
Hierarchical Reinforcement Learning with Self-Play for Robotic Air Hockey

Master in the field of study "Computational Engineering" by Yuheng Ouyang
Date of submission: September 25, 2024

1. Review: Prof. Jan Peters
2. Review: Puze Liu
3. Review: Dr. Davide Tateo
Darmstadt



TECHNISCHE
UNIVERSITÄT
DARMSTADT

field of study:
Computational Engineering
Institut
Intelligente Autonome
Systeme

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

Hiermit erkläre ich, Yuheng Ouyang, dass ich die vorliegende Arbeit gemäß § 22 Abs. 7 APB der TU Darmstadt selbstständig, ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe mit Ausnahme der zitierten Literatur und anderer in der Arbeit genannter Quellen keine fremden Hilfsmittel benutzt. Die von mir bei der Anfertigung dieser wissenschaftlichen Arbeit wörtlich oder inhaltlich benutzte Literatur und alle anderen Quellen habe ich im Text deutlich gekennzeichnet und gesondert aufgeführt. Dies gilt auch für Quellen oder Hilfsmittel aus dem Internet.

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

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

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

Darmstadt, 25. September 2024

Yuheng Ouyang

Abstract

This thesis proposes an improved Hierarchical Reinforcement Learning (HRL) method, specifically the Termination Soft Actor-Critic (TSAC) algorithm running on the OCAD framework. To begin, we extended the Option-Critic Architecture (OCA) to an OCA with deterministic intra-option policies (OCAD), and subsequently implemented the Termination Soft Actor-Critic algorithm within this framework, namely the TSAC algorithm, which simultaneously learns the termination critic. Unlike the original OCA, our method adopts continuous options and efficiently learns the option critic, actor, and termination critic using the Markov Decision Process (MDP) replay buffer.

To validate our method, we chose to apply it in an air hockey environment using the LBR iiwa robotic arm. This decision was made because the widely applied 7-degree-of-freedom robotic arm has high-dimensional observation and action spaces, and the air hockey environment provides long-horizon and highly dynamic scenarios, thereby offering a comprehensive way to thoroughly test and validate our approach.

In our proposed method, the agent consists of two layers: a low-level agent and a high-level agent. The low-level agent functions as a deterministic controller, receiving options from the high-level agent and translating them into control commands. In contrast, the high-level agent is trained using the TSAC algorithm. In this research, we constructed and tested two types of low-level agents: Constrained Neural Motion Planning with B-splines (CNP-B) and Acting on the Tangent Space of the Constraint Manifold (ATACOM). After evaluating their performance, we eventually adopted ATACOM.

Furthermore, we conducted a series of experiments to validate the necessity of the two-layer structure and the effectiveness of the termination critic in our method. Finally, to enhance the agent's performance, we focused on two key aspects: Curriculum Learning (CL) and Self-Learning (SL). The experimental results demonstrated that both CL and SL significantly improved the agent's performance, with SL further enhancing the agent's generalization capability.


Zusammenfassung

Diese Arbeit schlägt eine verbesserte hierarchische Verstärkungslern-Methode (HRL) vor, konkret den Termination Soft Actor-Critic (TSAC)-Algorithmus, der auf dem OCAD-Framework ausgeführt wird. Zunächst haben wir die Option-Critic-Architektur (OCA) zu einer OCA mit deterministischen intra-Options-Politiken (OCAD) erweitert und anschließend den Termination Soft Actor-Critic-Algorithmus in diesem Framework implementiert haben, nämlich den TSAC-Algorithmus, der gleichzeitig den Terminationskritiker erlernt. Im Gegensatz zur ursprünglichen OCA verwendet unsere Methode kontinuierliche Optionen und erlernt effizient den Optionskritiker, Akteure und Terminationskritiker mithilfe des Markov-Entscheidungsprozesses (MDP) Replay-Puffers.

Um unsere Methode zu validieren, haben wir uns entschieden, sie in einer Air-Hockey-Umgebung mit dem LBR iiwa-Roboterarm anzuwenden. Diese Entscheidung fiel, weil der weit verbreitete 7-Grad-of-Freedom-Roboterarm hochdimensionale Beobachtungs- und Aktionsräume bietet und die Air-Hockey-Umgebung komplexe und hochdynamische Szenarien ermöglicht, was eine umfassende Möglichkeit bietet, unseren Ansatz gründlich zu testen und zu validieren.

In unserer vorgeschlagenen Methode besteht der Agent aus zwei Ebenen: einem low-level-Agenten und einem high-level-Agenten. Der low-level-Agenten fungiert als deterministischer Controller, der Optionen vom high-level-Agenten empfängt und sie in Steuerbefehle umsetzt. Im Gegensatz dazu wird der high-level-Agent mit dem TSAC-Algorithmus trainiert. In dieser Forschung haben wir zwei Arten von low-level-Agenten konstruiert und getestet: Constrained Neural Motion Planning mit B-Splines (CNP-B) und Acting on the Tangent Space of the Constraint Manifold (ATACOM). Nach der Bewertung ihrer Leistung haben wir schließlich ATACOM übernommen.

