POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Tesi di Laurea Magistrale in
Computer Science and Engineering
Ingegneria Informatica

Robotic Arm Control via Curriculum Deep Reinforcement Learning

Author:
Amirhossein Zhalehmehrabi

Student ID:
10832946

Advisor:
Prof. Marcello Restelli

Co-advisor:
Dr. Amarildo Likmeta

Academic Year:
2023-2024

Dedicated to my family.

Abstract

Reinforcement learning (RL), especially deep reinforcement learning (DRL), has achieved remarkable success in complex domains like game playing, robotics, and autonomous driving. Curriculum learning (CL) in RL, inspired by educational curricula, involves training agents on tasks of increasing difficulty to enhance learning efficiency and generalization. This thesis addresses the challenges of RL by applying both manual and automatic CL to robotic control in air hockey.

Air hockey is modeled as a Markov Decision Process (MDP), using joint positions and velocities for state and action spaces. Manual CL begins with a "Defend" task, where task complexity is incrementally adjusted, followed by a "Counter-Attack" task to develop offensive strategies. The automatic CL approach employs a teacher-student architecture for the "Hit" task, where the teacher assigns mini-tasks to guide the student's learning.

Empirical results demonstrate that manual CL effectively improves skill refinement and task adaptation, while automatic CL shows potential for adaptive training strategies. This research advances RL techniques for robotic control, offering insights into practical applications in dynamic environments.

Keywords: Reinforcement Learning, Robotics, Curriculum Learning

Abstract in lingua italiana

Lapprendimento per rinforzo (RL), in particolare l'apprendimento per rinforzo profondo (DRL), ha ottenuto un notevole successo in settori complessi come i giochi, la robotica e la guida autonoma. L'apprendimento del curriculum (CL) in RL, ispirato ai curricula educativi, prevede la formazione di agenti su compiti di difficoltà crescente per migliorare l'efficienza dell'apprendimento e la generalizzazione. Questa tesi affronta le sfide della RL applicando la CL sia manuale che automatica al controllo robotico nell'air hockey.

L'air hockey è modellato come un processo decisionale di Markov (MDP), utilizzando posizioni e velocità congiunte per gli spazi di stato e di azione. Il CL manuale inizia con un'attività di "Difesa", in cui la complessità dell'attività viene modificata in modo incrementale, seguita da un'attività di "Contrattacco" per sviluppare strategie offensive. L'approccio CL automatico utilizza un'architettura insegnante-studente per il compito "Hit", in cui l'insegnante assegna mini-compiti per guidare l'apprendimento dello studente.

I risultati empirici dimostrano che il CL manuale migliora efficacemente il perfezionamento delle competenze e ladattamento al compito, mentre il CL automatico mostra il potenziale per strategie di formazione adattative. Questa ricerca fa avanzare le tecniche RL per il controllo robotico, offrendo approfondimenti su applicazioni pratiche in ambienti dinamici.

Parole chiave: Apprendimento per rinforzo, robotica, apprendimento curriculare

Contents

1 | Introduction

In recent years, the rapid advancements in technology have propelled the field of machine learning to the forefront of innovation, revolutionizing various domains such as healthcare, finance, and autonomous systems. Machine learning, a subset of artificial intelligence, encompasses a spectrum of techniques that enable systems to learn from data and improve performance on specific tasks without being explicitly programmed. Broadly categorized into supervised, unsupervised, and reinforcement learning, machine learning algorithms exhibit diverse capabilities in solving a wide array of problems.

Reinforcement learning (RL) stands out as a particularly intriguing branch of machine learning, distinguished by its focus on sequential decision-making and interaction with an environment to achieve a defined goal. Unlike supervised learning, where algorithms learn from labeled data, and unsupervised learning, which explores patterns in unlabeled data, RL agents learn through trial and error, receiving feedback in the form of rewards or penalties based on their actions. This iterative process of exploration and exploitation enables RL agents to devise optimal strategies to maximize cumulative rewards over time.

Deep reinforcement learning (DRL) represents a fusion of reinforcement learning with deep neural networks, leveraging the expressive power of neural networks to handle high-dimensional input spaces and complex decision-making tasks. By approximating value functions or policy functions using deep neural networks, DRL algorithms have achieved remarkable success in challenging domains such as game playing, robotics, and autonomous driving.

Moreover, within the realm of RL, curriculum learning has emerged as a promising strategy to facilitate the training of RL agents. Inspired by the concept of curriculum in education, where learning tasks are organized in a progressive manner from simple to complex, curriculum learning in RL involves designing a sequence of tasks or environments with increasing levels of difficulty. By initially exposing the agent to simpler tasks and gradually increasing the complexity, curriculum learning aims to guide the agent's learning process, facilitating faster convergence and improved generalization to more challenging scenarios.

However, despite the remarkable progress in RL, DRL and Curriculum Learning, numerous challenges persist, hindering their widespread adoption and application in real-world scenarios. These challenges encompass issues related to sample inefficiency, exploration-exploitation trade-offs, stability, generalization, and safety concerns in complex and uncertain environments.

This thesis aims to delve into the nuances of RL and DRL, exploring their underlying principles, methodologies, recent advancements, and unresolved challenges. Specifically, this study focuses on addressing the challenges present in controlling a robot to play air hockey, a complex and dynamic task that demands precise control and rapid decision-making. By critically examining the existing literature on RL and DRL in the context of robotic air hockey and conducting empirical investigations tailored to this application, this research employs both manual and automatic curriculum learning approaches. These approaches are designed to progressively train the robot, enhancing its performance incrementally. This study endeavors to contribute novel insights and methodologies to advance the state-of-the-art in RL and DRL for robotic control in air hockey.

Motivation

Training a reinforcement learning (RL) agent presents a formidable challenge, contingent upon various factors such as the task's complexity, reward structure, and numerous other considerations. Among these challenges, a prominent obstacle arises when dealing with complex scenarios where employing sparse rewards proves impractical. Sparse rewards denote situations where the agent receives feedback only when achieving a specific goal state perfectly, typically indicated by a reward of nonzero value, while all other states yield a reward of zero. However, in intricate environments, the sheer complexity of the task renders it exceedingly difficult for the agent to discern the optimal actions solely from such sporadic feedback. Consequently, the agent encounters considerable difficulty in navigating its exploration process toward the goal state, impeding effective learning and convergence. It's worth noting that sparse rewards often provide the most accurate depiction of the task to the agent, preserving a high level of autonomy. Indeed, any alternative approach risks constraining the agent's autonomy by oversimplifying or distorting the task's objectives, potentially hindering its ability to learn and adapt to real-world challenges effectively.

Goal

In this thesis, the objective is to address the specific challenges encountered in controlling a robot to play air hockey, particularly focusing on enhancing the performance

of defense and counter-attack strategies. During the air hockey challenge, we employed manual curriculum learning to significantly improve these agents, which were crucial for our team's overall success. Additionally, recognizing the hit agent as the weakest link in our team, we explored an automatic curriculum learning approach post-challenge to improve its performance. This thesis demonstrates how employing curriculum learning, both manual and automatic, was essential in overcoming the challenges of training RL agents for complex tasks, leading to better adaptability and performance in the air hockey scenario.

Contributions

The thesis makes two significant contributions to the field of RL. Firstly, it introduces a manually designed curriculum learning framework tailored for training RL agents to excel in complex tasks, exemplified by the challenging domain of air hockey. Through participation in the Air Hockey Challenge[1], this approach aimed to enhance the agent's performance by systematically exposing it to tasks of increasing difficulty. However, upon encountering limitations inherent in manual curriculum design, the thesis proposes a novel methodology leveraging automatic curriculum learning. This innovative approach streamlines the process of task progression, dynamically adjusting the curriculum based on the agent's learning progress and performance. By transitioning from traditional approaches to automatic curriculum learning for other tasks, this thesis introduces an innovative approach aimed at improving the training of RL agents. While the automatic approach represents a novel direction, it has not yet surpassed the effectiveness of traditional reward function engineering methods. Nevertheless, this exploration provides valuable insights and lays the groundwork for future advancements in the field.

Thesis Structure

This thesis is organized into six chapters, each addressing different aspects of the research on manual and automatic curriculum learning for robot air hockey. Chapter 1 introduces the research problem, objectives, and significance of the study. Chapter 2 covers the preliminaries, providing foundational concepts and theoretical background necessary for understanding the subsequent chapters. In Chapter 3, we review related works, situating our research within the broader context of existing literature. Chapter 4 describes the robot air hockey challenge, detailing the task specifics. Chapter 5 outlines the methodology, including the proposed approaches and algorithms for manual and automatic curriculum learning. Finally, Chapter 6 presents the results, followed by a comprehensive discussion of the findings, implications, and future research directions.

2 | Preliminaries

2.1. Reinforcement Learning

Reinforcement learning[52] is an area of machine learning concerned with how an agent ought to take actions in a dynamic environment to maximize some notion of long term performance, commonly represented as the expected cumulative sum of a (possibly discounted) scalar reward signal.

RL is distinct from supervised learning in that supervised learning relies on training data that includes the correct answers, allowing the model to be trained directly on those answers. In contrast, reinforcement learning does not have access to the correct answers but instead relies on a reinforcement agent to determine the appropriate actions to perform a given task. Without a training dataset, reinforcement learning must rely on its own experiences to learn and improve using a trial-and-error approach.

RL employs algorithms that learn from outcomes and make decisions on which actions to take. Feedback is received after each action, helping the algorithm assess the correctness, neutrality, or incorrectness of its choices. This makes RL particularly useful for automated systems that need to make numerous small decisions without human guidance.

Overall, RL is an autonomous and self-teaching system that learns through trial and error. By performing actions and aiming to maximize rewards, it learns by doing in order to achieve the best possible outcomes.

2.2. Markov Decision Process

A Markov Decision Process (MDP) is a mathematical framework used to model decision-making problems where an agent interacts with an environment over discrete time steps.

Definition 2.1 (Markov Decision Process)**.** *an MDP[52] is defined by a 6-tuple* $(S, A, P, R, \mu, \gamma)$

where:

- S *is the set of states.*
- A *is the set of actions available to the agent.*
- $P : S \times A \times S \to [0, 1]$ *is the transition probability function.* $P(s'|s, a)$ *denotes the probability of transitioning to state* s' *from state* s *under action* a*.*
- $R : S \times A \to \mathbb{R}$ *is the reward function.* $R(s, a)$ *denotes the immediate reward obtained when taking action* a *in state* s*.*
- $\mu : S \to [0, 1]$ *is the initial state distribution, representing the probability of starting in each state.*
- $\gamma \in [0, 1]$ *is the discount factor, which determines the importance of future rewards relative to immediate rewards. A discount factor close to 1 indicates a long-term focus, while a discount factor close to 0 indicates a short-term focus.*

MDP has Markov property which is crucial for RL. It ensures that the current state contains all the information needed to make decisions about future actions. This property states that the future state and reward depends only on the current state and action, not on the sequence of events that led to the current state.

Definition 2.2 (History)**.** *A history* h *is defined as a sequence of state-action-reward tuples* $(s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_n)$ *that represents the sequence of events an agent experiences in an environment.*

Definition 2.3 (Markov Property)**.** *A function* f *has the Markov property[52] if and only if any two histories* h *and* h_0 *that are mapped by* f *to the same state* $(f(h) = f(h_0))$ *also have the same probabilities for their next state,*

$$f(h) = f(h_0) \Rightarrow Pr\{S_{t+1} = s|H_t = h, A_t = a\} = Pr\{S_{t+1} = s|H_t = h_0, A_t = a\} \tag{2.1}$$

for all $s \in \mathcal{S}$ *and* $a \in \mathcal{A}$*.*

In other words, A stochastic process $X(t)$ *satisfies the Markov property if, for any time* t *and any sequence of times* $t_0 < t_1 < \ldots < t_n$*, the conditional probability of the future state* $X(t)$ *given the past states* $X(t_0), X(t_1), \ldots, X(t_n)$ *depends only on the present state* $X(t_n)$*, not on the entire history* $X(t_0), X(t_1), \ldots, X(t_{n-1})$*. Mathematically, the Markov property can be expressed as:*

$$\begin{aligned} P(X(t_{n+1}) = x|X(t_n) = x_n, X(t_{n-1}) = x_{n-1}, \ldots, X(t_0) = x_0) = \\ P(X(t_{n+1}) = x|X(t_n) = x_n) \end{aligned} \tag{2.2}$$

An MDP evolves over time through a sequence of actions taken by the agent. At each time step t, the agent selects an action a_t based on its current state s_t. The state and action space can be either finite or infinite, discrete or continuous. The environment then transitions to a new state s_{t+1} according to the transition probability function P, and the agent receives a reward $R(s_t, a_t)$. The goal of the agent is to find a policy, denoted as $\pi(a|s) = \Pr\{A_t = a|S_t = s\}$, that maximizes the expected cumulative discounted reward G_t over time.

Definition 2.4 (Expected Discounted Cumulative Reward)**.** *The expected discounted cumulative reward[52], also known as the expected discounted return, is a concept in reinforcement learning that quantifies the total expected reward an agent can accumulate over time while taking into account the temporal discounting of future rewards. Mathematically, it is defined as follows:*

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.3}$$

The discount factor γ is a parameter used to determine the importance of future rewards relative to immediate rewards. It's a value between 0 and 1, where:

- $\gamma = 0$ indicates that the agent only cares about immediate rewards.
- $\gamma = 1$ indicates that the agent equally values immediate and future rewards.

Discount factor also plays a crucial role in encouraging convergence and addressing the trade-off between short-term gains and long-term objectives.

MDPs are often compared to Markov chains, which only model the randomness without the decision-making aspect. By incorporating actions and rewards, MDPs enable the development of optimal policies for the agent, guiding it towards the most beneficial actions in the long run. They are widely used in fields like artificial intelligence, robotics, economics, and game theory.

2.3. Policies

In MDPs, the agent interacts with the environment by selecting actions based on the current state and the policy in place. The goal of the agent is to maximize expected cumulative discounted reward as defined in equation 2.3. A policy encapsulates the decision-making process of the agent, determining how it behaves in different situations, thus it is a strategy or rule that governs the agent's behavior in an environment. Formally, a policy is a mapping from states to a distribution over actions.

Definition 2.5 (Policy). *Let S be the set of states, A be the set of actions in an MDP and t the time step. A policy[52] is a mapping from states to probabilities of selecting each possible action. If the agent is following policy π at time t, then $\pi(a|s)$ is the probability that $A_t = a$ if $S_t = s$.*

$$\pi(a|s) = Pr\{A_t = a|S_t = s\} \tag{2.4}$$

The quality of a policy is typically evaluated based on the expected cumulative reward obtained when following that policy. An optimal policy is one that maximizes this expected cumulative reward.

2.3.1. Deterministic Policy

Deterministic policy is a special case of a policy where the probability distribution has the variance of 0 meaning that for each state, there is a single, specific action to be taken. Mathematically, $\pi(s)$ yields a single action.

Definition 2.6 (Deterministic Policy). *A deterministic policy can be defined as:*

$$\pi_d : S \to A \tag{2.5}$$

where:

- *S is the set of states.*
- *A is the set of actions.*
- *π_d is a deterministic policy mapping each state $s \in S$ into a single action $a = \pi(s)$.*

2.4. Value Functions

Value functions are essential for evaluating the quality of states or state-action pairs and for guiding the agent towards making decisions that lead to higher cumulative rewards. There are two primary types of value functions in RL: state value functions and action value functions.

2.4.1. State Value Function

State value functions, which are denoted as V, estimate the expected cumulative reward that an agent can obtain starting from a particular state and then following a certain policy.

Definition 2.7 (State Value Function (V))**.** *Let $M := (S, A, P, R, \mu, \gamma)$ be a MDP, and π a policy. The state value function $V^\pi(s)$ under policy π is defined as the expected return starting from state s and following policy π:*

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \middle| S_0 = s \right] \tag{2.6}$$

where:

- R_{t+1} *is the reward obtained at time step* $t+1$.
- γ *is the discount factor*
- S_0 *is the starting state.*
- E_π *indicates the expected value over the trajectories generated by applying policy* π.

2.4.2. Action Value Function

Action value functions, which are denoted as Q, estimate the expected cumulative reward that an agent can obtain starting from a particular state, taking a specific action, and then following a certain policy.

Definition 2.8 (Action Value Function (Q))**.** *Let $M := (S, A, P, R, \mu, \gamma)$ be a MDP, and π a policy. The action value function $Q^\pi(s, a)$ under policy π is defined as the expected return starting from state s, taking action a, and then following policy π:*

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \middle| S_0 = s, A_0 = a \right] \tag{2.7}$$

Where:

- S_0 *is the starting state.*
- A_0 *is the action taken at time step* 0.
- E_π *indicates the expected value over the trajectories generated by applying policy* π.

2.5. Optimality

In Reinforcement Learning (RL), optimality refers to achieving the best possible performance in terms of maximizing cumulative rewards while interacting with an environment. There are two main notions of optimality in RL: value optimality and policy optimality.

2.5.1. Value Optimality

Value optimality refers to the concept of finding an optimal policy that maximizes the expected cumulative reward over time. This is typically achieved by estimating the value function, which represents the expected cumulative reward that an agent can expect to receive while following a particular policy from a given state.

Definition 2.9. *Let S_0 be the state that the interactions started from, G_t be the cumulative reward obtained starting from state S_0 and π be any possible policy. Then the optimal value function, denoted as $V^*(s)$ is defined as:*

$$V^*(s) = \max_{\pi} \mathbb{E}[G_t | S_t = S_0, \pi] \tag{2.8}$$

Where:

- *S_t represents the state at time step t.*
- *E_π indicates the expected value.*

2.5.2. Policy Optimality

Policy optimality refers to the state of having found the best policy among the set of all possible policies. An optimal policy is the one that maximizes the cumulative reward over time.

Definition 2.10. *Let G_t be the cumulative reward and π be any possible policy. Then the optimal policy, denoted as π^* is defined as:*

$$\pi^* = \arg\max_{\pi} \mathbb{E}[G_t | \pi] \tag{2.9}$$

where E_π indicates the expected value.

Value optimality focuses on finding the optimal value function, while policy optimality

aims to directly identify the best policy without explicitly computing the value function.

2.6. Bellman equations

Bellman equations are fundamental recursive relationships that describe the value of states (or state-action pairs) in terms of the values of their successor states (or successor state-action pairs) under a given policy. Bellman equations are used extensively in RL algorithms to compute and update value functions efficiently. These equations provide the foundation for various RL algorithms, enabling agents to learn optimal policies through iterative updates of value functions.

2.6.1. Bellman Expectation Equation for State-Value Functions

The Bellman expectation equation for state-value functions expresses the value of a state s under a policy π as the expected immediate reward plus the discounted value of the next state under the same policy.

Definition 2.11 (Bellman Expectation Equation for State-Value Functions)**.** *Let $M := (S, A, P, R, \mu, \gamma)$ be a MDP, and π a policy. Bellman Expectation Equation for State-Value Functions[52] is defined as:*

$$V_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s] \tag{2.10}$$

Where E_π indicates the expected value over the trajectorioes generated by applying policy π.

2.6.2. Bellman Expectation Equation for Action-Value Functions

The Bellman expectation equation for action-value functions relates the value of a state-action pair (s, a) to the expected immediate reward plus the discounted value of the next state-action pair.

Definition 2.12 (Bellman Expectation Equation for State-Value Functions)**.** *Let $M := (S, A, P, R, \mu, \gamma)$ be a MDP, and π a policy. Bellman Expectation Equation for Action-Value Functions[52] is defined as:*

$$Q_\pi(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a] \tag{2.11}$$

Where E_π indicates the expected value over the trajectorioes generated by applying policy π.

2.6.3. Bellman Optimality Equation for State-Value Functions

The Bellman Optimality provides a way to express the optimal value of a decision problem recursively in terms of the optimal value of its subproblems.

Definition 2.13 (Bellman Optimality Equation for State-Value Functions)**.** *Let $M := (S, A, P, R, \mu, \gamma)$ be a MDP, and π a policy. Bellman Optimality Equation for State-Value Functions[52] is defined as:*

$$V^*(s) = \max_\pi \mathbb{E}\left[R_{t+1} + \gamma V^*(S_{t+1}) | S_t = s\right] \tag{2.12}$$

Where:

- *$V^*(s)$ denotes the optimal value function for state s in a MDP.*
- *$R(s, a)$ is the immediate reward obtained after taking action a in state s.*
- *$P(s, a, s')$ is the transition probability from state s to state s' after taking action a.*
- *γ is the discount factor.*

2.6.4. Bellman Optimality Equation for Action-Value Functions

The Bellman optimality equation for the action-value function $Q^*(s, a)$ expresses the optimal value of taking action a in state s in terms of the expected immediate reward plus the expected discounted value of the best action to take from the resulting state.

Definition 2.14 (Bellman Optimality Equation for Action-Value Functions)**.** *Let $M := (S, A, P, R, \mu, \gamma)$ be a MDP, and π a policy. Bellman Optimality Equation for Action-Value Functions[52] is defined as:*

$$Q^*(s, a) = \max_\pi \mathbb{E}[R_{t+1} + \gamma Q^*(S_{t+1}, A_{t+1}) | S_t = s, A_t = a] \tag{2.13}$$

Where:

- *$Q^*(s, a)$ is the optimal action-value function, representing the maximum expected return achievable by taking action a in state s.*
- *R_{t+1} is the immediate reward received after taking action a in state s.*

- S_{t+1} *is the resulting state after taking action* a *in state* s*.*
- A_{t+1} *is the action chosen maximizing* Q^* *in the next state.*
- γ *is the discount factor.*
- $\max_\pi$ *denotes the maximization over all possible policies.*

2.6.5. Optimality Condition

The optimality condition in the context of optimization problems refers to a condition or set of conditions that must be satisfied by the optimal solution. In other words, it characterizes the properties or criteria that the optimal solution must meet.

In the context of dynamic programming and the Bellman equations, the optimality condition can be expressed as follows:

Definition 2.15. *For a given optimization problem, the solution is considered optimal if and only if it satisfies the Bellman optimality equation, Equation 2.12 or 2.13. This equation encapsulates the principle of optimality, stating that the value of the optimal solution at any given state must equal the maximum expected return achievable from that state onward, considering all possible actions and transitions[52].*

In summary, the optimality condition in dynamic programming is met when the solution adheres to the Bellman optimality equation, ensuring that the value function or action-value function associated with the optimal policy satisfies the recursive relationship specified by the Bellman equation.

2.7. Solving an MDP

A solution to a Markov Decision Process (MDP) typically refers to finding the optimal policy that maximizes the expected cumulative reward over time. The optimal policy dictates the best action to take in each state to achieve the highest possible long-term reward.

