.47 \\ 100\ \exp[-2\,\|\boldsymbol{v}_{\text{puck}}\|] & \text{if touches termination line} \\ -\lambda & \text{else} \end{cases}$$

with $\cos\theta_s = \left\langle \frac{\boldsymbol{p}_{\text{puck}} - \boldsymbol{p}_{\text{ee}}}{\|\boldsymbol{p}_{\text{puck}} - \boldsymbol{p}_{\text{ee}}\|}, [0, sign(\boldsymbol{p}_{\text{puck},y})]^T \right\rangle$ and $\boldsymbol{p}_i$ being the initial mallet position. $r_{vel} = \max(0, 1 - (10\,(\exp[|\boldsymbol{v}_{\text{puck},x}|] - 1)))$ punishes velocity on the x axis.

The reward encourages the agent to move the mallet to the puck, but similar to the hit reward the approach is limited to being perpendicular to the side. When the puck has been hit, the agent is incentivised to move back to its initial position. Additionally, the pos-hit reward heavily punishes velocity on the x-axis in order to prevent movement on this axis. If the puck is too slow to bounce off and stops on the side the reward is zero. This position is hard to recover from and should therefore be prohibited. When the puck reaches the termination line in the middle of the table it gets a one time reward which scales with the inverse of the puck's velocity.

We consider an episode successful if the puck position at the last steps fullfills $-0.8 < \boldsymbol{p}_{\text{puck},x} < -0.4$ and $|\boldsymbol{p}_{\text{puck},y}| < 0.25$

Bottom initialization

For the bottom initialization, the tactic is similar. Instead of only moving on the y-axis we also want a slight upwards movement on the x-axis, to get in front of the agent's initial position. To much momentum on the y-axis ends in losing the puck in the opponents region, so being precise is important. The reward is defined as:

$$r_b = \begin{cases} \exp[-8\,\|\boldsymbol{p}_{\text{ee}} - \boldsymbol{p}_{\text{puck}}\| \cdot \text{clip}(\cos\theta_b, 0, 1)^2] - \lambda & \text{if has not hit} \\ 1 + r_x + 2 - |\boldsymbol{v}_{\text{puck},y}| + 0.5 - \|\boldsymbol{p}_{\text{ee}} - \boldsymbol{p}_i\| - \lambda & \text{if has hit and } \boldsymbol{p}_{\text{puck},y} < 0.47 \\ 100\ \exp[-2\,\|\boldsymbol{v}_{\text{puck}}\|] & \text{if touches termination line} \\ -\lambda & \text{else} \end{cases}$$

where $\cos\theta_b = \left\langle \frac{\boldsymbol{p}_{\text{puck}} - \boldsymbol{p}_{\text{ee}}}{\|\boldsymbol{p}_{\text{puck}} - \boldsymbol{p}_{\text{ee}}\|}, [0.2, 0.8\,sign(\boldsymbol{p}_{\text{puck},y})]^T \right\rangle$, $r_x = \frac{1}{0.1\sqrt{2\pi}} \exp[(\frac{0.75\,\boldsymbol{p}_{\text{puck},x}}{0.01})^2]$ and $\boldsymbol{p}_i$ is the inital mallet position.

The pre-hit reward stays the same, just the vector of the desired approach is replaced by a slightly upwards facing one. This should encourage the agent to hit the puck upwards in a controlled fashion. There are some more changes in the post-hit reward, where now the puck velocity on the y-axis is punished and the puck is rewarded for moving upwards on the x-axis. This encourages a slow movement of the puck. To ensure that the puck moves at all, being stuck on the side of the table results in no reward and there is a big bonus for hitting the termination line in the middle, which again scales inverse to the puck velocity.

An episode is successful if at the last timestep the $-0.8 < \boldsymbol{p}_{\text{puck},x} < -0.5$ and $|\boldsymbol{p}_{\text{puck},y}| < 0.25$

3.2.4 Repel

The repel task is used when an incoming puck is too hard to take control of. Instead of trying to reduce the momentum like the defence task, to goal is to repel the puck with as much momentum as possible, making it hard for the opponent to take control of the puck. The secondary objective is to hit the goal. It is however not desired to hit with less velocity in order to fulfil this objective.

The initialisation of the puck is the same as for the defend task of section 3.2.2. The only difference is that the termination line is at the opponent's goal.

To achieve a high velocity the agent should build up momentum to take a swing at the puck. This is hard when the puck is already moving fast because the swing has to intercept the trajectory of the puck at a perfect time. Thus a more reliable strategy for fast-moving pucks is to match the pucks y-position and only give it a little hit. Because the puck is incoming with a lot of momentum this small hit is enough to propel it back with a high velocity. Hence the reward function is:

$$r = \begin{cases} -50 & \text{if puck scores own goal} \\ -1 - \lambda & \text{if puck touched the back wall} \\ 0.3\,\exp[-3\,|-0.6 - \boldsymbol{p}_{ee,x}|] + 0.7\,r_y - \lambda & \text{if has not hit} \\ 1 + \boldsymbol{p}_{\text{puck},x} + 0.98 + min(\boldsymbol{v}^3_{\text{puck},x}, 5) - \lambda & \text{if has hit and } \boldsymbol{p}_{\text{puck},x} \leq 0.9 \\ 1 + \boldsymbol{p}_{\text{puck},x} + 0.98 + \min(\boldsymbol{v}^3_{\text{puck},x}, 5) + r_g - \lambda & \text{if has hit and } \boldsymbol{p}_{\text{puck},x} > 0.9 \end{cases}$$

where $r_y = \frac{1}{0.4\sqrt{2\pi}}\exp[(\frac{|\boldsymbol{p}_{\text{ee},y} - \boldsymbol{p}_{\text{puck},y}| - 0.08}{0.02})^2]$ and $r_g = 100\exp[-3\,|\boldsymbol{p}_{\text{puck},y}|]$.

The pre-hit reward incentivizes the agent to move the mallet to match the y-position of the puck while staying at the arbitrary x-position -0.6. The x-axis is weighted less the puck can build up momentum for a swing without being punished harshly. In the post-hit case moving along the x-axis and a high linear velocity along the x-axis are rewarded. To keep the values in a reasonable range the velocity reward is capped at 5. Additionally, there is a bonus for being around 0 on the y-axis when the puck approaches the goal. Conceding a goal or missing the puck is heavily punished to avoid it at all costs.