Darüber hinaus haben wir eine Reihe von Experimenten durchgeführt, um die Notwendigkeit der zwei-Schichten-Struktur und die Effektivität des Terminationskritikers in unserer Methode zu validieren. Schließlich haben wir auf zwei Schlüsselaspekte konzentriert, um die Leistung des Agenten zu verbessern: Curriculum Learning (CL) und Self-Learning (SL).



Die experimentellen Ergebnisse zeigten, dass sowohl CL als auch SL die Leistung des Agenten signifikant verbesserten, wobei SL die Generalisierungsfähigkeit des Agenten weiter steigerte.

Contents

1	Introduction	8
1.1	Proposed Method	8
1.1.1	Method Overview	8
1.1.2	Deployed Platform	9
1.1.3	Contribution	9
1.2	Related Work	10
1.2.1	Playing Air Hockey with Robots	10
1.2.2	Motion Planner	11
1.2.3	Curriculum Learning	12
1.2.4	Hierarchical Reinforcement Learning (HRL)	13
2	Fundamentals	15
2.1	Reinforcement Learning(RL)	15
2.1.1	Markov Decision Process	15
2.1.2	Bellman Equation	16
2.1.3	Policy Gradient Methods	17
2.2	Soft Actor Critic(SAC)	18
2.2.1	Entropy-Regularized Reinforcement Learning	18
2.2.2	Soft Actor-Critic	19
2.2.3	Learning the Policy	20
2.3	The Option-Critic Architecture(OCA)	20
2.4	CNP-B	22
2.5	ATACOM	23
3	Methods	26
3.1	Build Agent	26
3.2	Low Level Agent	27
3.2.1	Training CNP-B	27
3.2.2	ATACOM	28

3.3	Learning in High Level	29
3.3.1	High Level Agent	29
3.3.2	Intra-option Q-Learning	29
3.3.3	Learning Options	29
3.3.4	Learning Termination	33
3.3.5	Pseudo Code of High Level Training	34
3.4	Curriculum Learning	35
3.5	Self Learning	36
4	Experiment	37
4.1	Settings	37
4.1.1	Experiment Overview	37
4.1.2	Simulation Environment	38
4.1.3	Hyperparameters	40
4.2	Evaluation	41
4.2.1	CNP-B	41
4.2.2	Hierarchical Learning	43
4.2.3	Different Termination Probability	44
4.2.4	Curriculum Learning	46
4.2.5	Self Learning	47
4.2.6	Play Games	48
5	Conclusion and Future work	52

1 Introduction

1.1 Proposed Method

1.1.1 Method Overview

The thesis extends a hierarchical reinforcement learning (HRL) framework called the option-critic architecture (OCA) with deterministic intra-option policies, termed as OCAD. Moreover, we adopt the soft-actor-critic(SAC) algorithm[13], enabling it to simultaneously learn the termination critic, referred to the termination SAC(TSAC). It is worth noting that we have successfully implemented the OCAD with continuous options compared to the traditional discrete options. Please refer to Figure 3.1 for a clear flowchart of the structure and to Section 3.3 for the algorithm details.

The OCDA decomposes the task into two layers: low-level action control and high-level target selection, allowing the system to focus on learning high-level strategies without concerning itself with low-level trajectory planning. It is essentially a HRL approach. Reinforcement Learning (RL) faces significant challenges in handling complex, long-horizon tasks, especially in environments with high-dimensional state and action spaces[28, 5, 33]. HRL provides an effective solution to these challenges by breaking down complex tasks into simpler subtasks. In this framework, we learn the options of the Semi-Markov Decision Process (SMDP) using a MDP replay buffer. The higher-level policy aims to complete the overall task by selecting the optimal subtasks as its options. This task decomposition efficiently reduces the original task's long horizon into a shorter sequence of subtasks, each subtask acts as a high-level option that operates over a longer timescale than lower-level actions, a process known as temporal abstraction [4, 39]. HRL algorithms have demonstrated superior performance over standard RL in various long-horizon problems, such as continuous control [10, 23, 27], long-horizon games [20, 46], robotic manipulation [11, 12], and more.

The TSAC learns the termination critic by termination gradient shown in Equation 3.18, allowing for rapid switching between targets when necessary. This adaptability enables the system to better handle complex and dynamic environments.

The implemented TSAC algorithm can effectively learn the option critic, actor, and termination critic using the MDP replay buffer. This off-option method is also known as intro-option Q-learning. Due to the addition of an entropy bonus, the entropy of our policy is also regulated, thereby balancing exploration and exploitation.

1.1.2 Deployed Platform

We decided to validate our method in an air hockey environment using the LBR iiwa robot arm. This is a 7-axis arm known for its high degrees of freedom and flexibility. It is widely used in fields such as industrial automation, healthcare, research, and service robotics. Its additional degree of freedom enables high-speed, accurate movements and adaptability to dynamic changes, making it ideal for the requirements of the game. This also means that when validating our method, the algorithm’s adaptability to high dimension of observation and action space, as well as highly dynamic scenarios, can be fully tested by demonstrating its performance.