To find the solution to an MDP, specifically the optimal policy, various algorithms can be employed. Before introducing the algorithms, We have to introduce the idea which the algorithms lies on and some necessary steps which these algorithms use.

2.7.1. Bellman Operators

The Bellman operator[20] is a fundamental concept in the field of dynamic programming and reinforcement learning. It is central to the formulation and solution of optimization problems, particularly in the context of MDP.

In simple terms, the Bellman operator defines an equation that expresses the relationship between the value of being in a certain state and the value of taking a certain action in that state. It helps in finding optimal policies by iteratively updating value estimates based on the expected future rewards.

Definition 2.16 (Bellman Operator)**.** *Bellman Operator for state-value functions[52] is defined as:*

$$(B^\pi V)(s) \doteq \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')] \tag{2.14}$$

for all $s \in S$ *and* $V : S \to \mathbb{R}$*. The same can be defined for action-value function.*

The Bellman operator transforms a value function into a value function. The contraction property in the infinity norm refers to the property of a mapping (operator) that contracts distances between points in a metric space. In the context of the Bellman operator, the contraction property can be expressed using the infinity norm as follows:

$$||B^\pi V_1 - B^\pi V_2||_\infty \leq \gamma ||V_1 - V_2||_\infty \tag{2.15}$$

Here, $||\cdot||_\infty$ denotes the infinity norm, V_1 and V_2 are any two value functions, and γ is the discount factor. This property states that applying the Bellman operator reduces the difference between two value functions in the infinity norm by a factor of γ, guaranteeing convergence under certain conditions.

The contraction property is crucial for proving the convergence of iterative algorithms such as value iteration and policy iteration, which repeatedly apply the Bellman operator until convergence. Specifically, it ensures that the sequence of value functions generated by these algorithms converges to a unique fixed point, which corresponds to the optimal value function v^* in the case of optimality.

2.7.2. Policy Iteration Framework

The Policy Iteration Framework is a fundamental approach used in reinforcement learning and dynamic programming to find the optimal policy in a MDP. It involves iteratively

evaluating and improving a policy until convergence to the optimal policy is achieved. The framework consists of two main steps:

Policy Evaluation

In Policy Evaluation step, the current policy's performance is evaluated by estimating the value function (or action-value function) associated with that policy.

Definition 2.17 (Policy Evaluation). *Policy Evalution[52] is to compute the state-value function V_π for an arbitrary policy π. It is also referred as Prediction Problem.*

$$\begin{aligned} V^\pi(s) &= \mathbb{E}^\pi[G_t \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left(r + \gamma V_k(s')\right) \end{aligned} \tag{2.16}$$

for all $s \in S$.

If the environments dynamics are completely known, then 2.16 is a system of $|S|$ simultaneous linear equations in $|S|$ unknowns. However, for our purposes, since the environment's dynamics are not completely known, iterative solution methods are most suitable.

Definition 2.18 (iterative policy evaluation). *Consider a sequence of approximate value functions $v_0, v_1, v_2, \ldots$, each mapping S^{+1} to $\mathbb{R}$ (the real numbers). The initial approximation, v_0, is chosen arbitrarily (except that the terminal state, if any, must be given value 0), and each successive approximation is obtained by using the Bellman equation 2.10 as an update rule for v_π[52]:*

$$\begin{aligned} V_{k+1}(s) &= \mathbb{E}^\pi[G_t \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left(r + \gamma V_k(s')\right) \end{aligned} \tag{2.17}$$

if $k \to \infty$, the sequence of $\{V_k\}$ can be shown in general to converge to V_π.

Pseudo-code of this algorithm is represented in Algorithm 2.1

Policy Improvement

Policy improvement is the process of enhancing the current policy to obtain a better or optimal policy. Formally, policy improvement involves updating the policy based on the

[1] S^+ is what, is this correct way?

Algorithm 2.1 Iterative Policy Evaluation[52]

Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in S^+$, arbitrarily except that $V(\text{terminal}) = 0$
repeat
  $\Delta \leftarrow 0$
  for each $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end for
until $\Delta < \theta$

estimated value function to increase the expected cumulative reward.

Definition 2.19 (Policy Improvement)**.** *The process of making a new policy that improves on an original policy, by making it greedy with respect to the value function of the original policy, is called policy improvement[52]. Mathematically, it can be defined as:*

$$\pi'(a|s) = \arg\max_a Q_\pi(s, a) \tag{2.18}$$

Where:

- $\pi'(a|s)$ *represents the probability of taking action* a *in state* s *under the improved policy* π'*.*
- $Q_\pi(s, a)$ *is the action-value function for policy* π*, representing the expected cumulative reward starting from state* s*, taking action* a*, and then following policy* π*.*

2.7.3. Policy Iteration

Policy iteration is an iterative algorithm used in reinforcement learning for finding an optimal policy for a given MDP. It consists of two main steps: policy evaluation and policy improvement, which are alternated until convergence.

Definition 2.20 (Policy Iteration)**.** *Once a policy,* π*, has been improved using* v^π *to yield a better policy,* π_0*, we can then compute* v^{π_0} *and improve it again to yield an even better* π_1*. We can thus obtain a sequence of monotonically improving policies and value functions[52]:*

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} v_*,$$

where $\xrightarrow{E}$ denotes a policy evaluation and $\xrightarrow{I}$ denotes a policy improvement. Each policy is guaranteed to be a strict improvement over the previous one (unless it is already optimal). Because a finite MDP has only a finite number of policies, this process must converge to an optimal policy and optimal value function in a finite number of iterations.

Pseudo-code of this algorithm is represented in Algorithm 2.2

Algorithm 2.2 Policy Iteration[52]

Initialization:
for each state $s \in S$ **do**
  $V(s) \leftarrow$ arbitrary value in $\mathbb{R}$
  $\pi(s) \leftarrow$ arbitrary action in $A(s)$
end for
Policy Evaluation:
repeat
  $\Delta \leftarrow 0$
  for each state $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end for
until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)
Policy Improvement:
policy-stable $\leftarrow$ true
for each state $s \in S$ **do**
  old-action $\leftarrow \pi(s)$
  $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
  if old-action $\neq \pi(s)$ **then**
    policy-stable $\leftarrow$ false
  end if
end for
if policy-stable **then**
  stop and return $V^\pi \approx v_*$ and $\pi \approx \pi_*$
else
  go to Policy Evaluation
end if

2.7.4. Value Iteration

Value iteration is an iterative algorithm used to compute the optimal value function and optimal policy for a MDP. It's a method for solving reinforcement learning problems, particularly in cases where the state and action spaces are too large to exhaustively search through. The main idea behind value iteration is to iteratively update the value function

until it converges to the optimal value function. This algorithm does not require separate policy evaluation and policy improvement steps like policy iteration.

Definition 2.21 (Value Iteration). *Value iteration[52] is obtained simply by turning the Bellman optimality equation 2.12 into an update rule:*

$$\begin{aligned} V_{k+1}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma V_k(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma V_k(s')\right] \end{aligned} \tag{2.19}$$

for all $s \in S$. For arbitrary V_0, the sequence $\{V_k\}$ can be shown to converge to V_* under the same conditions that guarantee the existence of V_*.

Value iteration converges to the optimal value function and optimal policy in a finite number of iterations, assuming a finite state space and a finite set of actions for each state. It's a computationally efficient method for solving MDPs and is widely used in reinforcement learning and dynamic programming. The pseudo-code of this algorithm is represented in Algorithm 2.3

Algorithm 2.3 Value Iteration [52]

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation

Initialize $V(s)$, for all $s \in S^+$, arbitrarily except that $V(\text{terminal}) = 0$
repeat
  $\Delta \leftarrow 0$
  for each state $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end for
until $\Delta < \theta$
Output: a deterministic policy $\pi \approx \pi^*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

2.8. Temporal Difference Learning

Temporal Difference (TD) learning [53] is a technique used in reinforcement learning for estimating the value function of a MDP directly from experience. It combines ideas from both dynamic programming and Monte Carlo methods. TD learning is particularly useful when the model of the environment is unknown or incomplete. For understanding how TD Learning works, first the TD Error is explained.

2.8.1. Temporal Difference Error

Temporal difference error represents the discrepancy or the "error" between the expected value of a state or state-action pair and the actual observed value. It's essentially a measure of how surprised the RL agent is when it encounters a particular state or state-action pair.

Definition 2.22 (Temporal Difference (TD) Error)**.** *Let $M := (S, A, P, R, \mu, \gamma)$ be a MDP, and π a policy. In TD learning, the TD error is the difference between the estimated value of a state (or state-action pair) and the value obtained from the immediate reward plus the estimated value of the next state (or state-action pair). The TD error[52] δ_t at time step t is defined as:*

$$\delta_t \doteq [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{2.20}$$

Where:

- R_{t+1} *is the immediate reward received after taking action* a_t *in state* s_t*.*
- γ *is the discount factor.*
- S_{t+1} *is the next state the agent transitions to after taking action* A_t *in state* S_t*.*

2.8.2. TD Learning Algorithm

The TD learning algorithm updates the value function estimates based on the TD error. One common TD learning algorithm is the TD(0)[50] algorithm, which updates the value function after each time step using a fixed learning rate α.

Definition 2.23. *The update rule for TD(0) is as follows:[50]*

$$V(s_t) \leftarrow V(s_t) + \alpha\delta_t \tag{2.21}$$

Where:

- $V(s_t)$ *is the estimated value of state* s_t *before the update.*
- α *is the learning rate (a small positive constant).*
- δ_t *is the TD error at time step* t*.*

A pseudo-code of the TD(0) algorithm is represented by Algorithm 2.4. By iteratively

updating the value function estimates based on TD errors, TD learning algorithms can converge to accurate value function estimates, which in turn can be used to derive optimal policies for decision-making in the MDP.

Algorithm 2.4 TD(0) Algorithm [50]

Input π, the policy to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in S^+$, arbitrarily except that $V(\text{terminal}) = 0$
Loop for each episode:
    Initialize S
    Loop for each step of episode:
        A action given by π for S
        Take action A, observe R, S_0
        $V(S) \leftarrow V(S) + \alpha \left(R + \gamma V(S_0) - V(S)\right)$
        $S \leftarrow S_0$
    until S is terminal

2.9. Model-Free Reinforcement Learning

In model-free reinforcement learning (RL), the agent learns to make decisions by interacting directly with the environment without explicit knowledge of the environment's dynamics. The agent learns from experience (trial-and-error), typically through the use of value functions or direct policy search methods.

Model-free RL algorithms, such as Q-learning[59], SARSA[41], and Monte Carlo methods, directly learn policies or value functions from experience without requiring a model of the environment. This makes model-free reinforcement learning suitable for environments where the dynamics are complex or unknown.

There are two main types of model-free reinforcement learning algorithms: value-based methods and policy-based methods.

2.9.1. Value-Based Methods

Value-based methods aim to learn the value function, which estimates the expected cumulative reward that can be obtained from a given state or state-action pair under a certain policy. The value function can be used to derive an optimal policy by selecting actions that maximize the expected cumulative reward. Examples of value-based methods include Q-learning[59], SARSA[41], and Deep Q-Networks (DQN)[31].

2.9.2. Policy-Based Methods

Policy-based methods directly learn the policy, which is a mapping from states to a distribution over actions, without explicitly estimating the value function. The policy is updated to maximize the expected cumulative reward over time. Policy-based methods can be further divided into deterministic policy gradient methods and stochastic policy gradient methods. Examples of policy-based methods include REINFORCE [61], Proximal Policy Optimization (PPO)[44], and Trust Region Policy Optimization (TRPO)[43].

2.9.3. Model-free reinforcement learning algorithm steps

Model-free reinforcement learning algorithm typically involve the following steps:

1. **Initialization**: Initialize the parameters of the value function or policy.
2. **Interaction with the Environment**: The agent interacts with the environment by taking actions based on its current policy and receiving feedback in the form of rewards and next states.
3. **Update**: Update the parameters of the value function or policy based on the observed rewards and transitions. This is usually done using gradient descent or other optimization techniques.
4. **Repeat**: Repeat the process of interaction and update until convergence or a stopping criterion is met.

2.9.4. A Model-Free Algorithm:Q-Learning

One of the early breakthroughs in reinforcement learning was the development of an algorithm known as Q-learning [59]. Q-learning is a model-free reinforcement learning algorithm used to find an optimal policy for a MDP. It's particularly effective in environments where the agent doesn't have prior knowledge of the dynamics of the environment. Q-learning is known as an off-policy learning algorithm, meaning that it learns the value of the optimal policy independently of the agent's actions.

Definition 2.24 (Q-learning)**.** *The learned action-value function, Q, directly approximates q_*, the optimal action-value function, independent of the policy being followed. The agent updates Q based on experiences, which are tuples of the form $(S_t, A_t, R_{t+1}, S_{t+1})$.*

So, the update rule[52] is defined as follows:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right] \tag{2.22}$$

Where:

- α *is the step size or learning rate, determining the rate at which the action-value function is updated.*
- $\max_{a'} Q(S_{t+1}, a')$ *represents the maximum expected return the agent can achieve by taking any action in the next state* S_{t+1}*.*
- *The quantities* S_t*,* A_t*,* R_{t+1}*, and* S_{t+1} *are components of the agent's experience tuple, where* S_t *is the current state,* A_t *is the action taken,* R_{t+1} *is the reward received after taking action* A_t*, and* S_{t+1} *is the next state the agent transitions to.*

This dramatically simplifies the analysis of the algorithm and enabled early convergence proofs. Pseudo-code of this algorithm is represented in Algorithm 2.5.

Algorithm 2.5 Q-Learning[59]

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
Initialize $Q(s, a)$, for all $s \in S^+$, $a \in A(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$
Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose A from S using policy derived from Q (e.g., ϵ-greedy)
        Take action A, observe R, S_0
        $Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \max_a Q(S_0, a) - Q(S, A) \right]$
        $S \leftarrow S_0$
until S is terminal

2.10. Model-Based Reinforcement Learning

In model-based reinforcement learning, the agent learns a model of the environment's dynamics, which it uses to simulate possible future states and rewards. With this model, the agent can plan ahead and make decisions by considering the consequences of its actions.

In model-based RL, the agent uses its learned model of the environment to simulate possible trajectories and plan its actions. Planning algorithms, such as dynamic programming, Monte Carlo tree search, or model-based reinforcement learning algorithms, are

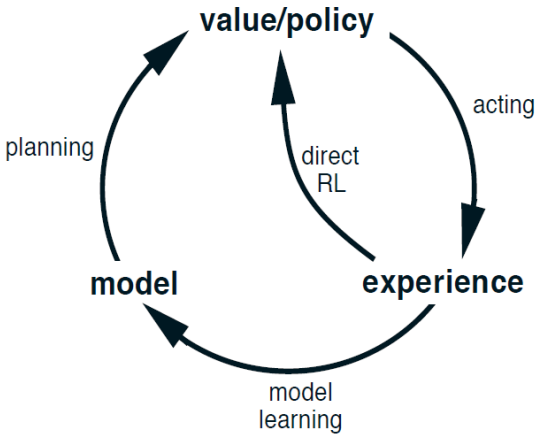


Figure 2.1: General workflow of agents in Model-Based approaches [52]

used to optimize the agent's policy. a general workflow of agents is represented in Figure 2.1.

2.10.1. A Model-Based Algorithm: Dyna

The Dyna[51] architecture integrates model learning with direct reinforcement learning, enabling agents to make use of simulated experiences to enhance their learning efficiency.

The key idea behind Dyna is to supplement the agent's real experiences (obtained through interaction with the environment) with simulated experiences generated by a learned model of the environment. By using the model to simulate additional experiences, the agent can potentially learn more quickly and effectively than by relying solely on real experiences. The architecture of Dyna is represented in Figure 2.2.

2.11. Function Approximation

The application of function approximators, such as Neural Networks, in estimating continuous value functions or implementing parametrized policies, has expanded the capabilities of RL to tackle more intricate tasks. In the following section, we will delve into the fundamentals of function approximation and Neural Networks, which will pave the way for our discussion on Deep RL algorithms in Section 2.12.

2.11.1. Basics of Function Approximation

A function approximation is a mathematical technique used to represent a complex or unknown function by a simpler one that is easier to work with or compute. This

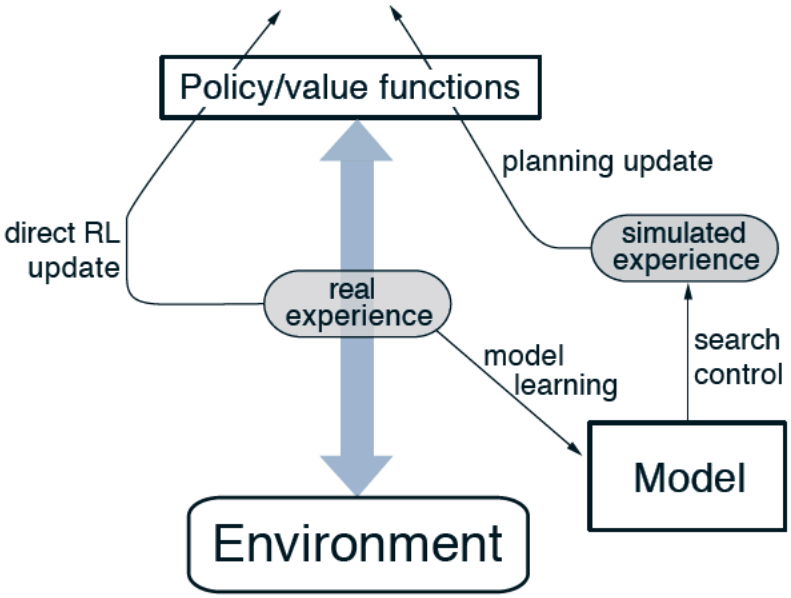


Figure 2.2: The general Dyna Architecture. Real experience, passing back and forth between the environment and the policy, affects policy and value functions in much the same way as does simulated experience generated by the model of the environment. [51, 52]

is particularly useful in various fields such as engineering, physics, economics, and machine learning, where it might be impractical or computationally expensive to directly manipulate or compute the original function.

The process of function approximation involves finding a function from a predefined class (e.g., polynomials, trigonometric functions, neural networks) that closely matches the behavior of the original function within a certain range or over a set of data points. This can be achieved using various methods such as interpolation, regression, or optimization techniques.

Error Calculation

The error in function approximation quantifies the difference between the original function and its approximation.

Definition 2.25 (Function Approximation Error)**.** *Let $f(x)$ be the original function and $g(x)$ be its approximation. We define the 1. Absolute Error as:*

$$E_{abs}(x_i) = |f(x_i) - g(x_i)| \tag{2.23}$$

and 2. Squared Error as:

$$E_{squared}(x_i) = (f(x_i) - g(x_i))^2 \tag{2.24}$$

Where x_i represents the input values (or data points) at which we are evaluating the functions $f(x)$ and $g(x)$.

The total error over a set of points can be calculated by averaging or summing the errors over all points. For example, the mean squared error (MSE) is often used.

Definition 2.26 (mean squared error)**.** *Mean squared error is a common metric used to evaluate the performance of a regression model. It calculates the average of the squares of the differences between the predicted values and the actual values*

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - g(x_i))^2 \tag{2.25}$$

where n is the number of data points.

Optimum Function Approximation

The optimum function approximation aims to minimize the error between the original function and its approximation. Mathematically, the optimum function approximation can be found by minimizing a suitable error metric.

Definition 2.27 (Optimum Function Approximation)**.** *Let $E(g)$ represent the error function, which depends on the choice of error metric and the approximation function $g(x)$. Then, the optimum function approximation $g^*(x)$ can be defined as:*

$$g^*(x) = \arg\min_g E(g) \tag{2.26}$$

This means $g^*(x)$ is the function $g(x)$ that minimizes the error metric $E(g)$.

2.11.2. Neural Networks

Neural networks have emerged as powerful tools in the domain of machine learning and artificial intelligence. They are computational models inspired by the structure and functioning of the human brain. In this section, we provide a brief explanation of neural networks.

Neuron

A neuron is the fundamental unit of a neural network. Mathematically, it can be represented as a function that takes inputs, performs computations, and produces an output.

Definition 2.28 (Neuron). *Let $x_1, x_2, ..., x_n$ denote the inputs to the neuron, $w_1, w_2, ..., w_n$ represent the weights associated with each input, and b denote the bias. The output y of the neuron is calculated using an activation function f as follows:*

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2.27}$$

Activation Function

An activation function introduces non-linearity into the output of a neuron, enabling neural networks to learn complex patterns in data. Commonly used activation functions include sigmoid, hyperbolic tangent (tanh), and rectified linear unit (ReLU). The plots of these activation functions are depicted in Figure 2.3.

Layer

A layer in a neural network is a collection of neurons that process inputs independently and produce outputs collectively. Layers are stacked sequentially to form the architecture of the neural network. There are typically three types of layers: input layer, hidden layers, and output layer.