An episode is successful when the puck has been hit towards the opponents' side. As this task is too easy, we will use the metric of scoring a goal, even though it isn't the primary objective. Thus an episode is successful when the goal has been scored.

3.3 Experiements

We will conduct an experiment with the planar robot solving each task. Additionally, we will deploy the IIWA in the same environments. We will name each experiment by combining the agent and task name, for example, the repel task with the IIWA robot is named *IiwaRepel*. To address all experiments with the same agent we will use *PlanarAirHockey* for the planar robot and *IiwaAirHockey* for the IIWA robot.

In order to get reliable results, each experiment will consist of 25 runs. The exception is the *PlanarHit* experiment, where a total of 100 runs are used to increase the validity of the results. The environment parameters for the *PlanarAirHockey* experiments and the *IiwaAirHockey* experiments can be found in table 3.1 and 3.2 respectively. We use the same parameters for ATACOM as Liu et al. [18], which can be found in their paper.

SAC is used as the reinforcement learning algorithm because in the original paper it outperformed the other common algorithms in the air hockey domain [18]. To achieve good results we fine-tuned the actor and critique learning rate as well as the target entropy of SAC. This was done by combining all actor and critique learning rates out of $[1e^{-4}, 5e^{-4}, 1e^{-3}, 3e^{-3}]$ with all target entropies from $[0, -3, -6, -9]$. Because the environments are similar and the reward functions operate on the same scale, the optimal parameters only happen to change with the agent. Additionally, we chose the optimal actor/critic network size out of $[[64, 64], [128, 128], [256, 256]]$. All SAC hyperparameters are listed in table 3.3. The implementation of SAC used in this thesis was provided by MushroomRL [7].

Environment Parameter	*PlanarHit*	*PlanarDefend*	*PlanarPrepare*	*PlanarRepel*
episode duration	2s	3s	2s	3s
discount factor	0.99			
simulation step size	1 / 240s			
acceleration limit $\boldsymbol{a}_{max}$	$[10, 10, 10]$			
velocity limit $\boldsymbol{a}_{max}$	$[2.3562, 2.3562, 2.3562]$			
intermediate steps	4			

Table 3.1: Parameters for *PlanarAirHockey* experiments

Environment Parameter	*IiwaHit*	*IiwaDefend*	*IiwaPrepare*	*IiwaRepel*
episode duration	2s	3s	2s	3s
discount factor	0.99			
simulation step size	1 / 500s			
acceleration limit $\boldsymbol{a}_{max}$	$[10, 10, 10, 10, 10, 10]$			
velocity limit $\boldsymbol{a}_{max}$	$[1.4835, 1.4835, 1.7453, 1.3090, 2.2689, 2.3562]$			
intermediate steps	10			

Table 3.2: Parameters for *IiwaAirHockey* experiments

SAC parameter	*PlanarHit*	*PlanarDefend*	*PlanarPrepare*	*PlanarRepel*	*IiwaAirHockey*
actor/critic learning rate			$1e^{-3}$		
target entropy	-3	-3	-3	-3	-6
epochs	1200	700	400	400	400
steps per epoch			5000		
steps per fit			1		
episodes per test			50		
actor/critic network size			[128, 128]		
batch size			64		
inital replay size			5000		
max replay size			200000		
soft update coefficient			$1e^{-3}$		
warm-up transitions			10000		
learning rate alpha			0.0003		

Table 3.3: Parameters for SAC

4 Experimental Evaluation

4.1 Learning Performance

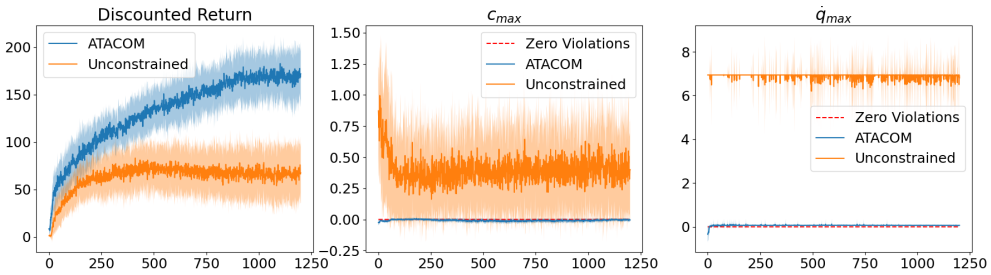
In this section, we will show the learning performance for every skill defined in chapter 3. We also plot the maximal constraint violations and maximal velocity limit violations per training epoch. We will divide the experiments into two groups, one using the planar robot and the other using the IIWA. To start we will evaluate the *PlanarAirHockey* experiments.

4.1.1 *PlanarAirHockey* experiments

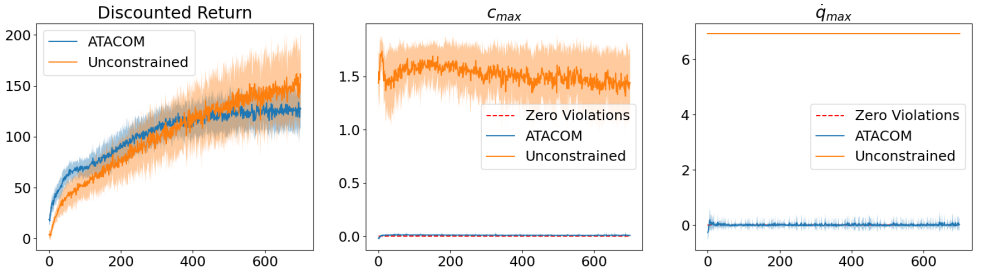
In order to rate the performance of ATACOM, we ran each experiment with and without ATACOM. The results in figure 4.1 show that the addition of ATACOM increases the maximal discounted return in four out of five cases. Additionally, ATACOM converges faster in most cases. The *PlanarPrepare* experiment in figure 4.1c highlights this convergence. At epoch 100 ATACOM has almost converged with a discounted reward of 58 while the unconstrained approach lags behind with a discounted reward of 21. We argue that the faster convergence is a direct result of the constraints shrinking the action space.