Moreover, playing air hockey with a robotic arm is inherently a long-horizon task that requires continuous control. And the dynamic and competitive environment requires the agent to employ swift, accurate, and adaptive strategies.

1.1.3 Contribution

Achievement

We proposed and validated a HRL method, implemented an improved algorithm called termination SAC(TSAC) over the OCAD, which makes the agent effectively learn the option critic, actor, and termination critic using the MDP replay buffer. And the continuous high level options are successfully executed.

We have successfully built two controllers capable of generating motion trajectories in air hockey games. One agent utilized a learning-to-plan framework called Constrained Neural Motion Planning with B-splines (CNP-B)[18], while the other implemented Acting on the Tangent Space of the Constraint Manifold (ATACOM)[25]. Both agents are also suitable for trajectory planning in other scenarios.

We have conducted experiments presented in Table 1.1.

Experiment Topic	Experiment Purpose
CNP-B	Train and evaluate the controller
Hierarchical Learning	Compare our method with one without HRL
Different Termination Probabilities	The influence of terminations
Curriculum Learning	The improvement using curriculum learning (CL)
Self Learning	The improvement using self learning(SL)
Play Games	The practical performance in competition

Table 1.1: Experiments and Purposes

In the experiments, we have evaluated the CNP-B and explored its improvement. We have validated our method and explored the possibility of enhancing the agent’s performance in air hockey games, primarily focusing on CL and SL aspects.

Conclusion

The CNP-B agent struggles to operate effectively under conditions with multiple terminations. The well-performing ATACOM agent is set as a low-level agent. Our proposed method works well and achieves ideal performance. It reduces the action space that needs to be learned, making exploration more efficient and lowering the task complexity. In practical applications, employing both SL and CL methods can significantly enhance the agent’s performance, ensuring stable behavior even when faced with previously unseen opponents.

1.2 Related Work

1.2.1 Playing Air Hockey with Robots

The development of air hockey-playing robots has advanced significantly in recent years, as researchers focus on various aspects such as control strategies, motion planning, and perception systems.

The multi-level control structure has been introduced, enabling the robot to manage

complex tasks more efficiently through a hierarchical processing architecture [29]. This architecture allowed for more organized and strategic gameplay by dividing tasks into different layers of control. Building on this foundational concept, the two-layer system further enhanced the robot's capabilities by integrating high-level tactics with low-level actions, enabling it to execute both offensive and defensive maneuvers more effectively and adaptively [38].

The approach incorporated visual feedback to adjust the robot's movements in real time, significantly improving its responsiveness during gameplay [6]. The integration of predictive planning represented another key advancement, enabling the robot to optimize its attack strategies by forecasting future puck positions and anticipating opponent movements. This predictive ability substantially increased the robot's competitiveness and allowed it to engage in more strategic play[16]. Further refinement was achieved through the application of optimal control theory combined with Bayesian tracking. This combination enabled the robot to accurately predict puck trajectories, strengthening its striking precision and decision-making capabilities [2].

More recent research emphasized the importance of developing highly reactive planning systems to handle the rapid pace of air hockey, achieving an effective balance between speed and accuracy through optimized planning algorithms that allowed the robot to respond swiftly to high-speed interactions[24]. The application of deep reinforcement learning also enabled the robot to adapt to various scenarios and opponents by learning optimal policies from experience[43].

1.2.2 Motion Planner

Neural motion planners have emerged as a promising alternative to classical motion planning approaches in recent years. Traditional planning methods, such as optimization-based and sampling-based planners, have been widely used for generating feasible trajectories in robotic applications. However, these methods often face challenges when dealing with complex, high-dimensional, or dynamic environments.

Optimization-based planners, like CHOMP [35] and TrajOpt [37], have shown effectiveness in generating smooth trajectories. Nonetheless, they are often computationally intensive and can struggle with non-linear constraints, leading to issues like getting stuck in local minima. On the other hand, sampling-based planners, such as RRT [21] and PRM [17], provide probabilistic completeness but often become inefficient as the dimensionality of the problem increases or when navigating narrow passages.

In contrast, neural motion planners leverage the learning capabilities of deep neural networks to model and solve planning problems more efficiently. By training on large datasets of demonstrations or using reinforcement learning, these planners can learn to generate feasible trajectories that adhere to task constraints. This approach enables them to handle high-dimensional inputs and adapt to complex dynamic environments more effectively. Recent works, such as Motion Planning Networks (MPNet) [34], have demonstrated that neural networks can significantly reduce the time required for planning by learning from prior experience, making them suitable for real-time applications.

The use of neural motion planners has opened new possibilities for handling motion planning tasks in highly dynamic and uncertain environments, showing potential for applications where classical methods may fall short. This area continues to be an active field of research, with ongoing efforts to improve their adaptability, robustness, and efficiency.