Feedforward Neural Network

A feedforward neural network is the simplest form of neural network architecture, where connections between neurons do not form cycles. Information within a neural network flows unidirectionally, traversing from the input layer through hidden layers to the output layer. The input layer serves as the entry point, receiving external data inputs. Hidden layers, situated between the input and output layers, engage in intricate processing of information passed from preceding layers. Ultimately, the output layer receives the refined output from the hidden layers, furnishing the final results. These results manifest in diverse forms tailored to address a wide array of tasks

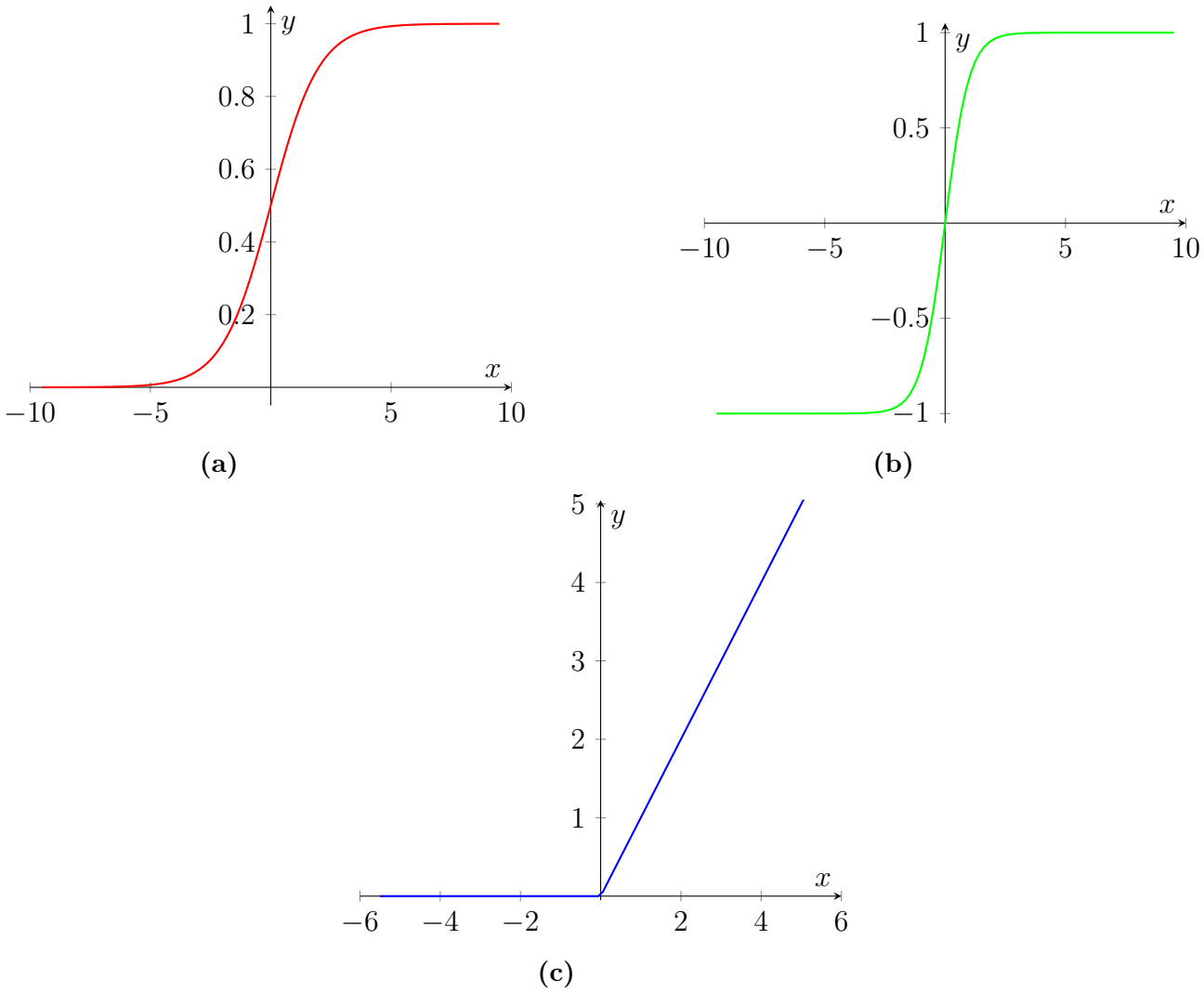


Figure 2.3: Graphs of various mathematical functions plotted on separate axes. Figure (a) shows the sigmoid function, (b) depicts the hyperbolic tangent function, and (c) illustrates the ReLU (Rectified Linear Unit) function.

2.11.3. Functioning of Neural Networks

The functioning of a neural network involves two main processes: forward propagation and backpropagation.

Forward Propagation

Forward propagation refers to the process of passing input data through the neural network to obtain predictions. During forward propagation, inputs are multiplied by the weights, biases are added, and activation functions are applied successively across the layers until the final output is obtained.

Backpropagation

Backpropagation[40] is an algorithm used to compute the gradients of the loss function with respect to the parameters of the neural network. It efficiently propagates the error backwards through the network, layer by layer, using the chain rule of calculus.

Definition 2.29 (Backpropagation)**.** *Let L denote the loss function of the neural network. based on [40], the gradients of the loss function with respect to the parameters of the network can be computed using the chain rule as follows:*

$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial a^{[L]}} \frac{\partial a^{[L]}}{\partial z^{[L]}} \frac{\partial z^{[L]}}{\partial \theta} \tag{2.28}$$

Where:

- $\frac{\partial L}{\partial a^{[L]}}$ *is the gradient of the loss function with respect to the activation output of the last layer.*
- $\frac{\partial a^{[L]}}{\partial z^{[L]}}$ *is the derivative of the activation function with respect to the weighted sum of inputs of the last layer.*
- $\frac{\partial z^{[L]}}{\partial \theta}$ *is the derivative of the weighted sum of inputs of the last layer with respect to the parameters θ.*

Similarly, the gradients for the parameters of the preceding layers can be computed recursively using the chain rule.

Gradient Update in Neural Networks

In neural networks, the process of updating the parameters (weights and biases) during training is crucial for learning from the data. This process involves computing the gra-

dients of a loss function with respect to the parameters and adjusting the parameters in the direction that minimizes the loss. The most commonly used optimization algorithm for this purpose is gradient descent[40].

Definition 2.30 (Gradient Descent). *Based on [40], gradient descent is an iterative optimization algorithm used to minimize the loss function by updating the parameters in the direction opposite to the gradient of the loss function. The general update rule for parameters θ using gradient descent is given by:*

$$\theta = \theta - \alpha \frac{\partial L}{\partial \theta} \tag{2.29}$$

Where:

- *θ represents the parameters of the neural network (weights and biases).*
- *α is the learning rate, which controls the size of the steps taken during optimization.*
- *$\frac{\partial L}{\partial \theta}$ is the gradient of the loss function $L(\theta)$ with respect to the parameters θ, calculated using Backpropagation algorithm.*

The gradient of the loss function is typically computed using the backpropagation algorithm, which efficiently calculates the gradients of the loss function with respect to each parameter in the network. By iteratively applying the gradient descent update rule and backpropagation algorithm, neural networks can learn to minimize the loss function and make accurate predictions on unseen data.

2.12. Deep Reinforcement Learning

The combination of Reinforcement Learning and Deep Artificial Neural Networks, with the emphasis on the multiple hidden layers denoted by "Deep," has allowed RL to successfully tackle tasks that were previously unsolvable by other methods. Deep reinforcement learning incorporates deep neural networks as function approximators to handle high-dimensional state spaces. By using deep learning techniques, DRL can effectively learn complex representations from raw sensory inputs (such as images or sensor data) and map them to appropriate actions. In this segment, we elaborate on our selection of Actor-Critic methods to achieve this objective. Additionally, we provide an overview of policy-based and value-based approaches, which are fundamental to understanding the Actor-Critic methods.

2.12.1. Value-Based Methods

Value-based methods focus on estimating the value function, which predicts the expected return of states or state-action pairs. The most well-known value-based algorithm is Q-learning[59]. In Q-learning, a Q-function $Q(s, a)$ is learned, representing the expected return of taking action a in state s and following the optimal policy thereafter. Deep Q-Networks (DQNs) [32] extend Q-learning to high-dimensional state spaces by using deep neural networks to approximate the Q-function. DQNs have achieved remarkable success in playing Atari games from raw pixel inputs. However, value-based methods often struggle with continuous action spaces and can be sample-inefficient due to the need for extensive exploration.

2.12.2. Policy-Based Methods

Policy-based methods, on the other hand, directly parameterize the policy, which maps states to actions. These methods optimize the policy by maximizing a cumulative reward objective. The REINFORCE algorithm [61] is a fundamental policy-based method that uses the policy gradient theorem to update the policy parameters. While policy-based methods can handle continuous action spaces more naturally and often exhibit better convergence properties, they can suffer from high variance in gradient estimates, leading to unstable training. Advanced techniques such as Trust Region Policy Optimization (TRPO) [43] and Proximal Policy Optimization (PPO) [44] have been developed to address these challenges by imposing constraints on policy updates, ensuring more stable and efficient learning.

2.12.3. Actor-Critic Methods

Actor-Critic (AC) methods combine the strengths of both value-based and policy-based approaches. These methods maintain two separate networks: the actor, which represents the policy, and the critic, which estimates the value function. The critic helps reduce the variance of the gradient estimates by providing a baseline (value estimate) to the actor. Notable Actor-Critic algorithms such as TRPO [43] and PPO [44] have achieved good empirical results. These algorithms work on-policy and utilize parameterized value or advantage functions based on Monte Carlo updates. These updates use full trajectories without bootstrapping, which leads to unbiased estimates but can result in higher variance due to potential deviations between trajectories during training. Additionally, these algorithms impose a constraint on the difference between the new and old policies during each update.

Also, Deep Deterministic Policy Gradient (DDPG) [28] and Soft Actor-Critic (SAC) [19] are two famous Actor-Critic algorithms, which utilize Temporal-Difference (TD) update rules. These methods are designed to handle continuous action spaces more effectively. DDPG combines the deterministic policy gradient with the actor-critic architecture, while SAC incorporates entropy regularization to encourage exploration and robustness. Once we explain how these algorithms work, we will have all the necessary tools to explain the proposed method.

2.13. Multi-Arm Bandit

The multi-armed bandit (MAB) problem is a decision-making problem where a decision-maker (or agent) is faced with a set of K options, called "arms." Each arm has an unknown reward distribution. At each time step, the decision-maker chooses an arm to pull, observes a reward drawn from the chosen arm's distribution, and aims to maximize the cumulative reward over a sequence of trials.

The challenge lies in balancing exploration and exploitation. Various algorithms, such as the Upper Confidence Bound (UCB)[3] is used to address this trade-off and maximize the cumulative reward.

The MAB problem finds applications in various fields, including online advertising, clinical trials, and RL, where decisions need to be made sequentially under uncertainty.

2.13.1. Upper Confidence Bound

The UCB algorithm is a classic approach to solve the multi-armed bandit problem by balancing exploration and exploitation. The basic idea is to select arms based on their potential for high rewards, incorporating uncertainty estimates into the decision-making process.

Consider K arms, indexed as $i = 1, 2, \ldots, K$.

Action Selection

At each time step t, the UCB algorithm selects the arm A_t based on the following rule:

$$A_t = \arg\max_i \left[Q_i(t) + c\sqrt{\frac{\ln(t)}{N_i(t)}} \right] \tag{2.30}$$

where:

- $Q_i(t)$ is the estimated mean reward of arm i up to time t,
- $N_i(t)$ is the number of times arm i has been selected up to time t,
- c is a constant that controls the exploration-exploitation trade-off. Typically, c is chosen based on theoretical analysis or empirical tuning.

Update Rules

After selecting arm A_t and observing the reward R_t:

$$N_{A_t}(t+1) = N_{A_t}(t) + 1 \tag{2.31}$$

$$Q_{A_t}(t+1) = \frac{R_t + N_{A_t}(t) \cdot Q_{A_t}(t)}{N_{A_t}(t+1)} \tag{2.32}$$

These update rules ensure that $Q_i(t)$ converges to the true mean reward of arm i as t goes to infinity.

Key Points

- The UCB algorithm balances exploration (through the uncertainty term) and exploitation (by selecting arms with high estimated rewards).
- It guarantees logarithmic regret, meaning that the cumulative regret (difference between the total reward obtained and the maximum possible reward) grows logarithmically with time.
- UCB is widely used in various applications, including online advertising, healthcare, and reinforcement learning.

3 | Related Works

In this chapter, we first explore the application of Deep Reinforcement Learning for continuous control problems. Secondly, we investigate the innovative application of curriculum learning within the domain of RL. Finally, we examine how reinforcement learning is used in robotics.

3.1. Deep RL for Continuous Control

Reinforcement learning (RL) has emerged as a powerful tool for training agents to achieve goals in complex environments. However, applying RL to robotics tasks often involves continuous control problems. In these problems, the agent's actions are not limited to a discrete set of choices, but rather take the form of continuous values. This presents unique challenges compared to discrete action spaces. For instance, continuous control tasks require the agent to learn precise motor skills and navigate high-dimensional action spaces efficiently [10]. In the following, the state-of-the-art algorithms for handling continuous control problems will be introduced.

3.1.1. Deep Deterministic Policy Gradient

DDPG[28] is a model-free, off-policy actor-critic algorithm designed for learning in continuous action spaces. It integrates the deterministic policy gradient (DPG)[45] with deep Q-learning (DQN)[31] approaches. Also, it is known as the Actor-Critic variant of DQN[45]. Understanding its implementation will be beneficial for comprehending the subsequent algorithm Soft Actor-Critic[19].

Actor-Critic Framework

DDPG employs an actor-critic framework comprising:

- The **actor** function $\mu(s|\theta^\mu)$, parameterized by θ^μ, maps states to actions.
- The **critic** function $Q(s, a|\theta^Q)$, parameterized by θ^Q, estimates the Q-value of action a in state s.

Objective Functions

1. **Actor:** The actor aims to maximize the expected return:

$$J = \mathbb{E}_{s\sim p^{\mu}, a\sim\mu}[Q(s, a|\theta^{Q})] \tag{3.1}$$

   where p^{μ} is the state distribution under policy μ.
   Then, the actor policy is updated using the sampled policy gradient:

$$\nabla_{\theta^{\mu}} J \approx \mathbb{E}_{s\sim p^{\mu}}[\nabla_a Q(s, a|\theta^{Q})|_{a=\mu(s)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})] \tag{3.2}$$

2. **Critic:** The critic is updated using the Bellman equation 2.10, aiming to minimize the following loss:

$$L = \mathbb{E}_{s,a,r,s'}[(Q(s, a|\theta^{Q}) - y)^2] \tag{3.3}$$

   Where:

   - $y = r + \gamma Q'(s', \mu'(s'|\theta^{\mu'})|\theta^{Q'})$ is the target. Q' and μ' represent the target critic and target actor networks, respectively.
   - r is the reward.
   - s' is the next state.
   - γ is the discount factor.

3. **Target Networks:** Target Networks weights $\theta^{Q'}$ and $\theta^{\mu'}$ are updated softly using a technique known as Polyak averaging[37] as follows:

$$\bar{\theta}_t = (1 - \alpha_t)\bar{\theta}_{t-1} + \alpha_t \theta_t \tag{3.4}$$

   where:

   - $\bar{\theta}_t$ is the averaged parameter vector at iteration t.
   - α_t is the decay factor (also called the step size or learning rate).
   - θ_t is the parameter vector at iteration t.

Pseudo-code of this algorithm is represented in Algorithm 3.1.

Exploration vs. Exploitation

Exploration is facilitated by adding noise, such as the Ornstein-Uhlenbeck process[56], to the actor's actions, balancing exploration with the exploitation of the learned policy.

Algorithm 3.1 DDPG Algorithm[28]

Require: Initial policy and Q-function parameters: θ and ϕ, empty replay buffer $\mathcal{D}$
Initialize target parameters: $\overline{\theta} \leftarrow \theta$, $\overline{\phi} \leftarrow \phi$
Initialize replay buffer $\mathcal{D}$
for episode $= 0, \ldots, M$ **do**
  Receive initial state s_0
  for $t = 0, \ldots, T$ **do**
    Execute action $a = \pi_\theta(s_t) + N_t$ and observe r_t, s_{t+1}
    Store transition: $\mathcal{D} \leftarrow \mathcal{D} \cup \{\langle s_t, a_t, r_t, s_{t+1} \rangle\}$
    Sample minibatch of N transitions from $\mathcal{D}$
    Update critic by minimizing Equation 3.3
    Update Actor by maximizing Equation 3.1
    Update target networks using Equations 3.4
  end for
end for

3.1.2. Soft Actor-Critic

Soft Actor-Critic[19] is a cutting-edge algorithm that has demonstrated exceptional performance in various continuous state-action benchmark environments, including robotics tasks. In order to enhance stability and mitigate overestimation bias [16, 54] in Q value estimates, SAC utilizes two parameterized critics, $Q_{\phi 1}$ and $Q_{\phi 2}$, which are trained on the same samples but differ in initialization. When an estimate is required, the minimum value between the two critics, $Q_{LB}(s,a) = \min(Q_{\phi 1}(s,a), Q_{\phi 2}(s,a))$, is computed. Additionally, SAC incorporates an entropy-regularized [69] framework that significantly improves exploration and robustness. This is achieved by modifying the objective to favor stochastic policies as follows:

$$J(\pi) = \sum_{t=0}^{T} \gamma^t \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t,a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t))] \tag{3.5}$$

where $\mathcal{H}$ is the entropy of the current policy.
The trajectory distribution induced by policy π_ϕ is denoted as ρ_ϕ, while α, referred to as temperature, determines the relative importance of entropy compared to standard reward. As a result, the Soft Bellman equation for Q is formulated as follows:

$$Q(s,a) = \mathbb{E}_{s' \sim p,\, a' \sim \pi} \left[r(s,a) + \gamma \left(Q(s',a') + \alpha \mathcal{H}(\pi(\cdot|s')) \right) \right] \tag{3.6}$$

where:

- $Q(s,a)$ is the action-value function.
- s' is the next state sampled from the environment's transition dynamics P.
- $r(s,a)$ is the immediate reward obtained after taking action a in state s.
- γ is the discount factor.
- π_ϕ is the policy parameterized by ϕ, which outputs a probability distribution over actions given states.
- a' is the next action sampled from π_ϕ given the next state s'.
- α is the temperature parameter that controls the level of entropy regularization.

The predominant approach in implementing SAC involves utilizing a squashed Gaussian policy. This means that the parameters (denoted as θ) are adjusted or optimized to discover the optimal values for both the mean and variance of the distribution produced as output for each input state:

$$\pi_\theta(s) = \tanh(\mu_\theta(s) + \sigma_\theta(s)\xi), \quad \xi \sim \mathcal{N}(0, I) \tag{3.7}$$

Therefore, the critic loss and actor objective are respectively:

$$\begin{aligned} L(\phi_i, \mathcal{D}) = \mathbb{E}_{\langle s,a,r,s' \rangle \sim \mathcal{D}} \Bigg[\Bigg(Q_{\phi_i}(s,a) - r - \gamma \Bigg(\min_{j=1,2} Q_{\bar{\phi}_j}(s',a') \\ - \alpha \log \pi_\theta(a'|s') \Bigg) \Bigg)^2 \Bigg], \quad a' \sim \pi_\theta(\cdot|s') \end{aligned} \tag{3.8}$$

$$J(\theta, \mathcal{D}) = \mathbb{E}_{s \sim \mathcal{D}} \left[\min_{j=1,2} \left(Q_{\phi_j}(s,a) - \alpha \log \pi_\theta(a|s) \right) \right], a \sim \pi_\theta(\cdot|s') \tag{3.9}$$

The pseudo-code of this algorithm is represented in Algorithm 3.2.

3.2. Curriculum Learning in Reinforcement Learning

The idea of using curricula to train artificial agents dates back to the early 1990s, with early applications in grammar learning[11] , robotics control problems[2] , and classifica-

Algorithm 3.2 SAC Algorithm

Require: Initial policy and Q-function parameters: θ and ϕ_i, empty replay buffer $\mathcal{D}$
1: Initialize target parameters: $\overline{\phi}_i \leftarrow \phi_i$
2: Initialize replay buffer $\mathcal{D}$
3: **for** each iteration **do**
4: **for** each environment step **do**
5: Execute action $a_t \sim \pi_\theta(\cdot|s_t)$ and observe r_t, s_{t+1}
6: Store transition: $\mathcal{D} \leftarrow \mathcal{D} \cup \{\langle s_t, a_t, r_t, s_{t+1}\rangle\}$
7: **end for**
8: **for** each training step **do**
9: Sample minibatch of N transitions from $\mathcal{D}$
10: Update critic by minimizing Equation 3.8
11: Update Actor by maximizing Equation 3.9
12: Update target networks using Equation 3.4
13: **end for**
14: **end for**

tion problems[5] . Directly solving complex tasks in reinforcement learning can be challenging due to issues such as hard-exploration, safety concerns, and other constraints. To address these challenges, researchers often construct sequences of easier tasks, gradually increasing in difficulty, to facilitate the learning process. In this section, we will discuss the key factors involved in building a curriculum for reinforcement learning. Following this, we will introduce various approaches to managing curriculum learning. Finally, we will present an important related work that is closely aligned with our research.

3.2.1. Parameters for Designing Curricula in Reinforcement Learning

Building a curriculum in Reinforcement Learning involves structuring a sequence of learning tasks that guide an agent from simple to complex behaviors. This approach, known as curriculum learning, can significantly enhance the efficiency and effectiveness of the training process. Several parameters can be adjusted to design an optimal curriculum, including task complexity, reward shaping, environment dynamics, state representation, transfer learning strategies, and etc.

Task Complexity

Task complexity refers to the difficulty level of the tasks presented to the agent. A well-designed curriculum starts with simpler tasks that gradually increase in complexity. This allows the agent to build foundational skills before tackling more challenging problems.

For example, in a navigation task, an agent might first learn to move in an open space before being introduced to environments with obstacles[33].

Reward and Initial/Terminal State Distribution Changes

Changing the distribution[33] is to generate distinct MDPs for intermediate tasks by modifying certain aspects of the MDP. Florensa et al.[13] introduced the concept of reverse curriculum generation, which involves an algorithm that produces a range of initial states that progressively move away from the desired goal. This approach relies on having at least one known goal state, which serves as a starting point for the expansion process. To generate nearby initial states, a random walk is taken from existing starting states, with actions selected using some level of noise perturbation. Florensa et al.[14] have also explored a reverse "forward" expansion strategy. This particular approach enables an agent to autonomously identify various objectives within the state space, consequently facilitating the exploration of said space. The authors achieve this identification using a Generative Adversarial Network (GAN) (Goodfellow et al.[17]), wherein the generator network suggests goal regions (defined subsets of the state space) and the discriminator assesses the suitability of the goal region in terms of the agent's current skill level. The agent undergoes training on tasks recommended by the generator. The objectives produced by the GAN evolve over time to mirror the varying complexity of the tasks, gradually transitioning from states that are near the initial state to those that are more distant. Instead of modifying the initial or terminal state distribution, another approach is to modify the reward function. SAC-X (Scheduled Auxiliary Control), introduced by Riedmiller et al.[39], is an algorithm that schedules and executes auxiliary tasks. These tasks enable the agent to efficiently explore its environment and make progress towards solving the final task. Auxiliary tasks have the same state, action, and transition function as the original MDP, but their reward function differs. The rewards used in these auxiliary tasks correspond to changes in raw or high-level sensory input. The method employed is a hierarchical reinforcement learning (RL) technique: it involves two main steps. Firstly, the system must acquire knowledge of intentions, which are policies governing auxiliary tasks. Secondly, it must learn the scheduler, which arranges the sequence of intention policies and auxiliary tasks. In order to grasp the intentions, the system maximizes the action-value function of each intention. This is achieved by following the starting state distribution that arises from implementing each of the other intention policies.

Environment Dynamics

Environment dynamics pertain to the properties and variability of the environment in which the agent operates. Initially, the environment can be simplified with fewer variables and less randomness. As the agent improves, the environment can be made more dynamic and stochastic, which can better prepare the agent for real-world scenarios. For example, the agent might first operate in a static environment and later adapt to environments where obstacles appear randomly[13].

State Representation

The representation of the state space can be altered throughout the curriculum. Early tasks might use a simplified or abstract representation of the state to reduce the learning burden. As the agent progresses, the state representation can become more detailed and realistic. This gradual increase in complexity helps the agent to develop a more sophisticated understanding of the environment[33].