The only experiment in which ATACOMs discounted reward is worse is the *PlanarDefend* one in figure 4.1b. Here the unconstrained agent has an easier time catching the puck because it does not have to worry about crashing into the backside of the table. Additionally, the agent can catch higher speed pucks because it ignores the velocity joint limits and thus can move faster. Thus the lower performance of ATACOM is caused by the specific constraints and not the algorithm in general.

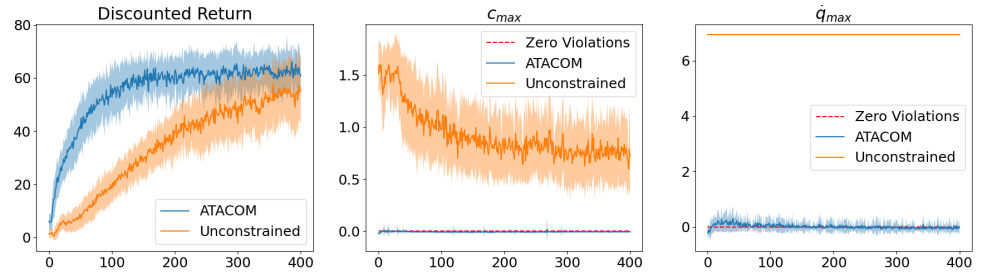
When we take a look at the maximal constraint violation we can see that ATACOM rarely violates any constraints. The maximum violation across all experiments is 0.1. While in theory, no constraint violation is acceptable, in practice the discretization of time always



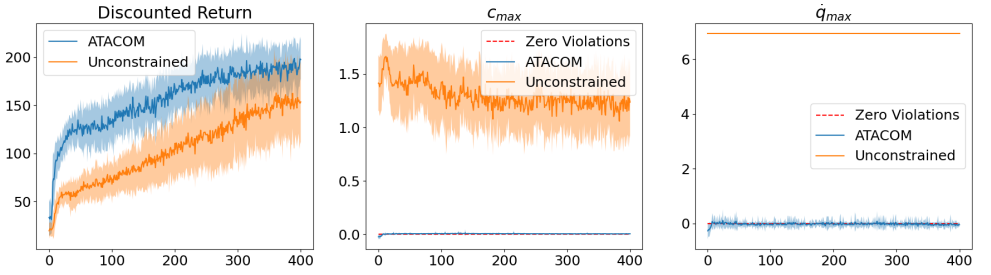
(a) *PlanarHit*



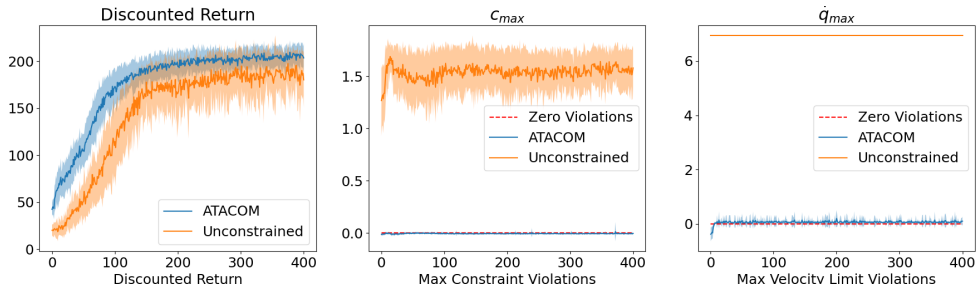
(b) *PlanarDefend*



(c) *PlanarPrepare* with side initialization



(d) *PlanarPrepare* with bottom initialization



(e) *PlanarRepel*

Figure 4.1: Training performance of the *PlanarAirHockey* experiments, the x-axis are epochs

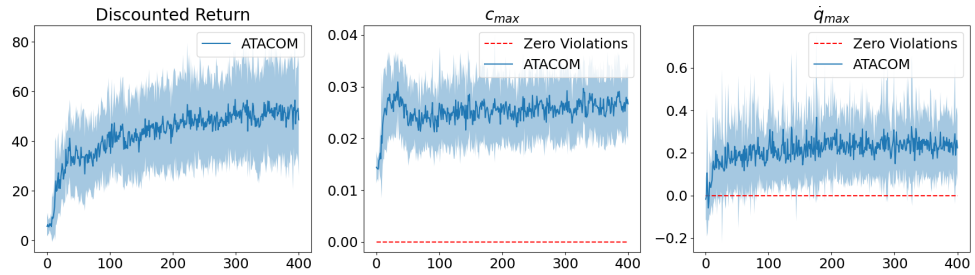
introduces small errors. Because of this discretization, we count these minor violations as acceptable.

The maximal velocity limit violations paint a similar picture. While the maximal violations are a bit bigger, especially at the beginning of training, they go down to an acceptable level towards the end. The maximal velocity limit violation across all experiments is 0.8.

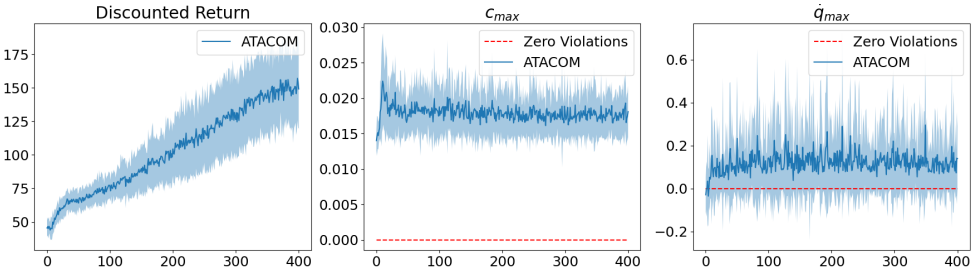
In conclusion, ATACOM was able to enforce the constraints during the entire training. From the big violations of the unconstrained experiment, we can see that these constraints have to be actively guarded and are not just fulfilled passively.