1.2.3 Curriculum Learning

Curriculum learning (CL) in reinforcement learning (RL) has emerged as a powerful strategy for improving both the speed and effectiveness of the learning process by structuring tasks to gradually increase in difficulty. The foundational idea of CL can be traced back to early studies in supervised learning, such as those presented in [9] and [36]. These studies demonstrated that training models on simpler tasks first, before progressing to more complex ones, improved convergence and performance. This approach allows learning algorithms to build a foundation of basic skills that can be refined and extended as task complexity increases.

In the context of RL, transfer learning has laid the groundwork for curriculum learning by showing how knowledge gained from simpler tasks can be effectively transferred to facilitate learning in more challenging environments [22, 44]. This transfer of knowledge is crucial in RL, where the exploration-exploitation trade-off often makes it difficult for agents to learn efficiently in complex tasks from scratch.

Recent works have made significant strides in formalizing CL frameworks for RL. For instance, [42] introduced a method for generating curricula using a directed acyclic graph (DAG) of tasks, where each node represents a task, and the edges define the progression from simpler to more complex tasks. This structure allows for automatic task sequencing, ensuring that the agent follows a learning trajectory that builds on previously acquired knowledge, thereby improving learning efficiency and performance.

Another notable contribution is the concept of reverse curriculum generation introduced in [30], which takes an innovative approach by starting the agent’s learning process with simpler, goal-like tasks and progressively moving towards the more challenging initial conditions. This reverse approach helps agents efficiently explore the state space and adapt to the complexities of the task environment.

Adaptive curriculum methods have also gained traction, where the task difficulty is adjusted dynamically based on the agent’s performance. The work in [26] proposed a “teacher-student” framework that automatically selects tasks based on the student’s current learning progress, ensuring that the agent is always challenged at an appropriate level. Similarly, [31] introduced a meta-learning approach that learns an optimal curriculum by evaluating the agent’s performance across different tasks, allowing the curriculum to adapt in real-time as the agent improves.

Despite these advancements, challenges remain in evaluating and standardizing curriculum learning across different RL domains. The diversity of task environments, agent capabilities, and learning objectives makes it difficult to establish universally applicable metrics for CL effectiveness. Addressing these challenges is essential for the broader adoption of curriculum learning in RL, as it holds the potential to significantly accelerate training and improve performance in complex decision-making tasks.

1.2.4 Hierarchical Reinforcement Learning (HRL)

Hierarchical Reinforcement Learning (HRL) has emerged as a powerful paradigm for tackling complex, long-horizon decision-making tasks by decomposing them into manageable subtasks, which enhances both exploration and learning efficiency. Early approaches, such as the Options Framework [39] and MAXQ value function decomposition [8], were instrumental in laying the foundation for HRL by introducing the concept of reusable skills or options, allowing agents to make decisions over extended time horizons. These frameworks excelled in abstracting temporal sequences of actions, making them particularly useful in tasks with sparse rewards or high-dimensional state spaces [4]. With the integration of deep learning, HRL has expanded its applicability to more complex domains, as demonstrated by neural network-based policy learning [46]. However, challenges such as efficient exploration, subtask discovery, and scalability of hierarchical policies persist as active research areas in HRL [32].

One of the most significant advancements in HRL is the option-critic architecture, which offers a more structured and end-to-end learning approach for hierarchical policies [3].

This architecture enables agents to learn both intra-option policies and termination functions through policy gradient methods, thereby eliminating the need for manual option definition. The option-critic framework’s ability to adaptively discover and refine options during training has greatly improved the flexibility and applicability of HRL.

Building upon the original option-critic architecture, the natural option critic [15] introduced natural gradient techniques to enhance the stability and efficiency of option learning, resulting in more robust performance in challenging environments. The Double Actor-Critic (DAC) architecture [47] further extended the option-critic framework by employing two actor-critic networks—one dedicated to selecting options and the other to learning option-specific policies. This separation allowed for more effective policy learning across different levels of the option hierarchy, especially in continuous action spaces. Additionally, the Soft Option Actor-Critic (SOAC) architecture [7] integrated the entropy-maximizing principles from the soft actor-critic framework into the option-critic model, promoting more exploratory behavior and achieving superior performance in stochastic environments.

A critical aspect of option execution within HRL is the decision of when to terminate an option. The introduction of the termination critic [14] addressed this by incorporating a critic that predicts the optimal timing for option termination, thereby mitigating issues associated with premature or excessively prolonged option execution and enhancing the agent’s efficiency in managing subtask transitions.

2 Fundamentals

2.1 Reinforcement Learning(RL)

Reinforcement Learning(RL) is learning how to map situations to actions, so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These two characteristics, trial-and-error search and delayed reward, are the most important distinguishing features of RL. The learner and decision maker is called the agent. The thing it interacts with, comprising everything outside the agent, is called the environment. These interact continually, the agent selecting actions and the environment responding to these actions and presenting new situations to the agent. The environment also gives rise to rewards, special numerical values that the agent seeks to maximize over time through its choice of actions[40].