Transfer Learning Strategies

Transfer learning involves leveraging knowledge gained from previous tasks to accelerate learning in new tasks. This can be implemented by reusing parts of the policy or value function learned in earlier tasks. Effective curricula use transfer learning to ensure that the skills and knowledge acquired in simpler tasks are applied to more complex tasks, thereby improving learning efficiency[13].

Co-Learning

Co-learning[33] is a multi-agent approach to curriculum learning, in which the curriculum emerges from the interaction of several agents (or multiple versions of the same agent) in the same environment. These agents may act either cooperatively or adversarially to drive the acquisition of new behaviors, leading to an implicit curriculum where both sets of agents improve over time. Sukhbaatar et al.[49] proposed a novel method called asymmetric self-play that allows an agent to learn about the environment without any external reward in an unsupervised manner. This method considers two agents, a teacher and a student, using the paradigm of "the teacher proposing a task, and the student doing it." The two agents learn their own policies simultaneously by maximizing interdependent reward functions for goal-based tasks. This process is iterated to automatically generate a curriculum of intrinsic exploration. Baker et al.[4] have proven that a physically grounded task can give rise to more intricate behaviors. In particular, their study revolves around a hide and seek game involving two teams of agents. The first team is tasked with hiding

using obstacles and environmental items, while the second team's objective is to locate the first team. Through their research, they successfully demonstrated that as one team developed a successful strategy, it exerted pressure on the other team to adapt and learn a counter-strategy. This iterative process led to the emergence of a curriculum consisting of progressively competitive agents.

Sequencing

One method of implementing curriculum learning involves sequencing [33], which can be utilized with either samples or tasks. The objective of sequencing, when presented with a collection of tasks or samples, is to arrange them in a manner that enhances the learning process. Various sequencing techniques are available, each based on its own underlying assumptions. A key assumption of curriculum learning is the ability to manipulate the environment in order to generate diverse tasks. The level of control that a practitioner possesses in shaping the environment will determine the suitability of different sequencing approaches. Bengio et al.[5] showed that ordering the training samples from simple to complex can improve learning speed and generalization ability in the supervised learning tasks. Many current Reinforcement Learning approaches use this idea, for example, Deep Q Networks (DQN)[32] use a replay buffer to store past state-action-reward experience tuples. At each training step, experience tuples are sampled from the buffer and used to train DQN in minibatches. The original formulation of DQN performed this sampling uniformly randomly. However, as in the supervised setting, samples can be reordered or "prioritized," according to some measure of usefulness or difficulty, to improve learning. Schaul et al.[42] proposed Prioritized Experience Replay (PER), which prioritizes and replays important transitions more. Important transitions are those with high expected learning progress, which is measured by their temporal difference (TD) error. Intuitively, replaying samples with larger TD errors allows the network to make stronger updates. As transitions are learned, the distribution of important transitions changes, leading to an implicit curriculum over the samples.

Human-in-the-Loop Curriculum Generation

An alternative method involves curriculum sequencing approaches that are manually conducted by humans[33] who are either domain experts with specialized knowledge in the problem domain, or by naive users who may lack understanding of the problem domain and/or machine learning. Stanley et al.[47] conducted a study that exemplifies the manual generation of curriculum by domain experts. In their research, they focused on enhancing the engagement of video games by enabling agents to adapt and improve

through player interaction. The NeuroEvolving Robotic Operatives (NERO) game was utilized as a platform, where virtual robots initially possessed no skills and had to acquire complex behaviors to effectively participate in the game. Acting as trainers, human players designed a curriculum consisting of various training scenarios to educate a team of simulated robots for military combat. The study demonstrated the successful training of these virtual robots in acquiring advanced battle tactics through the curriculum devised by the domain experts. Khan et al.[25] conducted behavioral studies to examine the teaching strategies employed by inexperienced users. Human participants were tasked with teaching a robot how to determine if an object can be grasped with one hand. The researchers identified three different teaching strategies used by the participants, with one aligning with the curriculum learning principle. This strategy involved beginning with simple concepts and progressively introducing more challenging tasks.

3.2.2. Approaches for Handling Curriculum in Reinforcement Learning

A critical aspect of reinforcement learning is the design and management of the curriculum, which guides the learning process by structuring the tasks and experiences that an RL agent encounters. Curriculum Learning (CL) can be broadly classified into two categories: automatic and manual curriculum learning. Additionally, CL strategies can be implemented in two primary ways: static and dynamic curriculum learning. This section will first discuss the principles, methodologies, advantages, and challenges associated with automatic versus manual curriculum learning. Following that, the differences between static and dynamic curriculum learning will be explored in detail.

Manual Curriculum Learning

In manual curriculum learning, the sequence of tasks or data samples is predefined by domain experts based on prior knowledge and heuristics. This involves creating a curriculum that dictates the order in which the training data is presented to the model. The key aspects of manual CL include:

- **Expert Knowledge**: Experts design the curriculum based on their understanding of the task complexity and the model's learning process.
- **Heuristic Design**: The curriculum often relies on heuristics or empirical rules to determine the progression of training samples.

Example: In training a robotic arm using RL, one might manually design a curriculum

where the robot first learns to move to a specified position in free space before progressing to tasks involving obstacle avoidance and finally to complex manipulation tasks like picking and placing objects. For instance, Pinto and Gupta [36] demonstrated that training robotic agents with a manually designed curriculum of incrementally challenging tasks improved the efficiency and effectiveness of the learning process.

Advantages:

- Leverages domain expertise to guide the learning process.
- Can be tailored to specific tasks and datasets.
- Often easier to understand and implement initially.

Challenges:

- Time-consuming and labor-intensive to design effective curricula.
- May not generalize well to different tasks or domains.
- Risk of human biases influencing the curriculum design.

Automatic Curriculum Learning

Automatic curriculum learning, on the other hand, involves algorithms that dynamically determine the sequence of tasks or data samples based on the model's performance and learning progress. This approach relies on automated techniques to adaptively construct the curriculum, often through reinforcement learning, optimization, or other adaptive methods.

- **Adaptive Algorithms**: Algorithms automatically adjust the training sequence based on certain criteria, such as model performance metrics or learning progress.
- **Optimization-Based**: Techniques like reinforcement learning or gradient-based optimization are used to identify the optimal sequence of tasks.

Example: Florensa et al.[13] proposed an automatic curriculum learning approach where tasks are sampled based on the learning progress of the agent, leading to more efficient learning in robotic manipulation tasks.

Advantages:

- Reduces the need for expert intervention.
- Can adapt to the model's learning dynamics in real-time.
- Potentially more scalable and generalizable across different tasks and domains.

Challenges:

- Increased computational complexity and resource requirements.
- Designing effective automated algorithms can be challenging.
- Potentially less interpretable compared to manual curricula.

Automatic vs. Manual

When comparing manual and automatic curriculum learning, several key distinctions emerge:

- **Design Effort**: Manual CL requires significant upfront effort and expertise, while automatic CL shifts this effort to designing adaptive algorithms.
- **Flexibility and Adaptability**: Automatic CL is more flexible and can adapt in real-time to the model's needs, whereas manual CL is static and predefined.
- **Scalability**: Automatic CL has the potential to scale across various tasks without extensive manual intervention, making it more suitable for diverse applications.

Static Curriculum Learning

Static curriculum learning involves a fixed, predefined sequence of training data or tasks determined before the training process begins. The sequence does not change based on the model's performance or learning progress. The key characteristics of static CL include:

- **Predefined Sequence**: The curriculum is designed and fixed before training starts.
- **Simplicity**: Implementation is straightforward as the sequence is static.

Example: In reinforcement learning, a static curriculum might involve starting with simple tasks and progressively introducing more complex ones in a fixed sequence. For instance, in training a robotic arm, the curriculum could start with tasks that require the robot to move to a specific point in an open space, then advance to tasks that involve avoiding obstacles, and finally progress to complex manipulation tasks like stacking blocks. This fixed sequence is determined before training begins and does not change based on the robot's learning progress. Just as Kumar et al.[26] applied a static curriculum learning approach in their self-paced learning framework for image classification , a similar static approach can structure the learning process for RL tasks to facilitate incremental learning.

Advantages:

- Simple to implement and understand.

- Ensures a structured learning progression.
- Can leverage domain expertise to design effective curricula.

Challenges:

- Lacks adaptability to the model's real-time learning needs.
- May not be optimal for all datasets or tasks.
- Fixed sequence might not cater to the individual learning pace of different models.

Dynamic Curriculum Learning

Dynamic curriculum learning, on the other hand, adapts the sequence of training data or tasks based on the model's performance and learning progress. This approach involves continuous assessment and adjustment of the curriculum during training.

- **Adaptive Sequence**: The curriculum evolves in response to the model's performance metrics.
- **Complexity**: Implementation involves sophisticated algorithms to dynamically adjust the training data sequence.

Example: In reinforcement learning, dynamic curriculum learning can be implemented by adjusting the difficulty of tasks based on the agent's success rate. Narvekar et al. (2020) demonstrated dynamic CL by adapting the task difficulty to the agent's current capabilities, which improved the efficiency of learning complex tasks [33].

Advantages:

- Tailors the learning process to the model's needs in real-time.
- Potentially more efficient as it focuses on areas where the model needs improvement.
- Can lead to faster convergence and better performance.

Challenges:

- More complex to design and implement.
- Requires real-time performance monitoring and adjustment mechanisms.
- Increased computational overhead compared to static CL.

Static vs. Dynamic

When comparing static and dynamic curriculum learning, several key distinctions emerge:

- **Flexibility**: Static CL is inflexible as it follows a predefined sequence, while dynamic CL adapts based on the model's learning progress.
- **Design Complexity**: Static CL is easier to design and implement, whereas dynamic CL involves sophisticated adaptive algorithms.
- **Efficiency**: Dynamic CL can be more efficient by focusing on the model's weaknesses and adapting in real-time, while static CL may not always provide the most optimal learning sequence.

3.2.3. ALP-GMM: A Curriculum Learning Approach for Deep RL[38]

An essential point of reference closely aligned with our research is the study conducted by Portelas et al.[38], situated within the realm of altering initial/terminal distribution and co-learning paradigms. They address the challenge of training Deep Reinforcement Learning (DRL) agents in environments where difficulty can be continuously adjusted through parameters in a automatic static way. The core issue is how to design an effective learning curriculum, which is the sequence of tasks presented to the student agent. Ideally, the curriculum should start with easier tasks and gradually increase in difficulty as the agent progresses. This allows the agent to learn foundational skills before tackling more complex challenges.

The work proposes a teacher algorithm that dynamically generates a curriculum for the student agent. The teacher doesn't have prior knowledge of the student's capabilities or the specific difficulty levels within the environment. To overcome this, the teacher relies on a concept called "Learning Progress" (LP). LP measures how much the student improves in each environment.

The key contribution of the paper is the introduction of a novel algorithm named ALP-GMM. This algorithm leverages Gaussian Mixture Models (GMMs) to model the student's LP across different environments. GMMs are statistical tools that can represent complex distributions. In this context, ALP-GMM uses GMMs to capture the distribution of learning progress across the entire parameter space of the environment.

By analyzing the GMMs, the teacher can identify regions corresponding to easy, difficult, or even unlearnable tasks for the student. This allows the teacher to strategically select environments that are within the student's current capabilities but still promote improvement. The paper compares ALP-GMM with existing algorithms and demonstrates its effectiveness in personalizing learning curriculums for different student agents,

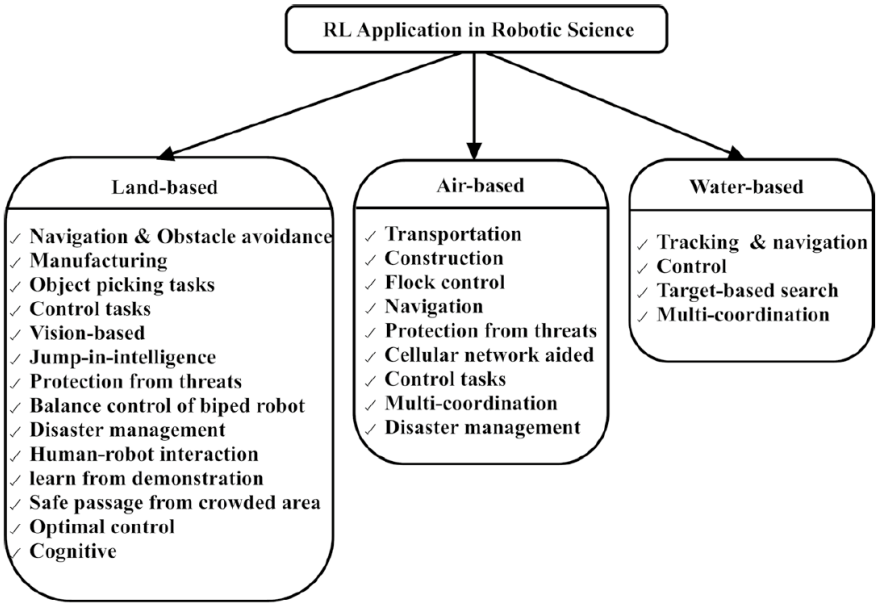


Figure 3.1: Application of RL in Robotic Science[46]

handling environments with varying ratios of learnable and unlearnable tasks, and even scaling to high-dimensional parameter spaces. Pseudo-code of this approach is presented in Algorithm 3.3.

Algorithm 3.3 ALP-GMM for Curriculum Learning[38]

Input: Initial environment parameters θ_0, maximum iterations T
Initialize Gaussian Mixture Model (GMM) with initial environment parameters θ_0
for iteration $t = 1$ to T **do**
    Sample environment parameters θ_t from GMM
    Train DRL agent in environment with parameters θ_t
    Compute learning progress $LP_t = P(\theta_t) - P(\theta_{t-1})$
    Update GMM with new sample (θ_t, LP_t)
end for

3.3. Applications of Reinforcement Learning in Robotics

Reinforcement Learning (RL) can be applied to a diverse range of tasks within robotics. A comprehensive overview of these potential applications is depicted in the accompanying figure 3.1.

In this section, a brief overview of the applications of Reinforcement Learning in Robotics is presented. Robots are classified into three categories - Water-based, Air-based, and Land-based - based on their functions. The subsequent discussion will focus on the uti-

lization of Reinforcement Learning in each type of Robot. The methods reviewed in this chapter are based on the survey[46]. We briefly introduce the methods most relevant to our problem and give an overview of the major challenges encountered. For a more detailed discussion, we refer the reader to the survey.

3.3.1. Application of Reinforcement Learning in Water-based Robots

The use of autonomous underwater robots provides significant benefits in exploring the depths of the ocean, such as studying minerals beneath the Earth's crust and observing aquatic life. Nevertheless, the challenge lies in effectively controlling these robots due to the ever-changing nature of the underwater environment[46]. Consequently, researchers have devised multiple architectural solutions for underwater robots, as documented in existing literature[7, 8, 34, 66]. The subsequent section will present various tasks and the corresponding papers related to those tasks.

Tracking and Navigation

Cables are extensively deployed in the vast ocean to facilitate communication and internet connectivity. It is imperative to monitor these cables in order to identify faults or carry out necessary maintenance. Hence, in a publication by Palomeras et al.[34], a hybrid method has been devised utilizing a natural actor-critic (NAC) algorithm in conjunction with LSTD-Q(λ) for the purpose of cable tracking mission.

Control

An adaptive control mechanism that can effectively handle parameter variation and noise is essential for AUVs. Carlucho et al.[8] implemented a combined framework of DeepRL and Actor-critic algorithm on Nessie VII AUVs to achieve adaptive control.

Target-search

At times, conducting target-based searches in the deep-sea becomes essential. Consequently, Cao et al.[7] have devised a comprehensive methodology that combines DeepRL and DQL[31] algorithms, utilizing a dual Q-network, to facilitate target-based searches. This approach has been successfully applied to Neptune-I AUVs.

Multi-Coordination

In intricate settings such as the deep-sea, effective coordination among multiple AUVs is essential. Therefore, Yu et al.[66] have devised a hierarchical architecture based on behavior, incorporating fuzzy logic and Q-learning. This methodology has been evaluated through testing in a 2vs2 water polo match.

3.3.2. Application of Reinforcement Learning in Air-based Robots

Airborne robots have been utilized in a wide range of fields such as surveillance, disaster management, real-time communication, and multi-coordination operations. However, in order to navigate through uncertain environments, such as turbulent air or loss of GPS connection, future aerial robots must possess enhanced intelligence[46]. This entails the ability to operate efficiently and autonomously by relying on their own onboard sensors. Researchers have explored different topologies to tackle the challenges encountered by these aerial vehicles, as documented in literature[9, 12, 22, 23, 27, 62, 64, 67, 68]. The subsequent section will present various tasks and the corresponding papers related to those tasks.

Transportation

Transporting time-sensitive items such as body organs on time poses a significant challenge. Faust et al.[12] introduced a model-free RL approach with continuous inputs for efficient cargo delivery via quadcopter. The quadcopter is required to transport a suspended cargo and transfer it to a ground-based robot in a swing-free manner.

Construction

Unmanned Aerial Vehicles (UAV) offer numerous benefits in the construction industry. They enhance communication, ensure safety, and conduct surveys through real-time imaging of the site. Consequently, a novel approach for construction planning utilizing quad-rotor is introduced by dos Santos et al.[9] , which combines RL and heuristic techniques.

Flock Control

Within a group, followers must adhere to the guidance of the leader. Hung and Givigi[22] proposed a method for controlling a group using Q-learning, in which the rate of learning for followers is adjusted according to Peng's Q(λ)[35].

Navigation

Navigating a UAV through a dynamic environment poses a significant and formidable challenge. In their article, Zhu et al.[68] put forth a novel approach utilizing DQN[32] algorithms for the shepherd game. This game involves the aerial robot establishing communication with ground vehicles and guiding them sequentially to secure areas while avoiding obstacles along the way.

Protection from threats

The operation of UAVs is significantly threatened by cyber-attacks. In order to address this issue, a mechanism was suggested by Xiao et al.[64] which utilizes DQL[31] to accelerate the learning process of agents in UAVs, enabling them to effectively respond to attacks even without knowledge of the attacker's model.

Control task

The control of a quadcopter in a stochastic environment poses significant challenges. Hwangbo et al.[23] and Lambert et al.[27] have introduced different approaches to address this issue. Hwangbo et al. focused on policy optimization, while Lambert et al. utilized model-based DeepRL with Model predictive control. Both studies aimed to enhance the control mechanisms of quadcopters.

Multi-coordination

The article by Zeng et al.[67] introduces an energy-efficient and continuous movement control (E^2CMC) algorithm designed for managing multiple drones. A key highlight of this algorithm is its ability to enable efficient coordination among the multiple drones.

Disaster Management

In a complicated disaster setting, UAVs can be utilized for targeted search operations. Wu et al.[62] introduced a snake algorithm based on DeeRL for locating targets such as injured individuals.

3.3.3. Application of Reinforcement Learning in Land-based Robots

The land-based autonomous vehicle is a robot that functions independently without the need for human intervention. Land-based robots have numerous practical applications in various fields, including object retrieval, navigating through congested areas without collisions, and manufacturing sites. However, they encounter certain obstacles such as

uncertainties in the real-world environment and disturbances in feedback signals received from vision-based sensors. The subsequent section will present various tasks and the corresponding papers related to those tasks.

Navigation and obstacle avoidance

Navigating a robot through a crowded area and determining the best route to avoid collisions can be quite challenging. Various learning methods have been suggested in academic research (Xu et al.[65]; Whitbrook et al.[60]). It is crucial to be able to guide a mobile robot effectively in intricate surroundings. Consequently, Juang and Hsu[24] introduced a Q-learning approach inspired by ant-optimization in their publication. Additionally, Lv et al.[30] proposed a policy network based on a deep Q-network for path planning in a dynamic environment.

Manufacturing

Robots can effectively replace human operators in dangerous environments such as manufacturing sites with high temperatures. Tzafestas and Rigatos [55] have suggested a method for polishing metal surfaces, which utilizes sliding mode control and a fuzzy controller. The controller parameters are adjusted using reinforcement learning. Specifically, the reinforcement learning method employed is a fuzzy Q-learning algorithm, which combines the principles of fuzzy logic with Q-learning. This approach enables the system to learn and adapt the control policies dynamically by optimizing the fuzzy controller parameters based on feedback from the environment, resulting in improved performance in uncertain and variable conditions.

Object picking tasks

In both industrial and domestic settings, robots are required to perform tasks such as picking up objects from one location and transferring them to another. Breyer et al.[6] introduced a method that utilizes Deep Reinforcement Learning to train a 7 degrees of freedom(DoF) arm of ABB Yumi. This approach enables the robot to learn effective strategies for tasks involving grasping, reaching, and lifting. Notably, the control mechanism operates within a continuous action space.

Control tasks

Controlling a system with higher degrees of freedom is a highly intricate endeavor. A 7-DOF simulated robot was successfully controlled for reaching tasks using model-free RL, as demonstrated by Stulp et al.[48]. Additionally, Fu and Chen[15] developed a controller

for a 32-DOF humanoid robotic system for stair climbing based on policy gradient RL. Also, balancing a robot on a rotating frame poses a significant challenge. Therefore, Xi et al.[63] proposed the integration of model-based RL with model-free RL for the balance control of a biped robot on a rotating frame with an unknown velocity. Huang et al.[21] proposed a batch reinforcement learning algorithm with parameters, utilizing the actor-critic framework. This has been specifically developed to achieve optimal control of autonomous land vehicles (ALVs). In order to enhance efficiency, a least-square batch updating rule has been incorporated into the algorithm.

Vision based

A Controller for wheeled mobile robots has been designed using robust vision-based techniques combined with Q-learning as demonstrated by Wang et al.[58]. The vision sensor plays a crucial role in providing feedback to the controller. In a separate study by Gottipati et al.[18], a deep active localization system was developed using DeepRL and an end-to-end differential method. This system focuses on perception and planning in intricate environments such as mazes. The end-to-end differentiable method is utilized for training agents in simulations, followed by transferring the trained model directly to the real robot without requiring any further adjustments. The system is structured around two main modules: (1) Convolutional Neural Network (CNN) for perception and (2) DeepRL for planning.

Cognitive

To enable cognitive abilities similar to human decision-making, Wang et al.[57] proposed a multi-task learning approach. This approach trains a non-linear feedback policy that enhances the robot's autonomy by allowing it to handle multiple tasks more effectively.