4.1.2 *IwiaAirHockey* experiments

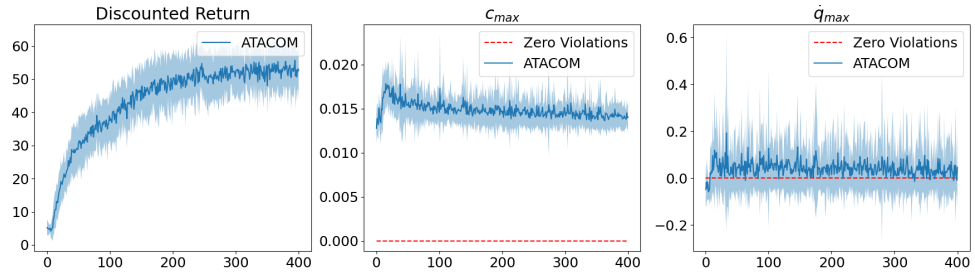
For the IIWA robot, we only ran experiments with ATACOM. A more detailed comparison with unconstrained SAC can be the subject of future work. Furthermore, these results are only preliminary and lack some fine-tuning. That being said the figure 4.2 shows that the IIWA was also able to learn all tasks. However, the performance, in terms of cumulative discounted reward, is worse compared to the *PlanarAirHockey* experiments. The biggest performance loss occurs with the hitting task. Here the maximal discounted reward sinks from 183 to 57. In simpler tasks like the side initialized prepare one, the IIWA performs better. Its discounted reward is 60 and thus only six points behind the *PlanarRepel* one with 66. In general, we are confident that we can increase the performance of the IIWA in the future.



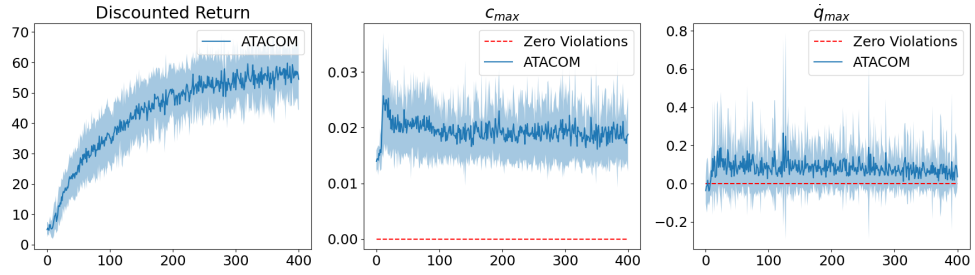
(a) *IiwaHit*



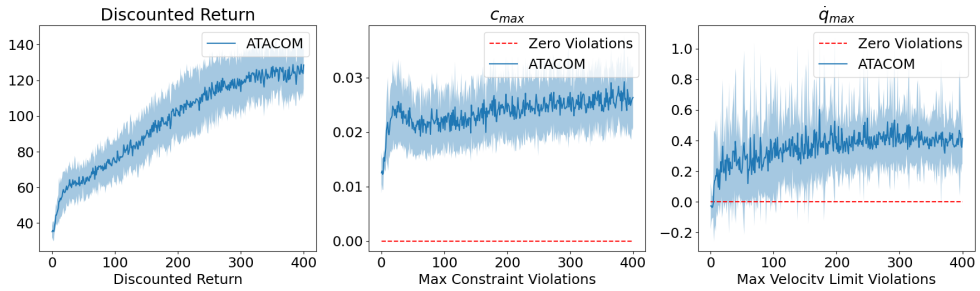
(b) *IiwaDefend*



(c) *IiwaPrepare* with side initialization



(d) *IiwaPrepare* with bottom initialization



(e) *IiwaRepel*

Figure 4.2: Training performance of the *IiwaAirHockey* experiments, the x-axis are epochs

When it comes to maximal constraint violations the gap between the *PlanarAirHockey* and the *IiwaAirHockey* experiments vanishes. Despite the more complex constraints, the constraint violations are on a lower level than the *PlanarAirHockey*. The maximal constrained violation across all experiments is 0.04. That's not even half of the maximal constraint violation in the *PlanarAirHockey* experiment.

The maximal velocity limit violation is 1. This is very similar to the *PlanarAirHockey* experiment, which had 0.8. Thus ATACOM is also able to enforce these more challenging constraints on a complex robot.

4.2 Analysis of Results

In the following, we will analyse the performance of our reward functions. To do that we will evaluate the differences between the learned behaviours of the planar agent and the desired behaviours. Furthermore, we will investigate where and why the learned behaviour fails. We will structure this section by evaluating each task separately, stating with the hitting task.

4.2.1 Hit

In the hitting task, the learned behaviour matches the desired behaviour in the sense that it either tries to shoot a straight shot at the goal or tries to bounce the puck against a sidewall. However it does so in a predictable manner: Whenever the puck spawns to the

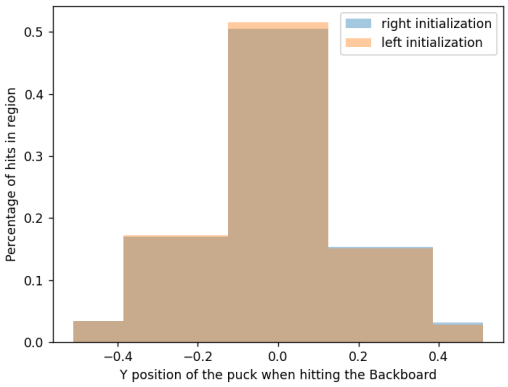


Figure 4.3: Histogramm of the pucks y-position when hitting the backboard in the *Planar-Hit* experiment

left or in the middle of the table, the agent goes behind the puck and shoots it straight to the goal. If the puck spawns on the right the agent will shoot the puck into the right wall to bounce it into the goal. We hypothesise that this happens because of the kinematics of the robot arm. Given the initial configuration towards the right of the table, it is easy to reach behind a puck on the left side but hard to reach behind one on the right side. To support the hypothesis we repeat the experiment with a mirrored agent initialisation.

The histogram in figure 4.3 compares the left and right initialisation. We can see that the end position of the puck follows a very similar normal distribution where 52 per cent of the pucks hit the goal. Thus the agents' initial configuration has no impact on the final puck position. When the puck doesn't hit the goal it hits the backboard to the left or right of the goal, which is also an acceptable outcome of the behaviour. The puck does not hit the backboard in only 2 per cent of the cases, which is a very good performance.