2.1.1 Markov Decision Process

MDPs are a classical formalization of sequential decision making, where actions influence not only immediate rewards, but also subsequent situations, or states, and, through them, future rewards. Thus MDPs involve delayed reward and the need to trade off immediate and delayed reward.

More specifically, the agent and environment interact at each of a sequence of discrete time steps, $t = 0, 1, 2, 3, \dots$. At each time step t , the agent receives some representation of the environment's *state*, $S_t \in \mathbb{S}$, and on that basis selects an *action*, $A_t \in \mathbb{A}$. The probability distribution over actions conditioned on states is called *policy*, $\pi(a | s)$. One time step later, in part as a consequence of its action, the agent receives a numerical

reward, $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$, and the expected rewards for state–action–next-state triples as a three-argument function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S}' \rightarrow \mathbb{R}$:

$$r(s, a, s') \doteq \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a, S_t = s'] \quad (2.1)$$

and finds itself in a new state, S_{t+1} . We call the sequence $\{S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \dots\}$ as *trajectory*. We usually use a *discounting* γ in practice and try to select actions so that the sum of the discounted rewards it receives over the future is maximized. The expected *discounted return* defined as:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \quad (2.2)$$

In a finite MDP, the random variables R_t and S_t have well-defined discrete probability distributions dependent only on the preceding state and action. For particular values of $s' \in \mathcal{S}$ and $r \in \mathcal{R}$, there is a probability of those values occurring at time t , called *dynamics/transition* of the MDP:

$$p(s', r \mid s, a) \doteq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\} \quad (2.3)$$

2.1.2 Bellman Equation

In discounted problems, the *value function of a policy* π is defined as the expected return:

$$V_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right] \quad (2.4)$$

and the *action-value function for policy* π :

$$Q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right] \quad (2.5)$$

The Bellman equation expresses a relationship between the value of a state and the values of its successor states. For any policy π and any state s , the following consistency condition

holds between the value of s and the value of its possible successor states:

$$\begin{aligned}
V_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) [r + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s']] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) [r + \gamma V_\pi(s')]
\end{aligned} \tag{2.6}$$

There is always at least one policy that is better than or equal to all other policies. This is an *optimal policy*. We denote all the optimal policies by π_* . The optimal state-value function, denoted V^* , is defined as $V^*(s) \doteq \max_\pi V_\pi(s)$, the Bellman equation for V^* , or the *Bellman optimality equation*:

$$\begin{aligned}
V^*(s) &= \max_{a \in \mathcal{A}(s)} q_*(s, a) \\
&= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma V^*(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V^*(s')].
\end{aligned} \tag{2.7}$$

The Bellman optimality equation for Q^* is

$$\begin{aligned}
Q^*(s) &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}a') \mid S_t = s, A_t = a] \\
&= \sum_{s',r} p(s', r \mid s, a)[r + \gamma \max_{a'} Q^*(S_{t+1}, a')].
\end{aligned} \tag{2.8}$$

2.1.3 Policy Gradient Methods

Policy gradient methods [41, 19] address the problem of finding a good policy by performing stochastic gradient descent to optimize a performance objective over a given family of parameterized stochastic policies, π_θ . The policy gradient theorem [41] provides expressions for the gradient of the average reward and discounted reward objectives with respect to θ . In the discounted setting, the objective is defined with respect to a

designated start state (or distribution) s_0 : $\rho(\theta, s_0) = \mathbb{E}_{\pi_\theta} [\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_0]$. The policy gradient theorem shows that:

$$\frac{\partial \rho(\theta, s_0)}{\partial \theta} = \sum_s \mu_{\pi_\theta}(s | s_0) \sum_a \frac{\partial \pi_\theta(a | s)}{\partial \theta} Q_{\pi_\theta}(s, a), \quad (2.9)$$

where $\mu_{\pi_\theta}(s | s_0) = \frac{\sum_{t=0}^{\infty} \gamma^t \mathbb{P}(s_t=s | s_0)}{\sum_{t=0}^{\infty} \gamma^t}$ is a discounted weighting of the states along the trajectories starting from s_0 . In practice, the policy gradient is estimated from samples along the on-policy stationary distribution. [45] showed that neglecting the discount factor in this stationary distribution makes the usual policy gradient estimator biased. However, correcting for this discrepancy also reduces data efficiency. For simplicity, we build on the work of [41] and assume the stationary distribution is the discounted one.

2.2 Soft Actor Critic(SAC)

Soft Actor-Critic (SAC)[13] is an off-policy actor-critic algorithm based on the maximum entropy RL framework. In the framework, the actor aims to simultaneously maximize expected return and entropy, a measure of randomness in the policy. The higher entropy refers to the higher random act of the agent. This has a close connection to the exploration-exploitation trade-off: increasing entropy results in more exploration, which can accelerate learning later on. It can also prevent the policy from prematurely converging to a bad local optimum.