3.3.4. Navigating Constraints: The ATACOM Approach

Liu et al.[29] delve into the intersection of reinforcement learning and robotics, with a specific emphasis on tasks conducted under stringent physical constraints. Robotics often encounters challenges where actions must be executed within predefined boundaries to ensure safety or achieve specific objectives. These constraints can range from mechanical limitations to environmental factors. The work investigates how RL algorithms can effectively address these constraints by operating within what the authors term the 'constraint manifold.' This manifold represents the multidimensional space defined by the various constraints imposed on the robotic system. By navigating this manifold, RL algorithms aim to learn optimal policies that adhere to the constraints while achieving

desired tasks efficiently. This study is crucial for advancing robotics in real-world applications where strict adherence to constraints is essential. By integrating RL techniques with the constraint manifold framework, the research contributes to enhancing the autonomy, adaptability, and safety of robotic systems in complex and dynamic environments. Pseudo-code of this approach is represented in Algorithm 3.4.

Algorithm 3.4 ATACOM for Robot Reinforcement Learning[29]

Input: Initial policy π_0, maximum iterations T
Initialize constraint manifold $\mathcal{M}$ and tangent space $\mathcal{TM}$
for iteration $t = 1$ to T **do**
    Sample action a_t from policy π_t on $\mathcal{TM}$
    Apply action a_t and observe new state s_{t+1}
    Project s_{t+1} onto $\mathcal{M}$ if necessary
    Compute reward r_t based on state s_{t+1} and constraints satisfaction
    Update policy π_{t+1} using reinforcement learning algorithm
end for

3.3.5. Final Discussion on Related Work

Our research aligns with the domain of Land-based Robot Control, focusing on both 3-DOF and 7-DOF robotic arms. In a crucial aspect of our experimental setup, we employed ATACOM [29] as a foundational low-level controller. This system facilitates the translation of the agents' actions into precise joint velocities and positions, allowing us to focus on the task space of end-effector positions and manage the constraints via ATACOM.

In examining the applications of reinforcement learning (RL) in robotics across water-based, air-based, and land-based platforms, several key challenges and methods have emerged.

In water-based robotics, the dynamic and often unpredictable underwater environment poses significant challenges for control and navigation. Researchers have addressed these challenges through various methods, including hybrid algorithms combining RL with other techniques like actor-critic algorithms and deep RL. These methods have been applied to tasks such as tracking and navigation, adaptive control, target-search, and multi-coordination.

Air-based robots face obstacles like turbulent air and GPS signal loss, demanding enhanced intelligence for efficient and autonomous operation. RL methods have been pivotal in addressing these challenges, with applications ranging from transportation and construction to flock control and protection from threats. Researchers have employed diverse

RL algorithms such as DQN and actor-critic approaches to tackle navigation, control, and multi-coordination tasks.

In land-based robotics, challenges include navigating crowded environments, object manipulation, and control tasks in systems with high degrees of freedom. RL techniques have been instrumental in addressing these challenges, with applications spanning navigation and obstacle avoidance, manufacturing, object picking tasks, and control tasks. Various RL approaches, including Q-learning, deep RL, and model-based RL, have been utilized to achieve effective navigation, control, and perception in land-based robotic systems.

Furthermore, the recent work on navigating constraints, exemplified by the ATACOM approach, underscores the importance of addressing physical constraints in robotic tasks. By integrating RL with the constraint manifold framework, researchers aim to enhance the autonomy, adaptability, and safety of robotic systems operating under stringent constraints.

Comparatively, our research in land-based robot control, focusing on both 3-DOF and 7-DOF robotic arms, aligns with these broader challenges and methods. Leveraging the ATACOM system as a foundational low-level controller, we aim to address similar challenges in managing constraints while achieving desired tasks efficiently.

4 | Robot Air Hockey Challenge

The Air Hockey Challenge is an advanced competition in robot learning, pushing participants to develop autonomous air hockey-playing agents. It focuses on overcoming real-world challenges like environmental disturbances, observation noise, and physical constraints. Participants must develop robust agents capable of real-time decision-making and adapting to dynamic environments. The competition comprises simulated stages and a final real-world deployment, aiming to bridge the sim-to-real gap. In this chapter, we will introduce the air hockey game and explore the Air Hockey Challenge. This will include a detailed look at the rules and dynamics of air hockey, followed by an overview of the competition.

4.1. What is Air Hockey

Air hockey is a fast-paced table game played between two players who use handheld mallets to hit a lightweight puck across a smooth, low-friction playing surface with the goal of scoring in the opponent's goal. The game is typically played on a specially designed table that produces a cushion of air to minimize friction, allowing the puck to glide quickly and smoothly. The table is bordered by raised edges to keep the puck in play, and each end has a slot-like goal. Players use their mallets to strike the puck, aiming to score goals while defending their own goal area. Air hockey requires quick reflexes, strategic planning, and precise control, making it an engaging and competitive game.

4.2. Challenge Motivation

The motivation behind the Air Hockey Challenge is to advance the development of robotic and AI technologies by focusing on a highly dynamic and real-time task. The challenge serves as a benchmark for testing and improving robotic capabilities in a controlled yet competitive environment. It specifically aims to address key issues such as the simulation-to-reality gap, the need for robust and efficient planning under real-world constraints, and the application of machine learning techniques to robotics. By engaging

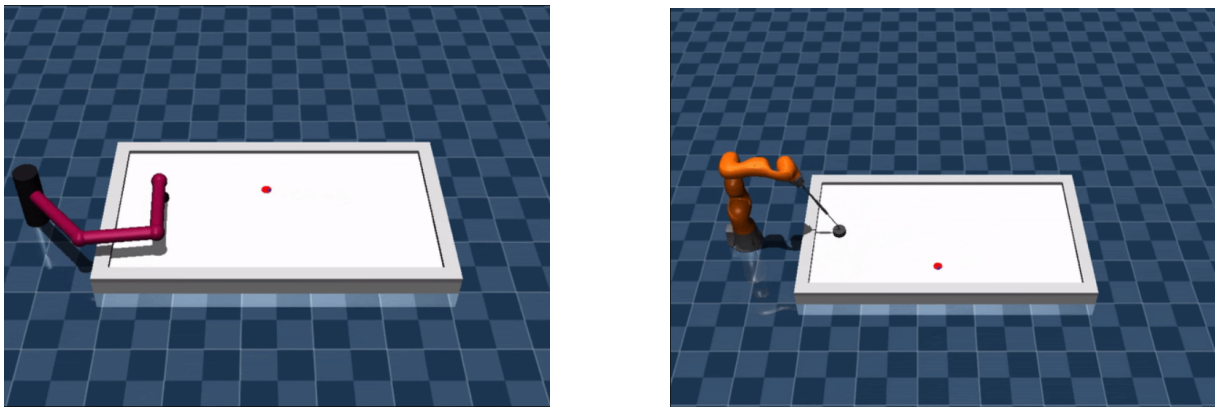
Figure 4.1: A vibrant air hockey table setup, showcasing the classic mallets and puck used in the fast-paced, competitive game. The smooth, low-friction surface is designed to allow the puck to glide effortlessly, creating a dynamic and exciting playing experience.

participants in designing algorithms that can handle the fast-paced and unpredictable nature of air hockey, the challenge promotes innovation and fosters the integration of advanced AI into physical systems[1].

This competition also encourages collaboration between various fields, including robotics, machine learning, and control systems, pushing the boundaries of what autonomous agents can achieve in real-world scenarios. Through tasks that range from simple puck handling to full-fledged matches, the challenge aims to create agents that can operate seamlessly in both simulated and actual environments, thus bridging the gap between theoretical research and practical application[1].

4.3. Challenge Organization

The Air Hockey Challenge is structured into three main phases: the Warm-Up Phase, the Qualifying Phase, and the Tournament Phase. Each phase involves distinct tasks and objectives aimed at testing the capabilities of robot agents in playing air hockey under varying conditions.



(a) (b)

Figure 4.2: (a) Example of the Warm-Up Phase in the Air Hockey Challenge, featuring a 3-DoF robot practicing defensive maneuvers. (b) Example from the Qualifying Phase, showcasing a 7-DoF KUKA iiwa14 LBR robot in defensive scenario. These phases are designed to progressively test and enhance the participants' robotic systems.

4.3.1. Warm-Up Phase

The Warm-Up Phase ran from February 20th to June 4th and involves a simplified environment designed to help participants familiarize themselves with the challenge tasks and the provided API. During this phase, participants worked with a 3-degree-of-freedom (3-DoF) robot in an ideal environment without disturbances. The focus was on basic tasks such as hitting the puck to score goals and defending against incoming pucks. An illustration of this phase is provided in Figure 4.2a.

4.3.2. Qualifying Phase

The Qualifying Phase occurred from June 5th to August 11th. In this phase, participants used a more advanced general-purpose robot, the KUKA iiwa14 LBR. This phase introduced more complex scenarios, and actuator limitations. Participants' agents were evaluated based on their performance in tasks such as hitting, defending, and preparing the puck for optimal positioning. Solutions were submitted via a docker image to a cloud server for evaluation. Teams had to submit a one-page report summarizing their approach by the end of this phase. An illustration of this phase is provided in Figure 4.2b.

4.3.3. Tournament Phase

The Tournament Phase began on August 14th and included preparation and competition periods, concluding with the final tournament in October and November. The

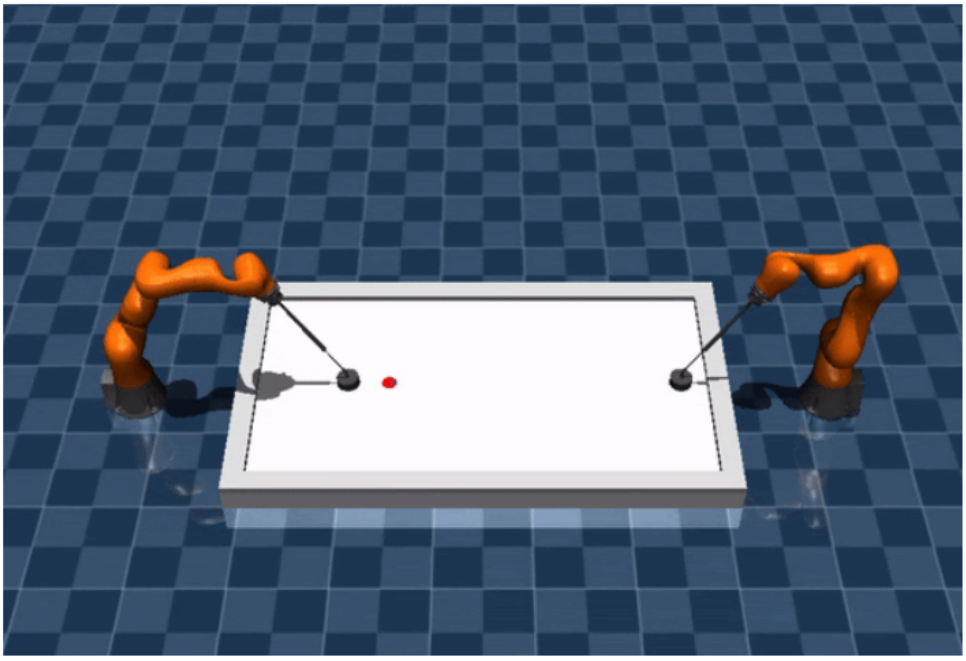


Figure 4.3: Illustration of the Tournament Phase in the Air-Hockey-Challenge. The image showcases two robotic arms actively engaged in an air hockey match, highlighting the competitive aspect of the challenge. This phase tests the performance and adaptability of the robots in a dynamic environment, crucial for evaluating their proficiency in real-time decision-making and precision control.

top-performing teams from the Qualifying Phase advance to this stage, where they developed a single agent capable of playing full games of air hockey. This phase included a double round-robin tournament where agents compete against each other under the standard game rules, with opportunities for participants to adjust and improve their agents between rounds. The final evaluation involved both simulated and real-world environments. An illustration of this phase is provided in Figure 4.3.

4.3.4. Tasks of the Phases

The Air Hockey Challenge consists of three primary tasks: Defend, Hit, and Prepare. To enhance the efficiency and competitiveness of the Air Hockey Challenge, we introduced an additional task: the Counter-Attack task. In this section, these tasks are explained.

Defend Task

The Defend task focuses on the robot's ability to prevent the puck from entering its goal. This task evaluates the robot's defensive strategies and its capability to adapt to fast-paced and unpredictable puck trajectories. The primary goal is to maximize the

number of successful blocks, demonstrating robust defensive performance under varying conditions.

Hit Task

In the Hit task, the robot is required to strike the puck towards the opponent's goal with optimal force and accuracy. This task tests the robot's offensive capabilities, including its ability to generate strategic shots and control the puck's speed and direction. The effectiveness of the robot in this task is measured by its scoring frequency and its ability. The objective is to achieve consistent and powerful hits while maintaining control over the puck's trajectory.

Prepare Task

The Prepare task involves initializing the puck near the table border. The objective is for the robot to hit the puck in such a way that it lands in a predefined area on the table, positioning it advantageously for a subsequent Hit task. This task assesses the robot's precision and planning, ensuring that it can set up optimal conditions for executing offensive plays.

Counter-Attack Task

The Counter-Attack task is similar to the Defend task but with a strategic offensive twist. Instead of merely stopping the puck, the robot aims to shoot it back towards the opponent's goal immediately after blocking it. This task evaluates the robot's ability to transition quickly from defense to offense, leveraging the momentum of the blocked puck to launch a counter-attack. We introduced this task, as it was not part of the initial challenge and no evaluation on this task was performed. The primary objective is to capitalize on defensive actions by turning them into scoring opportunities, thereby demonstrating a seamless blend of defensive resilience and offensive agility.

4.4. Challenge Framework

In this section, the framework of the challenge will be introduced. The challenge was structured around two primary elements: the Agent and the environment. The Agent had the capability to interact with the environment through the AirHockeyChallengeWrapper interface. A schema of the framework can be seen in Figure 4.4.

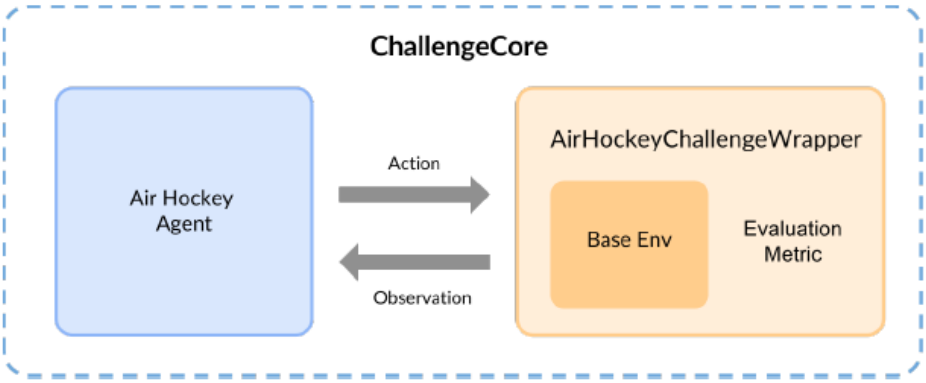


Figure 4.4: Challenge Framework[1]

4.4.1. AirHockeyChallengeWrapper

The `AirHockeyChallengeWrapper`, provided by the organizers, processes the information necessary for the challenge evaluation. This wrapper manages the interaction between the robot and the environment, ensuring that all necessary data is collected and processed effectively.

4.4.2. Control System

At a lower level, the robot is controlled by a Tracking Controller, specifically a FeedForward-PD controller. This controller is responsible for sending torque commands to the robot, as defined by the following equation:

$$\text{cmd} = M(q)\ddot{q}_d + c(q, \dot{q}) + g(q) + K_p(q_d - q) + K_d(\dot{q}_d - \dot{q}) \tag{4.1}$$

Where:

- $M(q)$ represents the inertia matrix,
- $c(q, \dot{q})$ is the Coriolis and centrifugal forces,
- $g(q)$ is the gravity compensation,
- K_p and K_d are the proportional and derivative gain matrices, respectively,
- q and $\dot{q}$ are the joint positions and velocities,
- q_d and $\dot{q}_d$ are the desired joint positions and velocities.

4.4.3. Trajectory Interpolation

A Trajectory Interpolator, by default a cubic polynomial, is used to interpolate the trajectory points between two consecutive commands. This interpolation ensures smooth motion and accurate following of the desired path. Additionally, a Joint Safety Limiter is implemented to prevent the commands from exceeding the position or velocity limits, ensuring safe operation of the robot. A schematic of this control paradigm is shown in Figure 4.5.

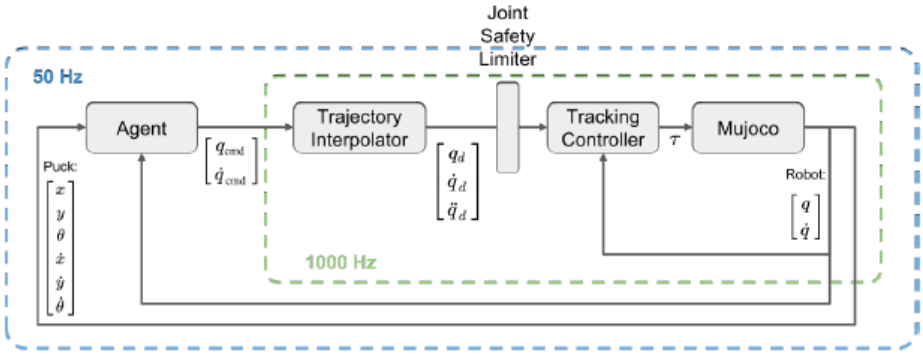


Figure 4.5: Control paradigm[1].

4.4.4. Simulation Environment

The simulation is carried out using the MuJoCo simulator. The simulation frequency is set to 1000Hz, while the control frequency is set to 50Hz. The environment provides observations that include:

- **Puck Position and Velocity:** $[x, y, z, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\psi}]$, with ψ representing the yaw angle.
- **Joints Position and Velocities:** $[q, \dot{q}]$.
- **Opponents Mallet Position:** $[x_o, y_o, z_o]$ (if the environment includes an opponent).

Puck positions and opponent's mallet positions are expressed in meters for the x, y and z coordinates, while ψ is expressed in radians. The velocities are given in meters per second and radians per second. Joint positions and velocities are expressed in radians and radians per second, respectively.

4.4.5. Desired Control Actions

The desired control actions consist of the desired joint positions and velocities during the warm-up phase. During the qualifying and tournament phases, participants can customize the desired trajectory interpolation order of the Trajectory Interpolator. The possible interpolation methods are:

- **Cubic Interpolation (3):** The action command contains the desired position and velocity, with a cubic polynomial used to interpolate intermediate steps.
- **Linear Interpolation (1):** The action command contains the desired position, with a linear polynomial used for interpolation.
- **Quadratic Interpolation (2):** The action command contains the desired position, using a quadratic function based on the previous position, velocity, and desired position.
- **Quartic Interpolation (4):** The action command contains the desired position and velocity, with a quartic function using previous and desired positions and velocities.
- **Quintic Interpolation (5):** The action command contains the desired position, velocity, and acceleration, with a quintic function computed using previous and desired states.
- **Linear Interpolation (-1):** The action command contains the desired position and velocity, with acceleration computed from the velocity derivative, useful to avoid oscillatory behaviors.
- **None:** The agent sends a complete trajectory between each action step, specifying the position, velocity, and acceleration of each joint.

This comprehensive framework ensures a robust evaluation of robotic systems in the Air-Hockey Challenge, driving advancements in control strategies and real-time robotic performance.

4.5. Robot Constraints in the Evaluation of the Air-Hockey Challenge

The evaluation of robotic systems in the Air-Hockey Challenge requires adherence to a set of constraints to ensure safety, reliability, and fairness. These constraints cover various

aspects of the robot's operation, including joint limits, velocity limits, acceleration limits, jerk limits, and operational boundaries. Below is a detailed description of the robot constraints used in the evaluation.

4.5.1. Joint Position and Velocity Limits

To prevent mechanical failure and ensure safe operation, each joint of the robot is subjected to specific position and velocity limits:

- **Joint Position Limits:** These limits define the maximum and minimum angular positions that each joint can achieve. Exceeding these limits could result in mechanical damage or loss of control. The limits are defined as:

  $$q_{\min} \leq q \leq q_{\max}$$

  Where q represents the joint position, and $q_{\min}$ and $q_{\max}$ are the minimum and maximum allowable positions, respectively.

- **Joint Velocity Limits:** These limits restrict the maximum angular velocity of each joint to prevent excessive wear and ensure precise control. The limits are defined as:

  $$\dot{q}_{\min} \leq \dot{q} \leq \dot{q}_{\max}$$

  Where $\dot{q}$ represents the joint velocity, and $\dot{q}_{\min}$ and $\dot{q}_{\max}$ are the minimum and maximum allowable velocities, respectively.

4.5.2. Acceleration and Jerk Limits

To ensure smooth and controlled movements, each joint is also subject to acceleration and jerk limits:

- **Joint Acceleration Limits:** The acceleration of each joint is constrained as follows:

  $$\ddot{q}_{\min} \leq \ddot{q} \leq \ddot{q}_{\max}$$

  Where $\ddot{q}$ represents the joint acceleration, and $\ddot{q}_{\min}$ and $\ddot{q}_{\max}$ are the minimum and maximum allowable accelerations, respectively.

- **Joint Jerk Limits:** The rate of change of acceleration (jerk) for each joint is constrained to ensure smooth control actions:

  $$\dddot{q}_{\min} \leq \dddot{q} \leq \dddot{q}_{\max}$$

Where $\dddot{q}$ represents the joint jerk, and $\dddot{q}_{\min}$ and $\dddot{q}_{\max}$ are the minimum and maximum allowable jerks, respectively.

4.5.3. Operational Boundaries

To ensure the robot operates safely and effectively in the air hockey environment, specific operational boundaries are enforced:

- **Table Interaction Limits (z-axis):** The mallet, controlled by the robot, must not penetrate the surface of the table. This constraint is enforced to prevent damage to the table and ensure realistic interactions with the puck. Additionally, the mallet is restricted from moving too far above the table to maintain effective play and avoid unrealistic scenarios.

  The position of the mallet along the z-axis must satisfy:

  $$z_{\min} \leq z \leq z_{\max}$$

  Where $z_{\min}$ is set slightly above the table surface to prevent penetration, and $z_{\max}$ is set to a height that maintains realistic play conditions.

- **Table Interaction Limits (x and y axes):** The mallet must not collide with the boundaries of the table to ensure effective play and avoid unrealistic scenarios.

  The position of the mallet along the x and y axes must satisfy:

  $$x_{\min} \leq x \leq x_{\max}$$
  $$y_{\min} \leq y \leq y_{\max}$$

  Where $x_{\min}$ and $x_{\max}$, $y_{\min}$ and $y_{\max}$ limits are set to prevent collisions with the table boundaries.