The final puck position is influenced by the spawn location of the puck. Figure 4.4 shows the correlation between the initial position of a puck and its performance. We can see that there are areas in which the performance is significantly lower. The sides of the spawn area are an example, in which the agent struggles to perform. The areas of high performance are divided into two blobs, where one covers a slightly bigger area. This bigger blob is always on the side where the robots' initial configuration is. Thus the initial configuration influences the performance of a given puck. The reason for this becomes obvious when we compare the learned behaviours. As we expected in the hypothesis the

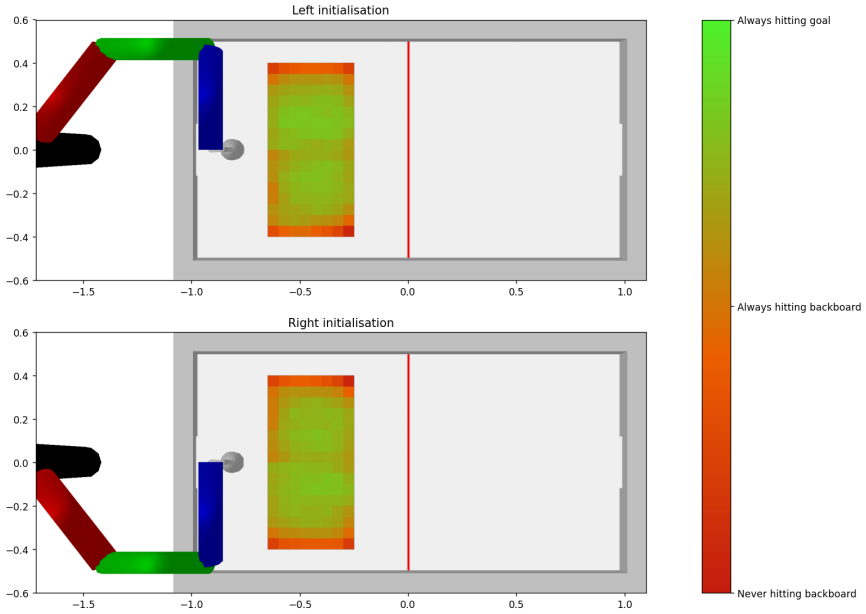


Figure 4.4: Correlation between the pucks initial position and its discounted reward in the *PlanarHit* experiment

left initialisation mirrored the behaviour of the right one.

So far we only looked at the average performance across all experiments. If we consider only consider the best policy the accuracy shoots up to 79 per cent. However, even with this performance, there is a flaw in the behaviour which can be exploited. The behaviour is directly correlated with the puck position. Thus the opponent can know the type of shot he has to defend before it occurs. This is of course undesirable in a competitive setting. We theorize that a more fitting approach would be to train a policy for direct shots and one for bank shots, from which a slightly random higher-level policy chooses. This would make the agent less predictable.

In conclusion, the agent was able to learn the desired behaviour from the reward function. It can perform this behaviour with high accuracy and only fails in hard configurations, like when the puck is at the brim of the agents' reach. However, deploying this agent against a human opponent can be problematic as it is very predictable.

correlation	$\boldsymbol{p}_{puck,x}$	$\boldsymbol{p}_{puck,y}$	$\boldsymbol{v}_{puck,x}$	$\boldsymbol{v}_{puck,y}$	$\boldsymbol{v}_{puck,angular}$
J	0.012	-0.004	0.391	0.024	0.007

Table 4.1: Correlation between the discounted reward and inital puck parameters in *PlanarDefend*

4.2.2 Defend

For the defending task, the learned behaviour tries to stop the puck by reflecting it into the sidewall, where it ricochets between the walls. While it also tries to catch the puck in certain situations, this often fails, resulting in a low 43 per cent accuracy. The learned behaviour matches the desired behaviour, making our reward function sufficiently informative. Despite the right behaviour, the performance is bad because the actions need to be very precise to succeed. A small error in the behaviour can already cause a failure in this task.

When the agent fails to stop the puck it usually slowly floats back to the opponent, which is undesired because we gift control of the puck to the opponent. With these consequences, an accuracy of 43 per cent is low, however, we anticipated the task to be hard.

To investigate where we lose performance, we look at the correlation between the discounted reward and the initial puck parameters in table 4.1. It shows a strong correlation between the pucks' x-velocity. All other parameters have a significantly weaker correlation to the discounted reward. Thus restricting the velocity should increase the performance.

In figure 4.5 the puck velocity was binned into three options. For each bin, a histogram of the pucks' end position is depicted. In the top row of the histogram are all the pucks which hit the termination line. These are the failures that would coast back to the opponent. Therefore this row has a separate scale. The histograms show that the performance rises significantly with a slower puck. With an accuracy of 70 per cent, the defence task performs well under the condition that the puck is slow enough.

To get a greater insight into the performance we looked into the performance correlation with the remaining initial parameters. For further analysis, the puck velocity will be set to slow. The figure 4.6 shows that the performance is high when the puck spawns in the middle is shot roughly straight or when the shot spawns on the side is shot directly into the wall. From this data, we assume that the agent has trouble defending pucks that travel

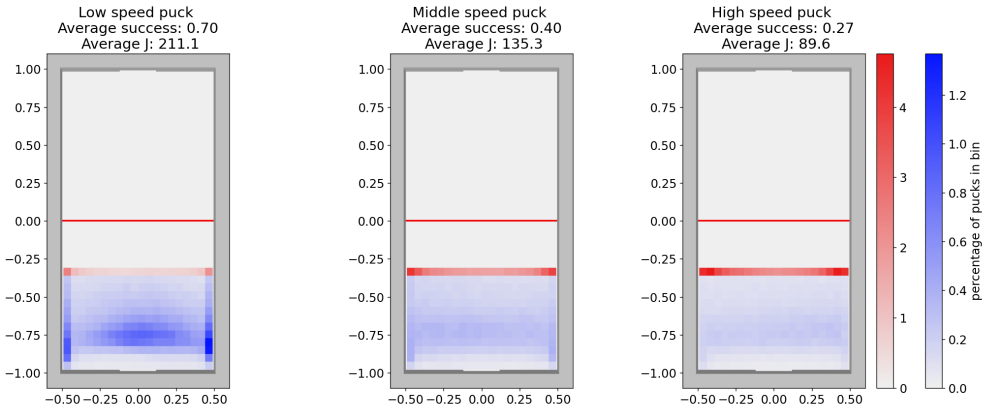


Figure 4.5: Histogramm of the pucks end position in the *PlanarDefend* experiment, binned into three velocity options