2.2.1 Entropy-Regularized Reinforcement Learning

Let x be a random variable with probability mass or density function P . The entropy H of x is computed from its distribution P according to

$$H(P) = -\mathbb{E}_{x \sim P} [\log P(x)]$$

In entropy-regularized reinforcement learning, the agent gets a bonus reward at each time step proportional to the entropy of the policy at that time step. This changes the RL

problem to:

$$\pi^* = \arg \max_{\pi} \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t (R(s_t, a_t, s_{t+1}) + \alpha * H(\pi(\cdot | s_t))) \right]$$

where $\alpha > 0$ is the trade-off coefficient. Then we define the slightly-different value functions in this setting. V^π is changed to include the entropy bonuses from every time step:

$$V^\pi(s) = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) + \alpha \sum_{t=1}^{\infty} \gamma^t H(\pi(\cdot | s_t)) \mid s_0 = s, a_0 = a \right]$$

With these definitions, V^π and Q^π are connected by:

$$V^\pi(s) = \mathbb{E}_{a \sim \pi} [Q^\pi(s, a) + \alpha H(\pi(\cdot | s))]$$

and the Bellman equation for Q^π is

$$\begin{aligned} Q^\pi(s, a) &= \mathbb{E}_{s' \sim P, a' \sim \pi} [R(s, a, s') + \gamma(Q^\pi(s', a') + \alpha H(\pi(\cdot | s')))] \\ &= \mathbb{E}_{s' \sim P} [R(s, a, s') + \gamma V^\pi(s')] \end{aligned}$$

2.2.2 Soft Actor-Critic

SAC concurrently learns a policy π_ψ and two Q-functions $Q_{\theta_1}, Q_{\theta_2}$. We'll start by taking our recursive Bellman equation for the entropy-regularized Q^π from earlier and rewriting it slightly using the definition of entropy:

$$\begin{aligned} Q^\pi(s, a) &= \mathbb{E}_{s' \sim P, a' \sim \pi} [R(s, a, s') + \gamma(Q^\pi(s', a') + \alpha H(\pi(\cdot | s')))] \\ &= \mathbb{E}_{s' \sim P, a' \sim \pi} [R(s, a, s') + \gamma(Q^\pi(s', a') - \alpha \log \pi(a' | s'))] \end{aligned}$$

we can approximate it with samples:

$$Q^\pi(s, a) = r + \gamma(Q^\pi(s', \tilde{a}') - \alpha \log \pi(\tilde{a}' | s')), \tilde{a}' \sim \pi(\cdot | s')$$

SAC sets up the MSBE loss for each Q-function using this kind of sample approximation for the target. SAC uses the clipped double-Q trick and takes the minimum Q-value between the two Q approximators. The loss functions for the Q-networks in SAC are:

$$L(\theta_i, D) = \mathbb{E}_{(s, a, r, s', d) \sim D} [((Q_{\theta_i}(s, a) - y(r, s', d))^2)]$$

where the target is given by

$$y(r, s', d) = r + \gamma(1 - d) \left(\min_{j=1,2} Q_{\theta_{\text{targs},j}}(s', \tilde{a}') - \alpha \log \pi_\psi(\tilde{a}' | s') \right), \quad \tilde{a}' \sim \pi_\psi(\cdot | s')$$

2.2.3 Learning the Policy

The policy should, in each state, act to maximize the expected future return and expected future entropy. That is, it should maximize $V^\pi(s)$, which we expand out into

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_{a \sim \pi} [Q^\pi(s, a) + \alpha H(\pi(\cdot|s))] \\ &= \mathbb{E}_{a \sim \pi} [Q^\pi(s, a) - \alpha \log \pi(a|s)] \end{aligned}$$

The way we optimize the policy makes use of the reparameterization trick, in which a sample from $\pi_\psi(\cdot|s)$ is drawn by computing a deterministic function of state, policy parameters, and independent noise. Following the authors of the SAC paper, we use a squashed Gaussian policy, which means that samples are obtained according to

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

The policy is thus optimized according to

$$\max_{\psi} \mathbb{E}_{s \sim \mathcal{D}, \xi \sim \mathcal{N}} \left[\min_{j=1,2} Q_{\theta_j}(s, \tilde{a}_\psi(s, \xi)) - \alpha \log \pi_\psi(\tilde{a}_\psi(s, \xi) | s) \right]$$

The Loss function of the policy by using gradient descent is

$$L_\pi(\psi, \mathcal{D}) = \mathbb{E}_{s \sim \mathcal{D}, \xi \sim \mathcal{N}} \left[\alpha \log \pi_\psi(\tilde{a}_\psi(s, \xi) | s) - \min_{j=1,2} Q_{\theta_j}(s, \tilde{a}_\psi(s, \xi)) \right]$$

2.3 The Option-Critic Architecture(OCA)

The option-critic architecture tackles the problem of temporal abstraction. It derives policy gradient theorems for options and is capable of learning both the internal policies and the termination conditions of options, in tandem with the policy over options, and without the need to provide any additional rewards or subgoals.