4.5.4. Real-Time Constraints

Given the dynamic nature of the air hockey game, the robot must adhere to real-time constraints to ensure timely responses to environmental changes:

- **Control Frequency:** The control system must operate at a frequency of 50Hz to ensure responsive and accurate control of the robot.
- **Simulation Frequency:** The simulation of the environment, including physics calculations and sensor updates, must run at 1000Hz to provide a high-fidelity

representation of the game dynamics.

4.6. Challenge Evaluation

The evaluations focus on the agent's ability to handle environmental disturbances, push robot capabilities, ensure safe and reasonable behavior, perform dynamic motions, adapt to the sim-to-real gap, and develop both low-level control and high-level strategies. Participants must submit reports detailing their approaches at the end of the Qualifying and Tournament phases. The challenge evaluated teams through three stages: Warm Up, Qualifying, and Tournament. Each stage involves simulated tasks and the best three teams of tournament stage deployed their agents on real robots. Participants submitted their solutions in the form of Docker images, and for each phase, there were some metrics which was computed and reported to participants.

4.6.1. Evaluation of Warm-Up Phase

During the Warm-Up phase, participants worked an ideal, disturbance-free simulation environment. The primary goals of this phase were to familiarize participants with the tasks, environments, and provided APIs, as well as to acclimate them to the evaluation pipeline. Participants were tasked with completing hit and defend challenges. The agents' success rates were then reported, but this evaluation did not impact the final rankings. Instead, it provided teams with valuable feedback to assess their agents' performance.

This phase served as a preparatory step, ensuring participants were well-prepared for the more challenging subsequent stages of the competition, where various real-world issues and disturbances would be introduced.

4.6.2. Evaluation of Qualifying Phase

The evaluation during the Qualifying phase involves assessing the performance of participants' air hockey agents in three tasks: hitting, defending, and preparing. The agents are tested using a KUKA iiwa14 LBR robot in a realistic simulation environment.

Evaluation Metrics

1. **Success Rate:** Each task is run for 1000 episodes, and the success rate is calculated based on the defined success criteria for each task.
2. **Deployability:** This metric assesses the agent's adherence to constraints such as

joint position and velocity limits, end-effector position constraints, and computation time. Penalties are assigned for violations of these constraints.

Constraint Penalties

Deployability of the agents was critically evaluated using the following metrics:

1. **End-Effector's Position Constraints:** Penalties are applied if the end-effector exceeds the table boundaries or specified height limits.
2. **Joint Position and Velocity Constraints:** Penalties are given if joint positions or velocities exceed their limits.
3. **Computation Time Constraints:** Penalties vary based on how much the computation time exceeds the allowed 0.02 seconds per step.

Deployability penalties are scored based on the severity and frequency of constraint violations. The total penalty score determines the agent's deployability category:

- **Deployable**: Penalty score ≤ 500
- **Improvable**: $500 <$ Penalty score ≤ 1500
- **Non-deployable**: Penalty score > 1500

Teams qualified for the tournament phase based on deployability results. If an agent was not deployable, the team was eliminated from the competition. Overall, the Qualifying phase aims to ensure that the agents not only achieve their tasks effectively but also adhere to constraints that ensure realistic and safe robot operation.

4.6.3. Evaluation of Tournament Phase

The evaluation during the Tournament phase focuses on the overall performance of participants' air hockey agents in competitive matches. The agents compete using a KUKA iiwa14 LBR robot in a realistic simulation environment.

Evaluation Metrics

1. **Game Performance:** Agents compete in full-length air hockey matches, with their performance evaluated based on the number of goals scored against their opponents. Each game results in one of three outcomes: a win, draw, or loss, earning the agent 3 points, 1 point, or 0 points respectively. The final rankings are determined based on the total points accumulated by each team.

2. **Deployability:** This metric evaluates the agent's compliance with constraints such as joint position and velocity limits, end-effector position constraints, and computation time, similar to the Qualifying phase. If an agent exceeds these constraints beyond the allowed threshold, it will be disqualified from the game.

Overall, the Tournament phase aims to test the agents' ability to play air hockey effectively against various opponents while ensuring they adhere to operational constraints that promote realistic and safe robot functioning

5 | Methodology

In this chapter, we present the methodology employed to tackle the challenges posed by the air hockey game environment. More specifically, the manual approaches to solve the *defend* and *counter-attack* tasks and automatic approaches to solve the *hit* task will be discussed. The foundation of this methodology lies in the conceptualization of the air hockey challenge as an MDP, providing a structured framework for understanding the dynamics of the game and formulating effective strategies. The methodology is divided into two primary tasks: first, implementation of a manual curriculum learning approach, drawing upon personal expertise and insights into the game, second, an exploration of automatic curriculum learning through a teacher-student architecture. This chapter outlines the steps taken in both avenues of inquiry, get across the methodologies, algorithms, and experimental setups utilized to address the objectives of the study.

5.1. Air Hockey as an MDP

The air hockey game, while seemingly simple in its mechanics, presents a complex and dynamic environment ripe for exploration through the lens of MDPs. In this section, we delve into the conceptualization of air hockey as an MDP, get across the key components and dynamics that govern player interactions within this framework. By modeling the game as an MDP, we aim to capture the stochastic nature of gameplay, the influence of player actions on future states, and the overall goal of maximizing cumulative rewards. This section present the foundation upon which subsequent methodological approaches are built, providing a comprehensive understanding of the underlying structure of the air hockey challenge.

Furthermore, we introduce two distinct methodologies for modeling the MDP: one centered around the end-effector as the agent's control mechanism and informational focus, and another method wherein the joints themselves act as the primary method of control and observation for the agent. These approaches offer unique perspectives on the game dynamics, highlighting the importance of agent control and knowledge representation in navigating the complexities of the air hockey environment.

Additionally, the environment was configured to receive the next joint positions and velocities as action, and subsequently, the framework planned to reach these positions using various possible interpolations, such as linear interpolation, cubic interpolation, and others. This setup enhances the adaptability and responsiveness of the agent's control strategy, enabling it to effectively execute planned movements while maintaining smooth and accurate motion during gameplay. However, it's notable that the reward functions of both approaches vary depending on the specific setting and will be thoroughly discussed in each corresponding section.

5.1.1. End-Effector-Centric Modeling

State Space

When employing an end-effector-centric modeling approach within the framework of MDPs, the state space is defined by the positional coordinates and dynamics of the end-effector. In this paradigm, the agent's actions are directed towards controlling the end-effector's movements, without direct access to the underlying joint configurations or robot dynamics. Consequently, the state space encapsulates the spatial coordinates, and velocity of the end-effector, acting as the primary interface for the agent's decision-making process. When making decisions, the agent also incorporates information about the position and velocity of the puck. By focusing on the end-effector, the agent operates within a simplified representation of the robot's dynamics, abstracting away complexities associated with joint movements and internal mechanics. Additionally, a flag indicating whether a collision happened between the end-effector and the puck was added to the observation. The resulting state space is then scaled between -1 and 1. This streamlined perspective facilitates a more intuitive understanding of the air hockey environment, enabling the agent to effectively navigate the game space and optimize its strategies towards achieving desired outcomes.

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

Action Space

In the context of employing an end-effector-centric modeling approach, the action space is characterized by the agent's output of the desired next position of the end-effector. Rather than directly controlling the individual joint movements or exerting forces on specific parts of the robot, the agent's actions consist of specifying the desired spatial coordinates for the end-effector to move towards. These coordinates cover the translational

aspects of the end-effector's motion on the air hockey surface. By generating such output, the agent guides the end-effector's trajectory in alignment with its strategic objectives, whether it be offensive maneuvers to score goals or defensive tactics to block opponent shots. The action space thus represents a pivotal aspect of the decision-making process, enabling the agent to exert precise control over the end-effector's movements in pursuit of optimal performance within the air hockey environment. Additionally, when using end-effector position, for converting it to joint position and velocities, an optimization-based approach provided by the competition team was utilized. This optimization-based method ensures the seamless transformation of end-effector positions into corresponding joint configurations and velocities, facilitating efficient and accurate control of the robotic arm during gameplay.

5.1.2. Joint-Centric Modeling

State Space

In the joint-centric modeling approach, the state space is defined by the positional and velocity states of the robot's joints, providing a comprehensive representation of the robot's configuration and dynamics. In addition to this joint-level data, the agent also perceives the positional and velocity states of both the end-effector and the puck, enriching its understanding of the game dynamics. By integrating these multiple sources of information, the agent gains a better view of the air hockey environment, facilitating informed decision-making and precise control over the robot's movements. This expanded state space allows the agent to develop strategies that use the relationship between joint positions, end-effector movements, and puck behavior, improving its ability to handle the challenges of the air hockey game effectively.

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

Action Space

In the context of employing the joint-centric modeling approach, the action space is characterized by the agent's output of joint accelerations. Instead of directly specifying the next position of the end-effector, as in the end-effector-centric approach, the agent in this paradigm determines the accelerations to be applied to each individual joint of the robot arm. These accelerations govern the rate of change in joint velocities, thereby influencing the trajectory and dynamics of the end-effector's motion on the air hockey surface. By outputting joint accelerations, the agent exercises fine-grained control over

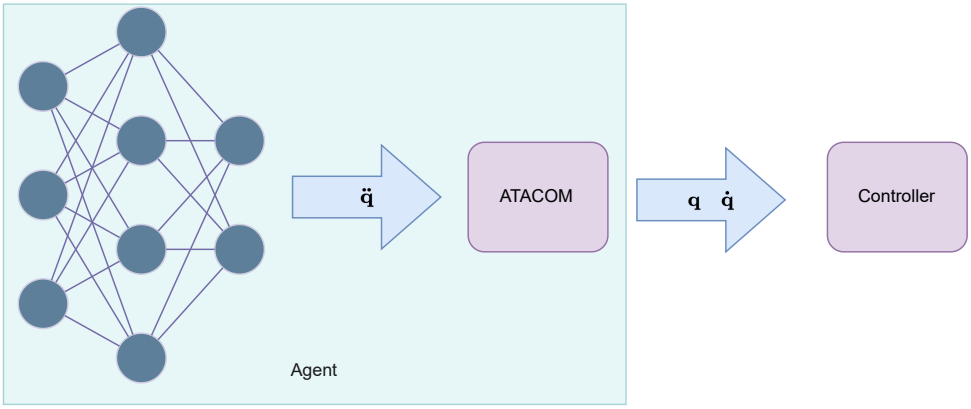


Figure 5.1: Overall architecture of the reinforcement learning (RL) agent. The agent utilizes a neural network to process inputs and output joint accelerations (represented as $\ddot{q}$). These accelerations are then converted into joint positions and velocities (represented as q and $\dot{q}$, respectively) by the ATACOM module. The controller uses these joint positions and velocities to perform the desired actions.

the robot's movements, adjusting the speed and direction of each joint's motion to achieve desired gameplay objectives. This action space allows the agent to dynamically adapt its control strategy in response to evolving game conditions, optimizing its performance and responsiveness within the air hockey environment. Furthermore, as shown in Figure 5.1 for converting the joint accelerations into the joint positions and velocities, ATACOM was utilized [29]. ATACOM ensures that the actions do not violate the robot's constraints, providing a mechanism for translating the agent's specified accelerations into feasible joint motions while maintaining stability and safety during gameplay.

5.2. Approach to Tackling the Challenge: DRL and Hierarchical Strategies

To tackle the challenge effectively, we structured our approach using DRL for the initial stages. We began by addressing foundational tasks individually, such as hitting and defending, which are critical components of the air hockey game. Each task was evaluated separately to ensure that our agents could perform these fundamental actions reliably and efficiently. This step-by-step method allowed us to build robust base policies that could handle specific aspects of the game with precision.

For the full air hockey game, we implemented a hierarchical strategy that integrated these base policies into a cohesive framework. This hierarchical approach involved a higher-level decision-making mechanism that determined the moments to switch between

different base policies during gameplay. By doing so, our agent could dynamically adapt to various in-game scenarios, making real-time decisions on whether to prioritize offensive or defensive actions based on the current state of the game.

This combined method of using DRL for base tasks and a hierarchical approach for overall game strategy was essential for managing the complexities of a full air hockey match. Specifically, we utilized SAC[19], and trained our agents using a manual curriculum. This curriculum involved progressively more challenging tasks, ensuring that our agents developed the necessary skills incrementally.

Additionally, the hierarchical approach with a switching mechanism was crucial for achieving a high level of performance. This approach allowed our agents to seamlessly transition between different strategies, optimizing their actions in response to the evolving dynamics of the game.

5.3. Manual Curriculum Learning Approach

In reinforcement learning, manually designing curricula is a strategic method to guide agent learning in complex environments. This section explores the methodologies and strategies used in manually designed curriculum learning to effectively support the agent's learning process. The focus is on its application in two critical tasks: *defend* and *counter-attack*. Through careful planning and structuring of learning experiences, the manual curriculum approach aims to equip the agent with the necessary skills to tackle these specific challenges within the air hockey domain. By outlining this approach and its application in defense and counter-attack tasks, this section provides a basis for understanding the efficacy and impact of manual curriculum learning on agent performance.

5.3.1. Initial DRL Strategy and the Necessity for a Curriculum Approach

Our training approach began with the SAC algorithm [19], leveraging two distinct reward functions. Before the end-effector made contact with the puck, we used the reward function defined in Section 5.1. Upon contact, the reward function switched to the one defined in Section 5.2 for precisely one step. Subsequent interactions with the environment were hidden from the agent, as they did not provide useful feedback on the agent's performance. Initially, we employed an end-effector-centric modeling approach. In the 3-DOF setting, this method performed well, achieving a success rate of over 0.7. However, when we switched to the 7-DOF setting, the problem of constraint violations arose,

causing the agent to be classified as undeployable.

To address these issues, we transitioned to a joint-centric approach, utilizing the ATACOM [29] to handle the constraint violations effectively. With this revised approach, our training resumed using SAC with the same reward functions. We anticipated that SAC would enable our agents to learn effective strategies for the air hockey tasks through trial and error, guided by these rewards.

$$r_t(s,a) = \|\mathbf{p}_{end-effector} - \mathbf{p}_{puck}\| \tag{5.1}$$

However, despite the theoretical strengths of SAC, we encountered significant limitations in practice. The agent's progress plateaued early, and it was unable to achieve the high performance levels we aimed for. This indicated that the tasks were too complex for the agent to learn effectively from scratch, even with the sophisticated learning capabilities of SAC. The agent struggled to generalize from the limited successful experiences it encountered, leading to suboptimal policy development.

$$r_t(s,a) = 1000 - 300 \cdot \|\dot{\mathbf{p}}_{puck}\| \tag{5.2}$$

Recognizing these challenges, we decided to adopt a curriculum learning approach. This method allows the agent to build foundational skills incrementally, improving its learning efficiency and overall performance.

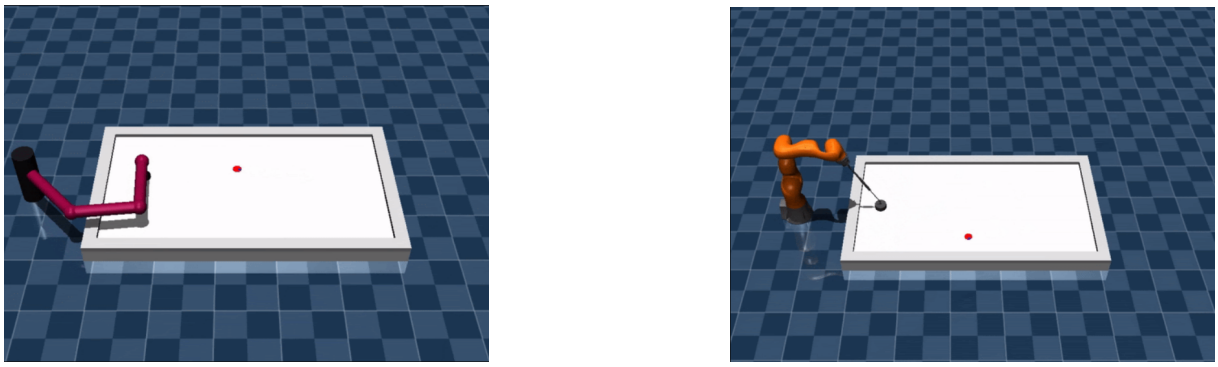
5.3.2. Approach to Manual Curriculum Learning in Air Hockey Defend

Defend Task

As explained in Section 4.3.4,The defend task involved a single robot on the field, with the puck initially launched from the opponent's side toward ours, setting the stage for strategic defensive maneuvers. The ultimate objective of the defend task was to stop the puck in our own side. To visually illustrate the defend task, Figure 5.2 depicts an example of a defend scenario in both 3-DoF and 7-DoF configurations, highlighting the initial conditions and dynamics of the defensive challenge.

Factors Influencing Task Complexity in the Defend Task

To regulate the task's complexity, three primary factors were controlled: the puck's initial position, its linear velocity, and its direction upon initialization.



(a) (b)

Figure 5.2: an example of defend scenario in both 3-DoF(a) and 7-DoF(b), where the agent strategically positions itself to intercept the puck (depicted by the red circle) launched from the opponent's side, showcasing the initial conditions and dynamics of the defensive challenge.

The initial position of the puck is a key factor in determining task difficulty. As the puck's starting position moves closer to the midpoint of the field, the challenge increases, requiring greater defensive skill from the agent.

The linear velocity of the puck is another crucial factor in determining task complexity. Increasing the puck's velocity proportionally raises the task's difficulty, requiring quicker reflexes and more precise defensive actions from the agent.

Additionally, the initial direction of the puck significantly influences task complexity. Increasing the range of launch angles makes bounces and paths more unpredictable, making defense harder.

Reward Function

In this task, each episode starts with the puck moving towards our goal. The agent incurs a substantial penalty of -2000 if the puck reaches our goal. Successful interaction with the puck earns the agent a reward inversely proportional to the puck's velocity norm, as described in Equation 5.2. If the puck stops in the predefined area, the agent receives a reward of +2000. Each episode concludes immediately after the interaction with the puck. This setup encourages the agent to engage with the puck and slow it down while defending the goal.

Training

To facilitate the training process using manual curriculum learning, the aforementioned factors were systematically manipulated to create three distinct classes of environments:

easy, medium, and hard tasks. Each class presented varying degrees of complexity, enabling the agent to progressively adapt and refine its defensive strategies.

The approach followed for manual curriculum learning involved dynamically adjusting the task complexity based on the agent's performance. When the agent encountered states where its training performance was suboptimal, the complexity of the task was increased from one class to the next using a sigmoid function, as shown in Equation 5.3. This function enabled a smooth and gradual progression between different levels of complexity, with transitions divided into 1000 steps. By iteratively adjusting the task complexity in response to the agent's learning progress, This approach helped the agent gradually adjust to more difficult defensive situations. Ultimately, this iterative curriculum design process enhanced the agent's adaptability and performance within the air hockey domain.

$$c(epoch_i) = (c_{target} - c_{init}) \cdot \sigma(-4.5 \cdot (epoch_i - epoch_{start}) \cdot step) \tag{5.3}$$

Where:

- c represents the bounds for the linear velocity, position, and initial direction of the puck.
- c_{target} denotes the target bound.
- c_{init} denotes the initial bound.
- $epoch_i$ is the current epoch.
- $epoch_{start}$ is the epoch at which the adjustment begins.
- $step$ determines the rate of change.
- $\sigma(x)$ is the sigmoid function, defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{5.4}$$

This equation adjusts the bounds c over time using a sigmoid function, which provides a smooth transition from the initial bound c_{init} to the target bound c_{target}. As the current epoch ($epoch_i$) increases, the sigmoid function gradually shifts the bounds from c_{init} to c_{target}, modulating the task difficulty over the learning process.

5.3.3. Approach to Manual Curriculum Learning in Air Hockey Counter-Attack

Counter-Attack Task

As explained in Section 4.3.4, the counter-attack task is similar to the defend task in many aspects, with the key difference being its ultimate goal. Instead of only focusing on defensive actions, the agent must strategically transit from defense to offense by shooting the puck towards the opponent's goal. This shift in focus introduces a dynamic element to the agent's decision-making process, as it must block incoming shots and also identify opportunities to launch counter-attacks.

Factors Influencing Task Complexity in the Counter-Attack Task

For the counter-attack task, the factors influencing task complexity remain consistent with those outlined for the defend task. However, an additional dimension was introduced to further test the agent's capabilities. In selected experiments, an additional robot acting as an opponent was incorporated into the environment, capable of executing defensive maneuvers to protect its goal.

The objective went beyond simple counter-attack training; it aimed to enhance the agent's abilities in scenarios involving an opponent. By confronting the agent with a dynamic opponent, the training environment mirrored real-world challenges more closely, fostering the development of robust defensive and offensive strategies.

Initially, the goal was to enable the agent to effectively replace the simulated opponent with increasingly skilled adversaries as its proficiency improved. This gradual increase aimed to push the agent's abilities to their limits, fostering continuous growth and adaptation in response to evolving challenges within the air hockey domain.

Reward Function

In this task, each episode begins with the puck moving towards our goal. The agent receives a substantial penalty (-100) if the puck reaches our goal. Successful interaction with the puck earns the agent a positively proportional reward based on the cube of the puck's velocity norm, as shown in Equation 5.5.

$$r_t(s, a) = 100 + 50 \cdot \|\dot{\mathbf{p}}_{puck}\|^3 \tag{5.5}$$

After this interaction, all future states and actions are excluded from the replay buffer,

as they do not provide useful learning information. The agent is only waiting to receive the last action's reward (interaction with the puck). If the puck hits the goal, the agent receives an additional substantial reward (+2000). Otherwise, a penalty based on the distance between the point where the puck hits the right side of the table and the center of the goal is applied, as shown in Equation 5.6. This setup encourages the agent to interact with the puck and increase its speed, causing it to hit one of the table's sides before reaching the goal, making it even harder for opponent to defend.

$$r_t(s, a) = -100 \cdot \|\mathbf{P}_{puck} - \mathbf{P}_{goal}\| \tag{5.6}$$

Training

To initiate training for the counter-attack task, the latest defend agent was employed as the starting point. Leveraging the learned defensive strategies, the agent was transitioned into a new task environment where the primary objective shifted towards executing effective counter-attacks. While the underlying MDP remained unchanged from the defend task, changes were made to the reward function to encourage offensive moves and successful shots on the opponent's goal..

It is notable that the counter-attack agent can be viewed as a continuation of the curriculum learning process initiated with the defend task. Building upon the foundation laid by the best defend agents, the counter-attack agent represents further refinement and adaptation of defensive strategies into offensive actions. By iteratively fine-tuning its capabilities through successive curriculum learning steps, the agent gradually expands its skills and adapts to the evolving challenges within the air hockey domain.

Additionally, similar to the defend task, the sigmoid function, as shown in Equation 5.3, is used for transitioning between complexities. This method ensures a smooth and gradual increase in task difficulty, facilitating the agent's learning process.