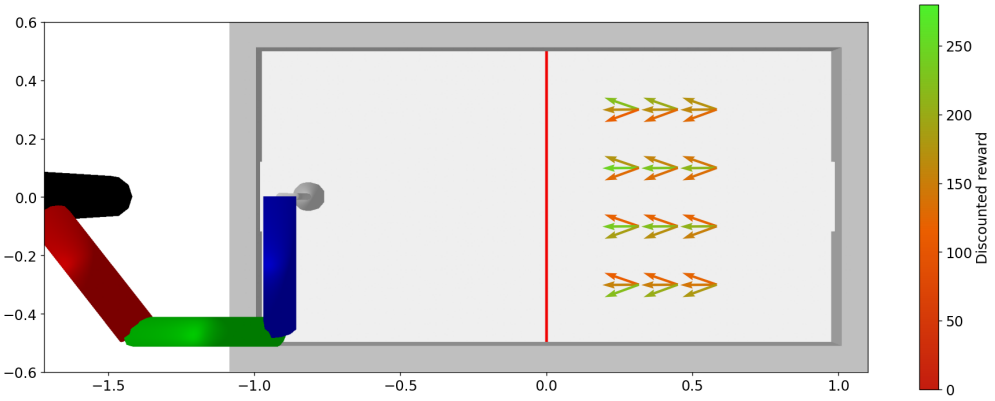


Figure 4.6: Correlation between the discounted reward and the initial puck velocity and angle in the *PlanarDefend* experiment. The velocity is fixed to be slow

towards the corners of the table. Given this insight, it makes sense to only try and defend a puck when it is on a slow trajectory toward the goal.

4.2.3 Prepare

In the prepare task, we have to evaluate two different initializations, which have separate reward functions. To start we will analyse the behaviour of the side initialization.

Side Initialization

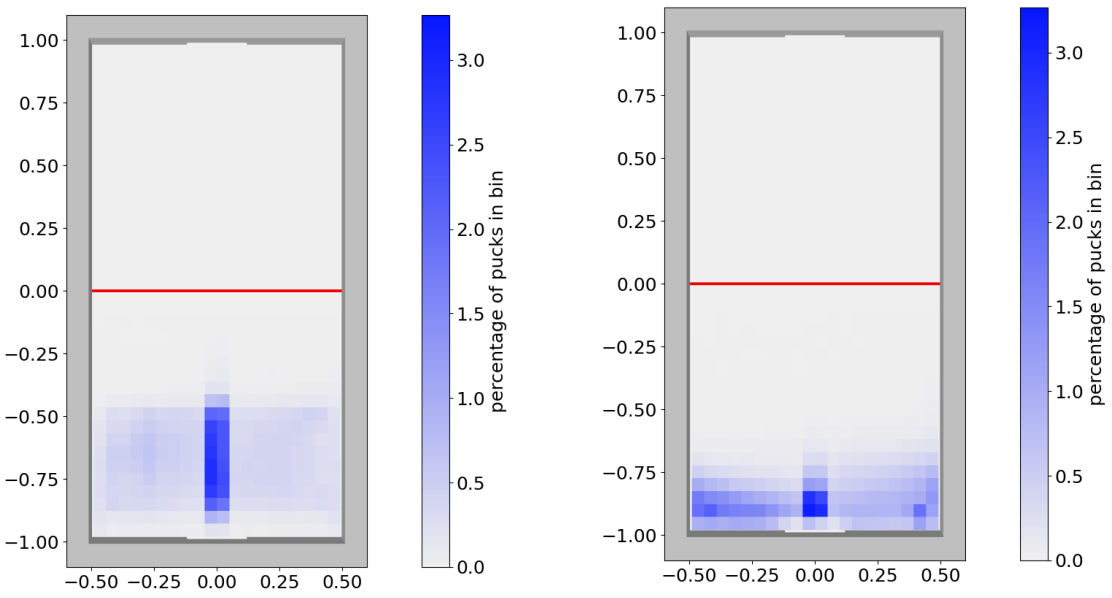
In terms of behaviours, the learned one matches the desired one very closely. The agent shoots the puck orthogonal to the sidewall and then moves out of the trajectory to not intercept the puck. Thus our reward function was able to encode the desired behaviour.

The histogram in figure 4.7a confirms that most pucks' end positions are at the termination line. While this is not the true final position of the pucks, it already shows that the agent never loses control of the puck by shooting it towards the opponent. If we can reliably stay in control of the puck, the prepare behaviour can be repeated until the final puck position is acceptable. The performance is pretty good with an average success rate of 65 per cent. The best policy even reaches 78 per cent.

To investigate where the performance is lost we will consider figure 4.8a. It shows the correlation between the initial puck position and the discounted reward. The figure indicated that the behaviour struggles when the pucks' initial position is close to the wall. This makes sense because we rely on the puck bouncing off the wall, which is not possible if it is already touching it. Additionally, the time the agent has to move out of the way is dependent on the distance between the puck and the wall. Thus it has to act quicker and closer to the constraints when the puck is close to the wall, making the behaviour hard to learn.