An option ω is defined as a triple $(I_\omega, \pi_\omega, \beta_\omega)$, where I_ω represents the initiation set, $\pi_\omega : S \times A \rightarrow [0, 1]$ is the intra-option policy, and $\beta_\omega : S \rightarrow [0, 1]$ denotes the termination function. The initiation set $I_\omega \subseteq S$ specifies where the option ω can be initiated. In many scenarios, I_ω is assumed to be the entire state space S , meaning that the option can be initiated at any state.

Once an option is initiated, the agent selects actions according to the intra-option policy π_ω . The option will continue to be executed until it is terminated based on the termination function β_ω , which provides the probability of termination at each state.

The framework of options allows the reinforcement learning problem to be extended from MDPs to SMDPs. In this extended framework, the agent's behavior is determined by a policy over options $\pi_\Omega : S \times \Omega \rightarrow [0, 1]$, where Ω is the set of all options. The value functions associated with the policy over options are defined as follows:

$$\begin{aligned} Q_\Omega(s, \omega) &= \sum_a \pi_\omega(a|s) Q_U(s, \omega, a), \\ Q_U(s, \omega, a) &= r(s, a) + \gamma \mathbb{E}_{s' \sim p(\cdot|s, a)} [U(\omega, s')], \\ U(\omega, s') &= (1 - \beta_\omega(s')) Q_\Omega(s', \omega) + \beta_\omega(s') V_\Omega(s'), \\ V_\Omega(s) &= \sum_\omega \pi_\Omega(\omega|s) Q_\Omega(s, \omega), \end{aligned}$$

where $Q_\Omega(s, \omega)$ is the value of an option ω initiated in state s , $Q_U(s, \omega, a)$ is the value of executing action a in the context of the option ω and state s , and $V_\Omega(s)$ is the value function over options.

The paper [3] introduced the Option-Critic Architecture (OCA), a policy gradient method designed to learn both the intra-option policies $\{\pi_\omega\}$, parameterized by θ_π , and the termination functions $\{\beta_\omega\}$, parameterized by θ_β . The goal of OCA is to maximize the expected discounted return, $Q_\Omega(s_0, \omega_0)$, by applying the Intra-option Policy Gradient Theorem and the Termination Gradient Theorem:

$$\begin{aligned} \Delta\theta_\pi &\propto \nabla_{\theta_\pi} \log \pi_{\omega_t}(a_t|s_t) Q_U(s_t, \omega_t, a_t) \\ \Delta\theta_\beta &\propto -\nabla_{\theta_\beta} \beta_{\omega_{t-1}}(s_t) (Q_\Omega(s_t, \omega_{t-1}) - V_\Omega(s_t)) \end{aligned}$$

At each time step t , OCA performs a gradient descent step to minimize the loss:

$$\frac{1}{2} (g_t - Q_U(s_t, \omega_t, a_t))^2$$

where g_t is the update target defined as:

$$g_t = r_{t+1} + \gamma \left((1 - \beta_{\omega_t}(s_{t+1})) Q_\Omega(s_{t+1}, \omega_t) + \beta_{\omega_t}(s_{t+1}) \max_{\omega'} Q_\Omega(s_{t+1}, \omega') \right)$$

This update target is also utilized in the Intra-option Q-learning framework [39].

2.4 CNP-B

In the paper [18] the author proposed a novel learning-to-plan framework, called *constrained neural motion planning with B-splines* (CNP-B), which generates plans satisfying an arbitrary set of constraints and computes them in a short constant time. This allows the robot to plan and replan reactively.

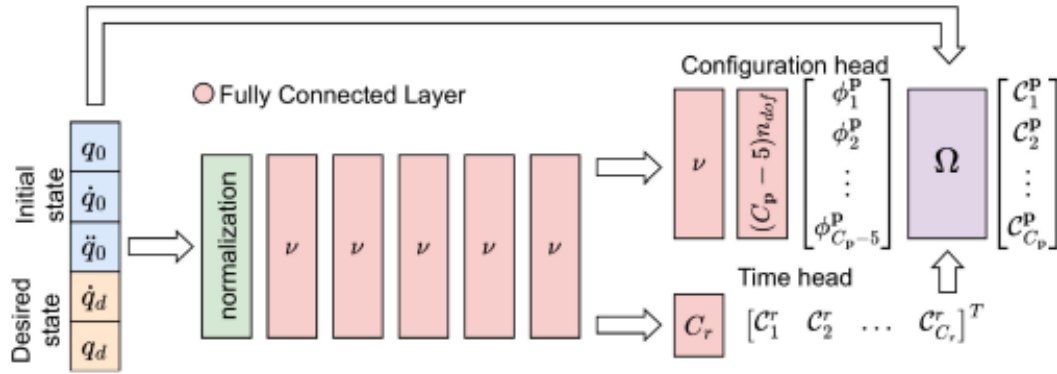


Figure 2.1: Architecture of the neural network used to determine $p(s)$ and $r(s)$ B-spline control points. The ω Block computes the control points of the configuration B-spline C^p