5.4. Automatic Curriculum Learning

In the field of reinforcement learning, the automatic design of curricula represents an innovative approach to autonomously guide agent learning in complex environments. This section examines the complexities of automatically generated curriculum learning, detailing the algorithms and strategies used to dynamically support the agent's learning process. Particular emphasis is placed on its application to the *hit task*. Through adaptive planning and real-time structuring of learning experiences, the automatic curriculum

approach aims to provide the agent with optimized skills to address this specific challenge within the air hockey domain. By outlining the details of this approach and its application in the hit task, this section lays the foundation for a thorough examination of the effectiveness and impact of automatic curriculum learning on agent performance.

The hit task posed significant challenges when relying solely on sparse rewards. While reward shaping techniques could potentially address these challenges, they also risk limiting the agent's autonomy. Given that the hit task was the weakest base policy we trained, improving it was crucial for enhancing the overall agent's performance. Therefore, in this section, we propose a novel teacher-student approach to develop an automatic curriculum that helps the agent effectively discover sparse rewards. This focused effort on the hit task was essential to sustain the agent's proficiency and ensure a more balanced and capable performance across all tasks.

5.4.1. Hit Task

As explained in Section 4.3.4, the hit task involves the agent's ability to accurately shoot the puck towards a designated target area. This task tests the agent's precision, control, and strategic planning as it must account for factors such as puck speed, angle, and timing to achieve successful hits. The agent learns to optimize its movements and force application to ensure the puck moves in the desired trajectory, mimicking the skillful maneuvers seen in professional air hockey gameplay. Mastery of the hit task is crucial as it forms the foundation for more complex behaviors in the game, contributing significantly to the agent's overall performance and competitiveness in the air hockey challenge[1].

5.4.2. Reward Function

To define the reward function, we must first identify the reward features.

Definition 5.1 (Reward Features(F))**.** *Reward features consist of specific micro tasks, such as the distance between the puck and the mallet, which are relevant and beneficial for accomplishing the main target task, including the main target task itself.*

All reward features provide dense rewards, except for the target task, which offers sparse rewards. Each reward feature has an importance coefficient.

Definition 5.2 (Importance Coefficient(C))**.** *The importance coefficient is a factor that signifies the significance of a specific reward feature when calculating the overall reward. Additionally, the importance coefficients must hold the following criterion: the second*

norm of the importance coefficients must sum to one. Mathematically, this can be expressed as:

$$\sum_{i=1}^{n} c_i^2 = 1 \tag{5.7}$$

Reward function in this approach will be calculated based on Equation 5.8.

$$r_t(s,a) = \sum_i f_{i,t}(s,a) \times c_{i,t} \tag{5.8}$$

Where:

- i is an index that iterates over all possible reward features.
- t is the time step.
- $f_{i,t}$ is the i'th reward feature calculated at time t.
- $c_{i,t}$ is the i'th importance coefficient.

5.4.3. Teacher

In a teacher-student architecture, the teacher is responsible for outputting the importance coefficients $\mathbf{C}$ in a manner that maximizes a specific criterion. This criterion can be either the learning progress of the student or the reward obtained by the student directly.

Definition 5.3 (Learning Progress)**.** *Learning progress in this context is calculated as follows: when the teacher outputs the importance coefficients* $\mathbf{C}$*, the student is trained using these coefficients for a determined number of steps. The average reward for* $\mathbf{C}$ *is then calculated, denoted as* r*. Next, the nearest neighbor to* $\mathbf{C}$*, denoted as* $\mathbf{C}'$*, is found. The reward of the student when it was using* $\mathbf{C}'$*, denoted as* r'*, is then considered. The learning progress is defined as the difference between these rewards:*

$$\textit{Learning Progress} = r - r' \tag{5.9}$$

State Space

The state of the teacher is determined only by the learning progress or the reward it observes from the student. This scenario effectively frames the teacher's task as a multi-armed bandit problem. In this context, each arm represents a different set of importance coefficients $\mathbf{C}$ that the teacher can sample. The teacher's objective is to identify the $\mathbf{C}$ that maximizes the chosen criterion, either learning progress or the student's reward, based on the feedback received from the student.

This approach allows the teacher to adaptively optimize its strategy to enhance the student's performance over time. However, it is important to note that this problem is not a simple multi-armed bandit problem, as the problem for the teacher changes over time. Thus, the teacher is dealing with a non-stationary multi-armed bandit scenario.

Action Space

In each iteration, the teacher must choose a new set of importance coefficients. Each importance coefficient corresponds to a reward feature, so the number of importance coefficients equals the number of reward features being used. There might be instances where the sampled $\mathbf{C}$ does not meet the constraint specified in Equation 5.7. To address this issue, we translate the problem of sampling $\mathbf{C}$ directly to sampling $\boldsymbol{\theta}$. These $\boldsymbol{\theta}$ values represent hyperspherical coordinates, where each θ within the bounds of 0 to $\frac{\pi}{2}$ is valid. This approach ensures that the resulting Cartesian coordinates inherently satisfy the constraint in Equation 5.7. Moreover, the length of $\boldsymbol{\theta}$ is exactly one less than that of $\mathbf{C}$, simplifying the teacher's action space.

We will now introduce the approaches and algorithms utilized in this thesis.

Learning Progress Upper Confidence Bound(LP-UCB)

In this approach, to simplify the action space, we discretized the teacher's action space, represented by the $\boldsymbol{\theta}$ values, into a finite set of values. This transformation simplifies the continuous space of $\boldsymbol{\theta}$, significantly reducing the complexity for the teacher. We then address the problem using an Upper Confidence Bound (UCB)[3] method. The teacher works based on the learning progress of the student as defined in Definition 5.3.

Reward Upper Confidence Bound(Reward-UCB)

This approach is similar to LP-UCB, but with a key difference: the teacher directly optimizes the reward obtained from the student's interactions with the environment.

Learning Progress Gaussian Mixture Model(LP-GMM)

In this approach, the entire action space is treated as continuous. Inspired by the work Portelas et al.[38], the teacher utilizes a Gaussian Mixture Model (GMM) to guide the learning process.

A Gaussian Mixture Model works by representing a distribution as a combination of multiple Gaussian distributions, each defined by its mean and variance. This allows for capturing complex data structures and modeling a variety of distributions. The GMM assigns a probability to each data point, indicating the likelihood that the point belongs to each of the Gaussian components. This probabilistic framework enables the model to adapt and refine its understanding of the data over time.

In this context, the teacher uses the GMM to monitor and assess learning progress, as defined in Definition 5.3.

5.4.4. Approach to Automatic Curriculum Learning in Air Hockey Hit

In this section, the automatic curriculum learning in hit task is explained.

Student State Space

The state space of the student is consistent with the descriptions provided in Sections 5.1.1 and 5.1.2, with the addition of importance coefficients as part of its state. Therefore, the overall state of the student is defined as follows: For joint-centric modeling:

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

For end-effector-centric modeling:

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

Here, $\mathbf{p}_{\text{puck}}$ and $\dot{\mathbf{p}}_{\text{puck}}$ represent the position and velocity of the puck, $\mathbf{q}$ and $\dot{\mathbf{q}}$ denote the joint positions and velocities, $\mathbf{p}_{\text{ee}}$ and $\dot{\mathbf{p}}_{\text{ee}}$ indicate the position and velocity of the end-effector, $\mathbf{C}$ encompasses the importance coefficients, and the $collision\_flag$ is a binary indicator of a collision event.

Student Action Space

The action space of the student remains unchanged from the manual curriculum learning setup.

Reward Features

The reward features used for training the agent are as follows:

- A reward based on the progress along the path of the end-effector and the puck.
- A reward based on the progress along the path of the puck and the goal.
- A reward based on the velocity of the end-effector.
- A target reward, which indicates whether the puck is scored.

The maximum end-effector velocity is used as a normalizing factor since it matches the maximum puck velocity. Therefore, this maximum velocity is used to normalize the reward related to the end-effector's velocity. The reward is calculated as follows:

$$R_{vel} = \frac{\|\text{velocity}_{ee}\|}{\|\text{velocity}_{ee,\max}\|}$$

Consider $P_{ee,\text{current}}$ to be the projection of the current end-effector position and $P_{ee,\text{previous}}$ to be the projection of the previous end-effector position, both on the vector $V_{ee\text{-to-puck}}$, which connects the initial end-effector position to the puck position. The reward related to the progress along the path for the distance between the end-effector and the puck is calculated as follows:

$$\text{displacement}_{ee,\max} = \text{velocity}_{ee,\max} \times \Delta t$$

$$R_{progress,ee\text{-to-puck}} = \frac{(P_{ee,\text{current}} - P_{ee,\text{previous}}) \cdot V_{ee\text{-to-puck}}}{\|\text{displacement}_{ee,\max}\|}$$

The progress-along-the-path reward for the distance between the puck and the goal is calculated similarly to the distance between the end-effector and the puck. All reward features, except for the target reward, are dense and normalized to a maximum value of 1 using the previously mentioned normalization method. However, since the reward is based on progress along the path, if the agent moves in the opposite direction, the negative reward can exceed -1. The target reward, given when the agent scores, is set to 2000.

Training

To train the automatic curriculum learning approach, two distinct environments were utilized: one for training and another for evaluation. Multiple instances of the training environment were created to enable parallel data collection.

At the start of each rollout for data collection, the teacher set the importance coefficients for the training environments. In the evaluation environment, the importance coefficients were all set to zero except for the one corresponding to the target task, which was set to one.

The student was then trained using the previously explained state, action, and reward structures. After a specified number of steps, the average reward obtained by the student in the training environments was calculated. To gain better insight into the agent's training status, evaluations are conducted after the training stage. Based on the teacher's operational approachwhether it focuses on learning progress or student rewardthe reward for the teacher was determined as previously described. Since the lower and upper bounds for the average reward are known, the calculated reward is interpolated between 0 and 1. This normalization enhances the convergence properties of the teacher algorithms.

The teacher updated itself using the reward it received and the $\boldsymbol{\theta}$ it output, which were used to calculate the importance coefficients ($\mathbf{C}$) assigned to the student. The teacher then sampled a new set of $\boldsymbol{\theta}$, calculated the corresponding importance coefficients ($\mathbf{C}$), and the process repeated. This iterative procedure allowed the teacher to continuously adapt and refine its strategy, optimizing the training process for the student. The Pseudo-code of this approach is represented in Algorithm 5.1.

5.5. Conclusion and Summary of Methods

In this thesis, we investigated curriculum learning strategies for training agents to perform specific tasks in air hockey, focusing on defense, counter-attack, and hit tasks. The methodologies employed included a manual curriculum for defense and counter-attack tasks, and an automatic curriculum learning approach for the hit task.

5.5.1. Summary of Methods

Manual Curriculum for Defense and Counter-Attack Tasks

- **Defense Task**: We implemented a structured manual curriculum that incrementally increased the difficulty of defensive scenarios. Unlike traditional curricula, we

Algorithm 5.1 Automatic Curriculum Learning Algorithm

Require: Initial teacher parameters: $\pi_{teacher}$, initial hyper-parameters: number of steps per training envronment N,

for each iteration **do**
  Sample $\boldsymbol{\theta}$ from teacher using $\pi_{teacher}$
  Calculate importance coefficients $\mathbf{C}$ from $\boldsymbol{\theta}$
  for each training environment instance **do**
    Set environment importance coefficients to $\mathbf{C}$ Execute N steps of data collection
  end for
  Evaluate student policy in evaluation environment with target task
  Compute average reward over training steps
  Calculate teacher reward based on learning progress or student reward
  Interpolate the teacher reward between 0 and 1
  Update teacher parameters $\pi_{teacher}$ using the interpolated teacher reward and the $\boldsymbol{\theta}$ used for calculating $\mathbf{C}$
end for

started with challenging task and then introduced the trained agent to easier environments. This approach allowed the agent to learn to defend in difficult situations first, subsequently enhancing its skills in simpler tasks. The progression continued by transitioning the agent through medium difficulty tasks before returning to hard tasks.

- **Counter-Attack Task**: To develop the counter-attack agent, we initially trained the agent in defense task. Afterward, by modifying the reward function, we shifted the focus to counter-attack task. Training the counter-attack task involved two levels of complexity, allowing the agent to effectively learn and adapt its counter-attacking strategies.

Automatic Curriculum Learning for Hit Task

- **Hit Task**: For the hitting task, we adopted an automatic curriculum learning approach using reinforcement learning. This method dynamically adjusted the difficulty of hitting challenges based on the agent's performance. The system continuously monitored metrics such as learning progress or the agent's average reward, and adjusted the focus of the student accordingly to maximize learning efficiency and performance.

6 | Results

In this chapter, we present the findings from our investigation into curriculum learning strategies applied to air hockey challenge. Initially, we delve into the outcomes of the manual curriculum learning approach implemented in the defend and counter-attack tasks, highlighting its efficacy and impact on performance metrics. Subsequently, we explore the automatic curriculum learning method utilized in the hit task, offering a comparative analysis to underscore the advancements and potential benefits of automated approaches. Through this examination, we aim to get across the relative strengths and challenges associated with both manual and automatic curriculum learning techniques in enhancing task-specific skills.

All the air hockey tasks were conducted in a simulated air hockey environment using an OpenAI Gym framework built on the MuJoCo simulation platform. The environment replicated a standard air hockey table, with sensors to detect puck position and motion. The simulation was run on a server with 88 Intel(R) Xeon(R) CPU E7-8880 v4 @ 2.20GHz CPUs and 94GB of RAM.

6.1. Manual Curriculum Learning

6.1.1. Defend Task

Experimental Setup

In this section, we detail the experimental setup for the manual curriculum learning applied to the defend task in the air hockey challenge. The goal is to provide a clear understanding of the methodology and conditions under which the experiment was conducted.

Unlike the usual approaches in curriculum learning, which start from simple and progressively move to more complex scenarios, this experiment adopted a reverse strategy. Initially, the agent was trained in the most complex environment, referred to as the "hard environment." This environment involved unpredictable puck trajectories, varying speeds,

and spins, requiring the agent to develop advanced defensive strategies from the outset.

Once the agent demonstrated consistent performance in the hard environment, it was transferred to the "easy environment." This simpler setting featured predictable puck movements with constant speeds and straight trajectories, providing the agent an opportunity to refine its basic defensive skills. After training in the easy environment, the usual curriculum learning approach was followed.

Results

The performance of the agent throughout the curriculum learning stages is illustrated in the Figure 6.1. Initially, the agent was trained in the hard environment for the first 3000 epochs, during which a steady increase in average return and success rate was observed. At epoch 3000, the agent trained in the hard environment was directly loaded into the easy environment. This transition led to a sharp spike in both average return and success rate, reflecting the agent's enhanced performance due to the significantly reduced complexity of the easy environment. From epochs 3500 to 4500, the training environment transitioned from easy to medium, resulting in a slight decrease in performance as the complexity increased. Training continued in the medium environment until epoch 6000, where the agent's performance stabilized. The subsequent transition back to the hard environment between epochs 6000 and 7000 caused another drop in performance, as the agent readjusted to the high complexity. Finally, from epochs 7000 to 7900, training exclusively in the hard environment resulted in a gradual recovery and improvement in both average return and success rate, demonstrating the agent's enhanced capability to handle the most complex scenarios.

Discussion

The results from the manual curriculum learning experiment shows several key insights into the agent's learning and adaptation processes. The initial training in the hard environment for the first 3000 epochs led to a steady increase in performance, indicating that the agent was able to develop effective defensive strategies despite the high complexity. The sharp spike in both average return and success rate upon transitioning the agent to the easy environment at epoch 3000 demonstrates the efficacy of the reverse curriculum learning approach. By starting with the most challenging scenarios, the agent was well-prepared to be expert in simpler environments. However, the subsequent performance drop during the transition from easy to medium and then from medium to hard environments underscores the need for a gradual increase in complexity to maintain performance stability. The final improvement in the challenging environment between epochs 7000

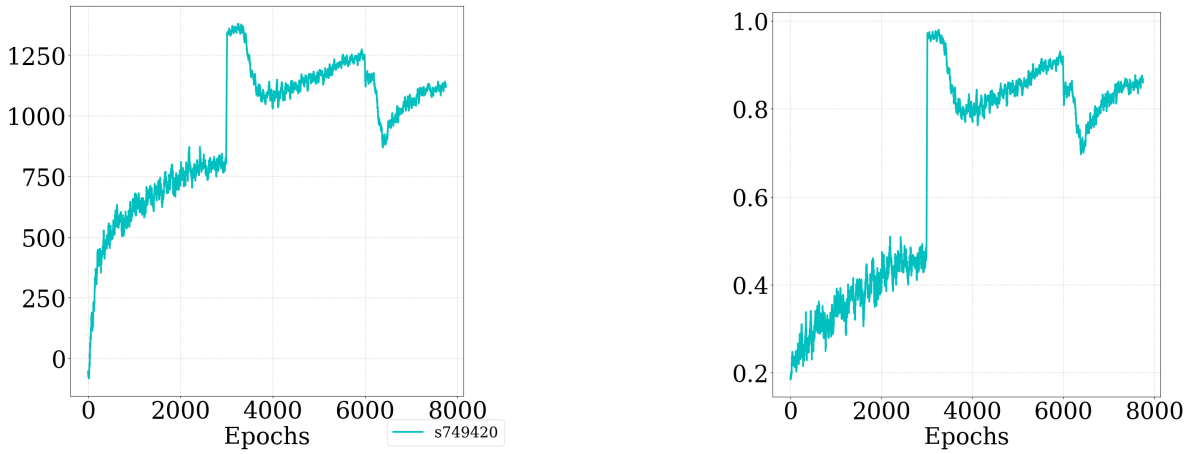


Figure 6.1: Performance metrics of the agent during curriculum learning. The average return (left) and average success rate (right) are plotted against the number of epochs. The training phases are as follows: hard environment (0-3000 epochs), easy environment (3000-3500 epochs), transitioning to medium environment (3500-4500 epochs), medium environment (4500-6000 epochs), transitioning back to hard environment (6000-7000 epochs), and hard environment (7000-7900 epochs). The plots highlight the agent's adaptation and performance changes across different training environments.

and 8000 suggests that the agent effectively strengthened its skills, becoming proficient in managing complex situations. These findings highlight the potential benefits of reverse curriculum learning in reinforcement learning tasks, suggesting that exposing agents to high complexity early on can enhance their overall adaptability and performance.

6.1.2. Counter-Attack Task

The factors influencing task complexity are consistent with those outlined for the defend task. Adding an opponent made the task excessively difficult, preventing the agent from solving it. Consequently, no results for this scenario are presented in this chapter.

Experimental Setup

In this section, we describe the experimental setup for the manual curriculum learning applied to the counter-attack task in the air hockey challenge. This setup follows the defend task previously outlined, utilizing the same hardware and simulation platform, but focusing on different episode termination conditions and curriculum progression.

The curriculum learning strategy for the counter-attack task built upon the defend task. Initially, the agent was trained using the defend task setup, as described previously. Upon

achieving the final stage of the defend curriculum, the agent was directly transferred to the counter-attack environment. This new environment featured a different reward function customized to the counter-attack objectives.

The first counter-attack environment was designated as the "medium environment." In this setting, puck trajectories and speeds were moderately unpredictable, requiring the agent to balance defensive maneuvers with offensive strategies. This phase allowed the agent to adapt its previously learned defensive skills to the new task's requirements.

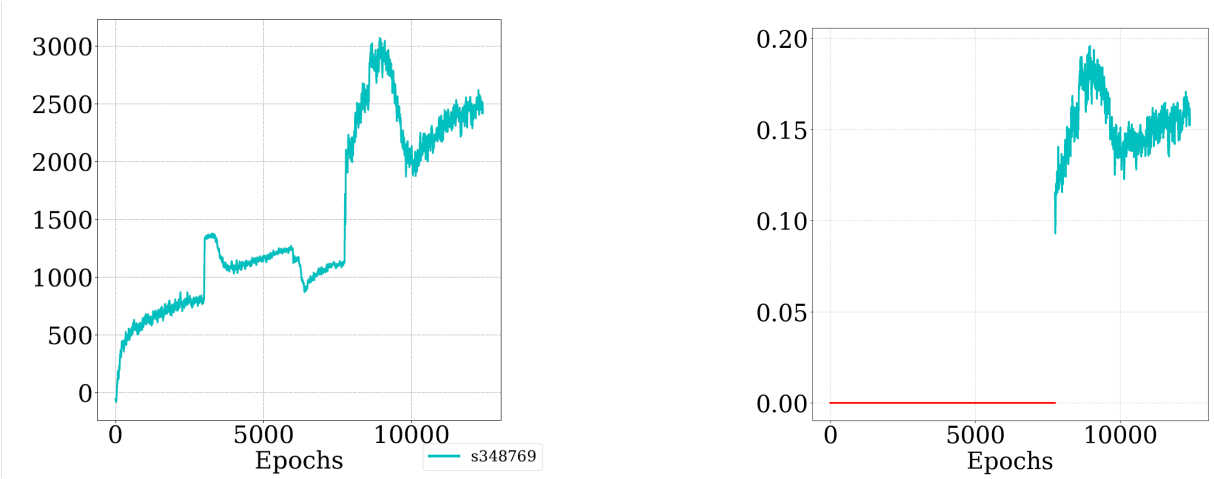
After demonstrating consistent performance in the medium environment, the agent transitioned to the "hard environment." The goal was to improve and enhance the agent's ability to respond to complex situations by building on the basic skills gained during the defense task and the intermediate training in the medium environment.

Results

The performance of the counter-attack agent throughout the various training stages is depicted in Figure 6.2. Initially, the agent was trained in the defend task up until epoch 8000. During this period, a consistent increase in average return was observed, demonstrating the agent's proficiency in the defend task. At epoch 8000, the training task switched to counter-attack in the medium environment. This change resulted in a significant increase in average return, as the agent adapted to the new task and environment, highlighting a marked improvement in performance and success rate started to be meaningful.

From epochs 9000 to 10000, the training environment transitioned from medium to hard. During this phase, the agent's performance exhibited variability, with notable peaks around epoch 9500, followed by a slight decline as the agent adjusted to the increased difficulty. This transition period is crucial as it reflects the agent's ability to adapt to progressively more challenging environments.

After epoch 10000, the agent was exclusively trained in the hard environment. Despite the initial drop in performance due to the increased difficulty, the agent's average return and success rate gradually stabilized, demonstrating an improved capacity to handle the high complexity of the hard environment. This stabilization indicates that the agent effectively learned and adapted to the counter-attack task, even under the most challenging conditions.