Bottom Initialization

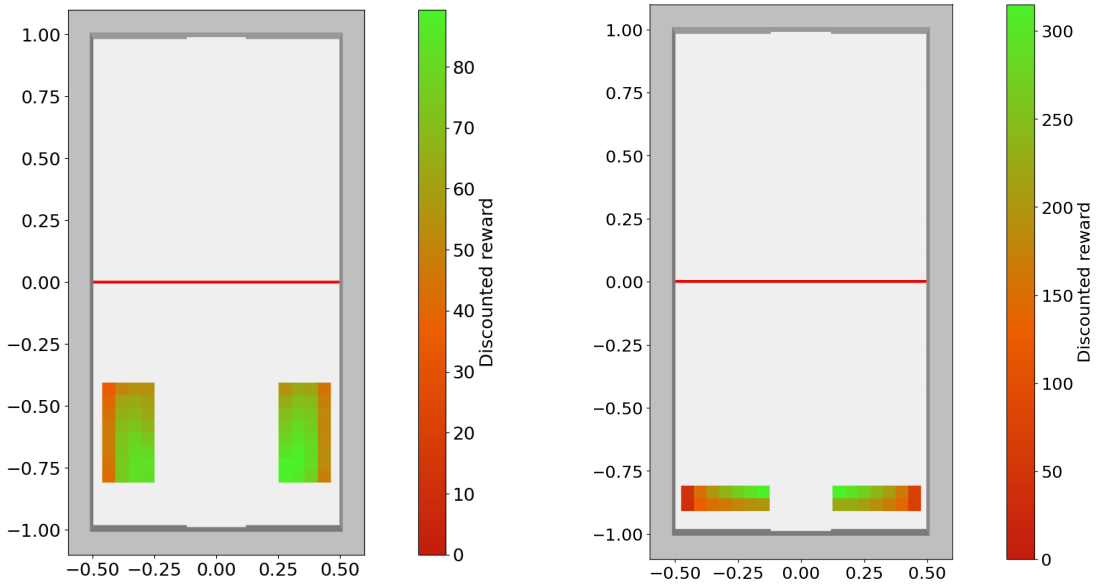
With the bottom initialization, the learned behaviour is roughly similar to the desired one. The puck is shot against the sidewall with a slight upwards momentum. With that slight upwards momentum the puck slowly moves upwards while ricocheting between the sidewalls. However, it does happen that this upwards momentum is missing. Then the



(a) Side Initialisation

(b) Bottom Initialisation

Figure 4.7: Histogramm of the latest puck position in the *PlanarPrepare* experiments



(a) Side Initialisation

(b) Bottom Initialisation

Figure 4.8: Correlation of the inital puck position and the discounted reward in the *Planar-Prepare* experiments

puck is shot orthogonal to the sidewall. This mistake explains the low success rate of 39 per cent, with the best policy reaching 60 per cent.

The difference in behaviour hints at a mistake in the reward function. We assume that more emphasis on the upwards momentum is required, which we will test in future work.

The histogram in figure 4.7b shows that the final puck position is all over the place. While there is a definite hotspot on the termination line, the surrounding areas are also present. It is important to remember that an episode ends long before the puck lost all its momentum. Thus the points in the histogram are not the true final puck positions. It is fair to assume that the momentum of the puck will carry it a bit higher than the positions depicted in the histogram. With that in mind, the final positions should be usable. The only unrecoverable positions are the ones that stop while touching a wall. The histogram shows that the puck never comes close to the opponents' area. Thus we can safely repeat the behaviour until a desired final puck position is achieved.

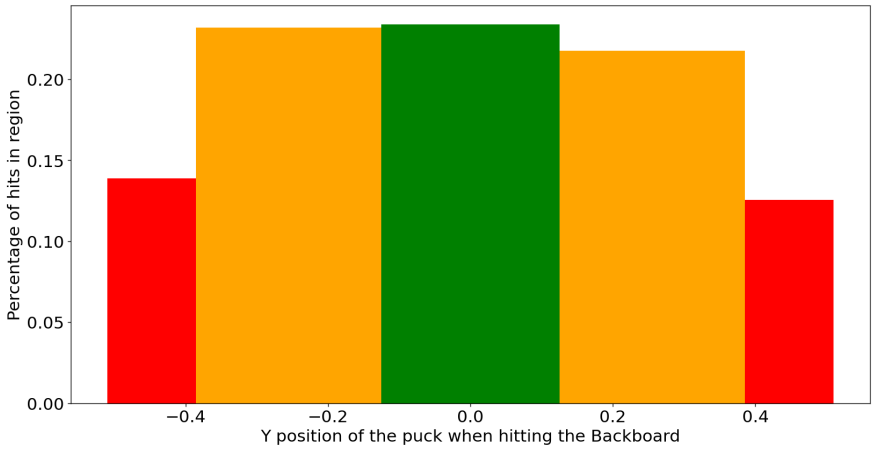


Figure 4.9: Histogramm of the pucks y-position when hitting the backboard in the *Planar-Repel* experiment. The green bar indicates that the goal has been hit.

The correlation plot in figure 4.8b is similar to the side initialisation. The performance drops when being close to the sidewall for the same reasons previously mentioned. Additionally, the performance drops when getting close to the bottom wall, which makes sense because getting behind the puck to give it upwards momentum becomes hard or impossible.

4.2.4 Repel

In the repel task the learned behaviour matches the pucks' y-position and gives it a little hit as soon as the puck is close to the mallet. It does not build up momentum to hit the puck with maximum velocity. This makes sense because the latter behaviour is far more error-prone and therefore not viable, especially when the incoming puck is already fast. Thus the reward function was informative enough.

The histogram in figure 4.9 shows that the puck hits the backboard 95 per cent of the time. In 24 per cent of the cases, the puck even scores a goal. The yellow bin to the left of the goal sees a bit more pucks than the one on the right. However, the difference is so small that this is not concerning.

As the overall performance is fully satisfactory, we did not further investigate performance loss. This is however not surprising as it is a simple behaviour. In the future, we would consider making this behaviour more difficult. This could be done by changing the focus to scoring a goal.

4.3 Putting the Pieces Together

Now that we achieve good performances with the planar robot in every task, we can build a higher-level framework to combine them. We note that this high-level policy is a hardcoded prototype. It is a simple state automaton that decides which of the actions to take, based on the pucks' position and velocity. The goal of this policy is not to achieve the best possible performance. Instead, we want to evaluate the interactions between the different tasks in a continuous game of air hockey.

To do that we let two agents running our high-level policy play against each other. The behaviour, depicted in this video [1], can be boiled down to the following. After the initial hitting of the puck, the opponent has to decide whether to repel or defend. If he chooses to repel the 2 agents engage in a repel battle until the puck reaches such a high momentum that one of the agents misses. On the other hand, if the agent defends the puck, he waits until the puck has lost all its momentum. Depending on the final position of the puck, the agent then hits the puck to the opponent or prepares the puck for a subsequent hitting task.

There are however some systematic problems that occur within the interaction between tasks. One problem is that waiting for the puck to stop after a defence or preparation takes very long. Additionally, due to the physics of the air hockey table, the most momentum is lost when the puck bounces off the side of the table. Thus the puck ends up stopping on the edge of the table quite often. This is problematic because we are not able to recover a puck that is touching the side of a table. A solution could be to redefine the hit task such that the puck does not have to be completely still. Instead hitting could occur after the puck reaches a slow enough momentum.

Another problem is that the area which the bottom initialised prepare task covers is simply too small. In a real game, the puck rarely stops in this area. This is however no apparent solution to this problem. If we increase the area the current desired behaviour will not be sufficient to solve the task. Thus we will address this problem in future work.

[1] https://youtu.be/kxVH4SOz8qY

On the other hand, the interactions between hit and defend or repel already work quite well. Additionally, the defence and hit combination looks great if the puck happens to stop in a good location.

5 Conclusion

This work shows that ATACOM can solve more complex tasks than the ones proposed in the original paper. To show the adaptability of the approach, we introduced the tasks hit, defend, prepare and repel. For each of these tasks, we were able to train the planar robot which resulted in promising results. These results confirm that our designed reward functions were able to encode the desired behaviour to a high degree and that the usage of ATACOM did not reduce the training performance.

Additionally, we trained the IIWA robot on the tasks. With that experiment, we showed that ATACOM could handle a more challenging set of constraints as well as a complex robot. However, we were not able to get the performance up to par with the planar robot.

Above all, there were no major constraint violations across all experiments. Thus ATACOM was able to ensure safety throughout the training. These results confirm our original hypothesis. Additionally, this performance paves the road for transferring our results onto a real air hockey table with human opponents. Furthermore, we were able to combine the tasks into a prototype of a general air hockey agent, which can play against itself. Here we noticed some limitations of the current definition of our tasks.

In the future, we want to eliminate these limitations to enable smooth interaction between all tasks. We assume that just changing the hitting task from a still puck to a slightly moving puck could increase the performance significantly.

Another interesting topic is the high-level policy. While at the moment we just hardcoded a simple logic, it is possible to improve the performance by fine-tuning. A further promising approach would be to use reinforcement learning to train this policy in an adversarial manner.

One more major point is to improve the performance of the IIWA robot to match the planar robots scores. If that is the case an investigation into the comparison between unconstrained performance and the ATACOM performance on the IIWA platform should also be interesting.

Bibliography

[1] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu. Safe reinforcement learning via shielding. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018. URL https://ojs.aaai.org/index.php/AAAI/article/view/11797.

[2] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6): 26–38, Nov 2017. ISSN 1053-5888. doi: 10.1109/msp.2017.2743240. URL http://dx.doi.org/10.1109/MSP.2017.2743240.

[3] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause. Safe model-based reinforcement learning with stability guarantees. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/766ebcd59621e305170616ba3d3dac32-Paper.pdf.

[4] Y. Chow, O. Nachum, E. A. Duéñez-Guzmán, and M. Ghavamzadeh. A lyapunov-based approach to safe reinforcement learning. *CoRR*, abs/1805.07708, 2018. URL http://arxiv.org/abs/1805.07708.

[5] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2019.

[6] G. Dalal, K. Dvijotham, M. Vecerik, T. Hester, C. Paduraru, and Y. Tassa. Safe exploration in continuous action spaces. 01 2018.

[7] C. D'Eramo, D. Tateo, A. Bonarini, M. Restelli, and J. Peters. Mushroomrl: Simplifying reinforcement learning research. *Journal of Machine Learning Research*, 22(131): 1–5, 2021. URL http://jmlr.org/papers/v22/18-056.html.

[8] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. *CoRR*, abs/1604.06778, 2016. URL http://arxiv.org/abs/1604.06778.

[9] J. García and F. Fernández. Safe exploration of state and action spaces in reinforcement learning. *J. Artif. Int. Res.*, 45(1):515–564, sep 2012. ISSN 1076-9757.

[10] J. García and F. Fernández. A comprehensive survey on safe reinforcement learning. *J. Mach. Learn. Res.*, 16(1):1437–1480, jan 2015. ISSN 1532-4435.

[11] A. Gasparetto, P. Boscariol, A. Lanzutti, and R. Vidoni. Trajectory planning in robotics. *Mathematics in Computer Science*, 6(3):269–279, 2012.

[12] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine. Soft actor-critic algorithms and applications. *CoRR*, abs/1812.05905, 2018. URL http://arxiv.org/abs/1812.05905.

[13] A. Hans, D. Schneegass, A. Schäfer, and S. Udluft. Safe exploration for reinforcement learning. pages 143–148, 01 2008.

[14] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *CoRR*, cs.AI/9605103, 1996. URL https://arxiv.org/abs/cs/9605103.

[15] S. Krishnan, A. Garg, R. Liaw, L. Miller, F. T. Pokorny, and K. Goldberg. HIRL: hierarchical inverse reinforcement learning for long-horizon tasks with delayed rewards. *CoRR*, abs/1604.06508, 2016. URL http://arxiv.org/abs/1604.06508.

[16] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In Y. Bengio and Y. LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1509.02971.

[17] P. Liu, T. D., H. Bou-Ammar, and J. Peters. Efficient and reactive planning for high speed robot air hockey. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021. URL https://www.ias.informatik.tu-darmstadt.de/uploads/Team/PuzeLiu/IROS_2021_Air_Hockey.pdf.

[18] P. Liu, D. Tateo, H. B. Ammar, and J. Peters. Robot reinforcement learning on the constraint manifold. In *5th Annual Conference on Robot Learning*, 2021. URL https://openreview.net/forum?id=zwo1-MdMl1P.

[19] P. Liu, D. Tateo, H. Bou-Ammar, and J. Peters. Efficient and reactive planning for high speed robot air hockey. *CoRR*, abs/2107.06140, 2021. URL https://arxiv.org/abs/2107.06140.

[20] T. SÜD. Testing of robots and robot controllers, 2018. URL https://www.tuvsud.com/en/industries/manufacturing/machinery-and-robotics/robotic-safety.