(a) Average Return (b) Average Success

Figure 6.2: Performance metrics of the counter-attack agent. The left plot (a) displays the average return over 13,000 epochs, illustrating the transition from the defend task training (0-7600 epochs) to the counter-attack task in the medium environment (7600-9000 epochs), transitioning from medium to hard environments (9000-10000 epochs), and finally training in the hard environment (10000-13000 epochs). The right plot (b) shows the average success rate, highlighting the agent's adaptation phase as it transitions from medium to hard environments and stabilizes during the hard environment training phase. The red color indicates that the success rate is not meaningful during the initial defend task training phase.

Discussion

The counter-attack task results highlight the agent's adaptability and the benefits of structured curriculum learning. Initially, the agent's performance in the defend task provided a strong foundation. Upon switching to the counter-attack task in the medium environment, there was a notable increase in both average return and success rate, indicating effective skill transfer from the defend task.

The transition to the hard environment introduced variability in performance, reflecting the increased complexity. However, the agent's gradual stabilization in average return and success rate during this phase demonstrates its capacity to adapt to more challenging conditions.

These findings underscore the efficacy of progressive training strategies, where the agent incrementally encounters higher complexity. The agent's ability to maintain and improve performance across varying tasks and environments showcases its robustness and potential for handling diverse scenarios.

6.2. Automatic Curriculum Learning

Experimental Setup

This section details the experimental setup for the automatic curriculum learning applied to the hit task in the air hockey challenge.

The automatic curriculum learning strategy employed a teacher-student architecture. Each experiment utilized 20 cores of the server. Number of steps before changing the importance coefficients are set to one rollout for each of the training environments, and each rollout consisted of exactly 1000 steps. Consequently, each importance coefficient underwent training for a total of 20,000 steps.

The LP-UCB and Reward-UCB algorithms are trained using three levels of discretization. This results in 4 reward features $\times$ 3 discretization levels, giving a total of 12 arms.

Results

Figure 6.3 illustrates the average return, average success and average target reward feature over time while evaluating for various approaches evaluated in this experiment. The following observations can be made based on the plot:

1. **Without Teacher:** The plot indicates that the approach without a teacher, relying only on sparse rewards, fails to increase the average return, resulting in no success

during the evaluation. Throughout the entire duration of the experiment, spanning over 20 million steps, the average return remains at zero. This demonstrates that relying only on sparse rewards is ineffective for learning in this context, as the agent is unable to improve its performance.

2. **Reward-UCB and LP-GMM:** Similar to the "without teacher" case, the Reward-UCB and LP-GMM approaches do not result in a significant increase in the average return. Even after 40 million steps, both methods fail to demonstrate any substantial improvement in performance. This suggests that these approaches are not suitable for the given task or environment.

3. **LP-UCB:** The LP-UCB approach shows a gradual increase in the average return over time. However, the improvement is minimal even after 40 million steps, indicating that while LP-UCB is capable of learning, it is highly sample inefficient. The agent requires a large number of steps to achieve a relatively small increase in performance, highlighting the method's inefficiency in terms of sample usage.

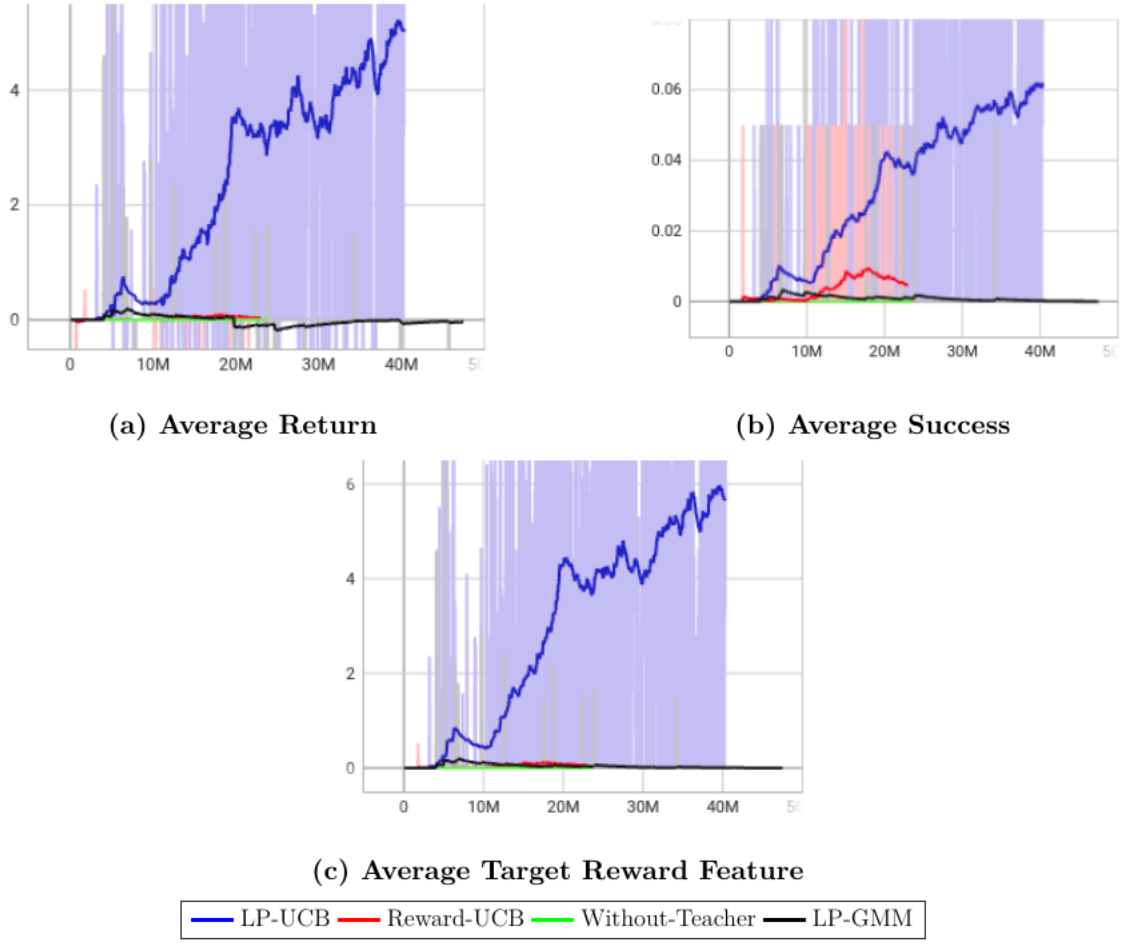
Conclusion

The analysis of the average return plot provides the following key insights:

- **Ineffectiveness of Sparse Rewards:** The approach without a teacher, which relies solely on sparse rewards, is ineffective for learning in this experiment. The agent's performance does not improve, as evidenced by the consistently zero average return.
- **Unsuitability of Reward-UCB and LP-GMM:** Both Reward_UCB and LP-GMM approaches do not lead to any significant enhancement in average return, indicating their unsuitability for the task at hand.
- **Sample Inefficiency of LP-UCB:** Although the LP-UCB method shows some improvement over time, but the improvement is achieved slowly. The minimal increase in average return after 40 million steps underscores the sample inefficiency of this approach.

Discussion on Learning Performance

The LP-UCB (blue) approach is the only one among the tested methods that demonstrates a capacity to learn the task, though at a very slow rate. This slow learning can be attributed to the complexity and variability introduced by the curriculum learning setup. In contrast, the reward-UCB (red), without-teacher (green), and LP-GMM (black) ap-



(a) Average Return

(b) Average Success

(c) Average Target Reward Feature

Figure 6.3: Performance metrics of the agent during automatic curriculum learning. The average return (a), average success rate (b) and average target reward feature (c) are plotted against the number of epochs. The plots highlight the agent's performance on the evaluation environment.

proaches do not exhibit significant learning. Their performance metrics remain relatively flat, indicating an inability to adapt and improve over time.

Discussion on Variance in Performance

A notable observation across all approaches is the high variance in their performance metrics. This high variance is particularly seen in the LP-UCB method, which, despite learning, shows considerable fluctuations. The primary reason for this high variance is the distribution shift caused by changing the importance coefficients during the training process. Each time the importance coefficients are adjusted, it results in a shift in the distribution of the task environment. This distribution shift leads to the agent exhibiting significantly different behaviors, thereby contributing to the observed high variance.

Discussion on Implications of Distribution Shifts

The impact of distribution shifts on learning stability is a critical insight from these results. The changes in importance coefficients are intended to facilitate the curriculum learning process by gradually introducing more complex scenarios. However, these changes also disrupt the agent's learned behavior, forcing it to continually adapt to new conditions. This adaptation process introduces instability and variability in performance, as reflected in the high variance seen in the plots.

6.3. Results of the Air Hockey Challenge

In this section, we present the results of the Air Hockey Challenge, which was conducted in three stages: warm-up, qualifying, and tournament.

6.3.1. Results of the Warm-up Phase

In the warm-up phase, we employed end-effector-centric modeling to simplify the problem. Using the SAC algorithm with reward engineering methods, we achieved a good success rate. However, the agent frequently violated constraints, resulting in its classification as undeployable. The results of this phase is presented in Figure 6.4.

6.3.2. Results of the Qualifying Phase

In this phase, we used joint-centeric modeling, outputting joint accelerations and then we passed the accelerations to ATACOM [29] to convert them to safe joint position and velocity commands. Using this approach, we could break less constraint and qualify to

Team Name	Hit Success	Defend Success	Max Penalty Points	Score
Air-HocKIT	69.8 %	47.3 %	385.0	58.5
Baseline	84.9 %	23.1 %	338.0	54.0
sprkrd	0.0 %	12.7 %	0.0	6.3
AJoy	0.0 %	12.7 %	0.0	6.3
Kalash Jain	0.0 %	12.7 %	0.0	6.3
RL3_polimi	85.0 %	12.2 %	4402.0	48.6

Figure 6.4: Results of the warm-up phase of the air hockey challenge.

Baseline-ATACOM	18.4 %	36.1 %	30.1 %	33.0	27.8
AJoy	18.4 %	36.1 %	20.0 %	108.0	25.8
Kalash Jain	0.0 %	19.5 %	8.3 %	0.0	9.5
RL3_polimi	22.7 %	61.6 %	36.2 %	920.0	41.0
AeroTron	33.2 %	23.2 %	66.5 %	594.0	35.9
CONFIRMTEAM	29.4 %	23.9 %	65.3 %	629.0	34.4
Tony	33.2 %	23.2 %	54.3 %	718.0	33.4

Figure 6.5: Results of the qualifying phase of the air hockey challenge.

enter to the tournament phase. The results of this phase is presented in Figure 6.5.

6.3.3. Results of the Tournament Phase

In this phase, we put all the agents together, and the team developed the high-level state machine. Using this approach, we participated in the one-by-one competitions. The results of this phase is presented in Figure 6.6.

Team Name	Round 1	Round 2	Total Points
AiRLIHockey	15	18	33
SpaceR	9	12	21
Air-HocKIT	5	15	20
AJoy	9	6	15
RL3_polimi	2	9	11
GXU-LIPE	2	1	3
CONFIRMTEAM	0	1	1

Figure 6.6: Results of the tournament phase of the air hockey challenge.

7 | Conclusion

This study explored various curriculum learning strategies to enhance the performance of RL agents in complex air hockey tasks. Initially, the SAC algorithm faced practical limitations, prompting a shift towards the ATACOM methodology and eventually the adoption of curriculum learning approaches.

Manual curriculum learning for the defend task demonstrated that starting in a challenging environment and progressively transitioning to easier settings enabled the agent to refine its defensive skills effectively. This strategic approach resulted in significant improvements in both average return and success rates, particularly when transitioning between complexity levels.

For the counter-attack task, leveraging the skills acquired from the defend task and applying them to a progressively challenging environment proved beneficial. The agent showed notable improvements in performance, successfully adapting its learned defensive strategies to offensive maneuvers, especially when transitioning from medium to hard environments.

The automatic curriculum learning approach further highlighted the potential of adaptive learning strategies. Among the evaluated methods, LP-UCB showed a gradual, yet minimal improvement in performance, indicating its capability to learn, though it was highly sample inefficient. In contrast, the Reward-UCB and LP-GMM approaches, similar to the sparse reward method, did not yield significant performance gains.

Overall, the implementation of curriculum learning, both manual and automatic, proved crucial in enhancing the agent's adaptability and performance in complex air hockey tasks. These findings underscore the importance of systematically structured learning environments and adaptive strategies in training RL agents for tasks requiring intricate skill development.

7.1. Future Directions

To mitigate the high variance and improve learning stability, future work could explore strategies for smoother transitions between different phases of the curriculum. This might involve more gradual adjustments to the importance coefficients or the introduction of mechanisms to help the agent maintain stability during distribution shifts. Additionally, investigating alternative methods that balance exploration and exploitation more effectively could enhance the learning rate and overall performance of the LP-UCB approach.

Furthermore, making the learning process more sample efficient is another crucial area for future research. Techniques such as importance sampling for the teacher could be employed to prioritize more informative samples, thereby reducing the number of samples needed for effective learning. Incorporating advanced replay buffer strategies or leveraging off-policy learning methods could also contribute to better sample efficiency, ultimately speeding up the learning process and improving overall performance.

Bibliography

[1] Air hockey challenge '23. `https://air-hockey-challenge.robot-learning.net`, 2023.

[2] M. Asada, E. Uchibe, and K. Hosoda. Cooperative behavior acquisition for mobile robots in dynamically changing real worlds via vision-based reinforcement learning and development. *Artificial Intelligence*, 110(2):275–292, 1999.

[3] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47:235–256, 2002.

[4] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch. Emergent tool use from multi-agent autocurricula. *arXiv preprint arXiv:1909.07528*, 2019.

[5] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[6] M. Breyer, F. Furrer, T. Novkovic, R. Siegwart, and J. Nieto. Comparing task simplifications to learn closed-loop object picking using deep reinforcement learning. *IEEE Robotics and Automation Letters*, 4(2):1549–1556, 2019.

[7] X. Cao, C. Sun, and M. Yan. Target search control of auv in underwater environment with deep reinforcement learning. *IEEE Access*, 7:96549–96559, 2019.

[8] I. Carlucho, M. De Paula, S. Wang, Y. Petillot, and G. G. Acosta. Adaptive low-level control of autonomous underwater vehicles using deep reinforcement learning. *Robotics and Autonomous Systems*, 107:71–86, 2018.

[9] S. R. B. dos Santos, S. N. Givigi, and C. L. Nascimento. Autonomous construction of multiple structures using learning automata: Description and experimental validation. *IEEE Systems Journal*, 9(4):1376–1387, 2015.

[10] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep

reinforcement learning for continuous control. In *International conference on machine learning*, pages 1329–1338. PMLR, 2016.

[11] J. L. Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1):71–99, 1993.

[12] A. Faust, P. Ruymgaart, M. Salman, R. Fierro, and L. Tapia. Continuous action reinforcement learning for control-affine systems with unknown dynamics. *IEEE/CAA Journal of automatica Sinica*, 1(3):323–336, 2014.

[13] C. Florensa, D. Held, M. Wulfmeier, M. Zhang, and P. Abbeel. Reverse curriculum generation for reinforcement learning. In *Conference on robot learning*, pages 482–495. PMLR, 2017.

[14] C. Florensa, D. Held, X. Geng, and P. Abbeel. Automatic goal generation for reinforcement learning agents. In *International conference on machine learning*, pages 1515–1528. PMLR, 2018.

[15] C. Fu and K. Chen. Gait synthesis and sensory control of stair climbing for a humanoid robot. *IEEE Transactions on Industrial Electronics*, 55(5):2111–2120, 2008.

[16] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[17] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[18] S. K. Gottipati, K. Seo, D. Bhatt, V. Mai, K. Murthy, and L. Paull. Deep active localization. *IEEE Robotics and Automation Letters*, 4(4):4394–4401, 2019.

[19] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[20] R. A. Howard. Dynamic programming and markov processes. 1960.

[21] Z. Huang, X. Xu, H. He, J. Tan, and Z. Sun. Parameterized batch reinforcement learning for longitudinal control of autonomous land vehicles. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49(4):730–741, 2017.

[22] S.-M. Hung and S. N. Givigi. A q-learning approach to flocking with uavs in a stochastic environment. *IEEE transactions on cybernetics*, 47(1):186–197, 2016.

[23] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, 2017.

[24] C.-F. Juang and C.-H. Hsu. Reinforcement ant optimized fuzzy controller for mobile-robot wall-following control. *IEEE Transactions on Industrial Electronics*, 56(10): 3931–3940, 2009.

[25] F. Khan, B. Mutlu, and J. Zhu. How do humans teach: On curriculum learning and teaching dimension. *Advances in neural information processing systems*, 24, 2011.

[26] M. Kumar, B. Packer, and D. Koller. Self-paced learning for latent variable models. *Advances in neural information processing systems*, 23, 2010.

[27] N. O. Lambert, D. S. Drew, J. Yaconelli, S. Levine, R. Calandra, and K. S. Pister. Low-level control of a quadrotor with deep model-based reinforcement learning. *IEEE Robotics and Automation Letters*, 4(4):4224–4230, 2019.

[28] 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. *arXiv preprint arXiv:1509.02971*, 2015.

[29] P. Liu, D. Tateo, H. B. Ammar, and J. Peters. Robot reinforcement learning on the constraint manifold. In *Conference on Robot Learning*, pages 1357–1366. PMLR, 2022.

[30] L. Lv, S. Zhang, D. Ding, and Y. Wang. Path planning via an improved dqn-based learning policy. *IEEE Access*, 7:67319–67330, 2019.

[31] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[32] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[33] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *Journal of Machine Learning Research*, 21(181):1–50, 2020.

[34] N. Palomeras, A. El-Fakdi, M. Carreras, and P. Ridao. Cola2: A control architecture for auvs. *IEEE Journal of Oceanic Engineering*, 37(4):695–716, 2012.

[35] J. Peng and R. J. Williams. Incremental multi-step q-learning. In *Machine Learning Proceedings 1994*, pages 226–232. Elsevier, 1994.

[36] L. Pinto and A. Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 3406–3413. IEEE, 2016.

[37] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17, 1964.

[38] R. Portelas, C. Colas, K. Hofmann, and P.-Y. Oudeyer. Teacher algorithms for curriculum learning of deep rl in continuously parameterized environments. In *Conference on Robot Learning*, pages 835–853. PMLR, 2020.

[39] M. Riedmiller, R. Hafner, T. Lampe, M. Neunert, J. Degrave, T. Wiele, V. Mnih, N. Heess, and J. T. Springenberg. Learning by playing solving sparse reward tasks from scratch. In *International conference on machine learning*, pages 4344–4353. PMLR, 2018.

[40] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[41] G. A. Rummery and M. Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[42] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[43] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[44] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[45] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. Pmlr, 2014.

[46] B. Singh, R. Kumar, and V. P. Singh. Reinforcement learning in robotic applications: a comprehensive survey. *Artificial Intelligence Review*, 55(2):945–990, 2022.

[47] K. O. Stanley, B. D. Bryant, and R. Miikkulainen. Evolving neural network agents in the nero video game. *Proceedings of the IEEE*, pages 182–189, 2005.

[48] F. Stulp, J. Buchli, A. Ellmer, M. Mistry, E. A. Theodorou, and S. Schaal. Model-free reinforcement learning of impedance control in stochastic environments. *IEEE Transactions on Autonomous Mental Development*, 4(4):330–341, 2012.

[49] S. Sukhbaatar, Z. Lin, I. Kostrikov, G. Synnaeve, A. Szlam, and R. Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. *arXiv preprint arXiv:1703.05407*, 2017.

[50] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3:9–44, 1988.

[51] R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990*, pages 216–224. Elsevier, 1990.

[52] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[53] G. Tesauro et al. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[54] S. Thrun and A. Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 connectionist models summer school*, pages 255–263. Psychology Press, 2014.

[55] S. G. Tzafestas and G. G. Rigatos. Fuzzy reinforcement learning control for compliance tasks of robotic manipulators. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 32(1):107–113, 2002.

[56] G. E. Uhlenbeck and L. S. Ornstein. On the theory of the brownian motion. *Physical review*, 36(5):823, 1930.

[57] J. P. Wang, Y. K. Shi, W. S. Zhang, I. Thomas, and S. H. Duan. Multitask policy adversarial learning for human-level control with large state spaces. *IEEE Transactions on Industrial Informatics*, 15(4):2395–2404, 2018.

[58] Y. Wang, H. Lang, and C. W. De Silva. A hybrid visual servo controller for robust grasping by wheeled mobile robots. *IEEE/ASME transactions on Mechatronics*, 15 (5):757–769, 2009.

[59] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[60] A. M. Whitbrook, U. Aickelin, and J. M. Garibaldi. Idiotypic immune networks in mobile-robot control. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(6):1581–1598, 2007.

[61] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

[62] C. Wu, B. Ju, Y. Wu, X. Lin, N. Xiong, G. Xu, H. Li, and X. Liang. Uav autonomous target search based on deep reinforcement learning in complex disaster scene. *IEEE Access*, 7:117227–117245, 2019.

[63] A. Xi, T. W. Mudiyanselage, D. Tao, and C. Chen. Balance control of a biped robot on a rotating platform based on efficient reinforcement learning. *IEEE/CAA Journal of Automatica Sinica*, 6(4):938–951, 2019.

[64] L. Xiao, C. Xie, M. Min, and W. Zhuang. User-centric view of unmanned aerial vehicle transmission against smart attacks. *IEEE Transactions on Vehicular Technology*, 67(4):3420–3430, 2017.

[65] X. Xu, C. Liu, S. X. Yang, and D. Hu. Hierarchical approximate policy iteration with binary-tree state space decomposition. *IEEE Transactions on Neural Networks*, 22(12):1863–1877, 2011.

[66] J. Yu, C. Wang, and G. Xie. Coordination of multiple robotic fish with applications to underwater robot competition. *IEEE Transactions on Industrial Electronics*, 63 (2):1280–1288, 2015.

[67] Y. Zeng, R. Zhang, and T. J. Lim. Wireless communications with unmanned aerial vehicles: Opportunities and challenges. *IEEE Communications magazine*, 54(5): 36–42, 2016.

[68] J. Zhu, J. Zhu, Z. Wang, S. Guo, and C. Xu. Hierarchical decision and control for continuous multitarget problem: Policy evaluation with action delay. *IEEE transactions on neural networks and learning systems*, 30(2):464–473, 2018.

[69] B. D. Ziebart. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. Carnegie Mellon University, 2010.

List of Figures

List of Tables

Acknowledgements

I would like to express my deepest gratitude to my advisor, Prof. Restelli, for his invaluable supervision and guidance throughout this journey. I am also immensely grateful to my co-advisor, Dr. Likmeta, for his unwavering support and encouragement. Finally, I extend my heartfelt thanks to my family, whose steadfast support and encouragement sustained me through the challenging days of this degree.