B-splines (Basis Splines) are a crucial curve representation method, widely applied in motion planning and trajectory generation. Mathematically, a B-spline is defined as a linear combination of control points, given by the following equation:

$$C(s) = \sum_{i=0}^n N_{i,k}(s) \mathbf{P}_i$$

where \mathbf{P}_i are the control points, $N_{i,k}(s)$ are the B-spline basis functions of degree k , and n is the degree of the curve. By adjusting these control points, we can precisely control the smoothness and shape of the curve, ensuring that the generated trajectory adheres to the required constraints, even in dynamic environments. This flexibility significantly accelerates trajectory generation under dynamic constraints.

In the CNP-B method, B-splines are used not only for generating initial trajectories but also for ensuring that the trajectory meets the specified boundary conditions, such as position, velocity, and acceleration at both the start and the end. These boundary conditions can be formulated as:

$$\begin{aligned}q(0) &= q_0, & \dot{q}(0) &= \dot{q}_0, & \ddot{q}(0) &= \ddot{q}_0 \\q(1) &= q_d, & \dot{q}(1) &= \dot{q}_d\end{aligned}$$

The optimization process minimizes the total loss function $L(\zeta)$, defined as:

$$L(\zeta) = \int_0^T L_t(q(t), \dot{q}(t), \ddot{q}(t)) dt$$

Thus, the CNP-B framework provides an effective way to represent trajectories on the constraint manifold, while its flexibility and efficiency allow for rapid motion planning under dynamic constraints.

2.5 ATACOM

ATACOM, which stands for *Acting on the Tangent Space of the Constraint Manifold*, is a novel reinforcement learning approach developed to address the challenge of safe exploration in environments with dynamic constraints[25]. ATACOM transforms the constrained reinforcement learning problem into an unconstrained one by allowing the agent to explore in the tangent space of the constraint manifold. This approach enables RL algorithms to remain within the feasible space defined by these constraints while maintaining high computational efficiency and robust learning performance.

In the context of RL, constraints such as joint limits, velocity limits, and task-specific requirements (e.g., keeping an end-effector on a surface) must be adhered to continuously. In ATACOM, these constraints are represented as a constraint manifold, where the system's state must evolve. The method leverages the tangent space of this manifold, enabling exploration without violating the constraints. The general state constraints can be expressed as:

$$f(q) = 0, \quad g(q) \leq 0$$

where $f(q)$ represents equality constraints and $g(q)$ represents inequality constraints. To handle these constraints, ATACOM utilizes the Jacobian of the constraint manifold to calculate the null space, representing the allowable directions of motion in the tangent space. The time derivative of the constraints is calculated as:

$$\dot{c}(q, \mu, \dot{q}, \dot{\mu}) = J_c(q, \mu) \begin{bmatrix} \dot{q} \\ \dot{\mu} \end{bmatrix} = 0$$

By ensuring that the trajectory remains within this null space, the constraints are continuously satisfied.

ATACOM further simplifies the constraint problem by converting inequality constraints into equality constraints with the introduction of slack variables μ , allowing the system to avoid over-constraining the RL algorithm. This reformulation is given by:

$$c(q, \mu) = \begin{bmatrix} f(q) \\ g(q) + \frac{1}{2}\mu^2 \end{bmatrix} = 0$$

In addition to equality constraints, ATACOM handles inequality constraints, such as joint limits and velocity limits, which restrict the feasible state-action space. By operating in the tangent space of the lower-dimensional manifold, ATACOM enables efficient exploration and faster learning, particularly in high-dimensional tasks.

The ATACOM approach has several advantages:

- It supports both equality and inequality constraints, maintaining all constraints below a predefined tolerance during learning.
- It does not require an initial feasible policy, allowing the agent to learn from scratch.
- It eliminates the need for manual safe backup policies, as the constraints are respected throughout the learning process.
- ATACOM can be integrated with any model-free RL algorithm, such as PPO, TRPO, DDPG, TD3, or SAC, making it highly flexible.

-
- By exploring the lower-dimensional manifold, ATACOM reduces the effective exploration space, leading to improved learning performance compared to unconstrained exploration.

3 Methods

We extend the option-critic architecture(OCA) with deterministic intra-option policies, referred to as OCAD. Under this structure we implement a SAC algorithm with termination condition, referred to as termination SAC(TSAC). The method inherits the hierarchical structure of OCA and control over entropy of SAC. It is worth noting that it is also applicable to continuous options. During the implementation, we add the entropy bonus to the option-value function and update option critic, termination critic and actor effectively using the MDP replay buffer.

3.1 Build Agent

In the method proposed in this paper, the agent is structured into two layers: a high-level agent that issues options and a low-level agent that receives these options and translates them into control commands. Additionally, the termination critic within the high-level agent determines whether to continue with the previous option or to adopt a new one. The structure of the agent can be seen in Fig. 3.1.

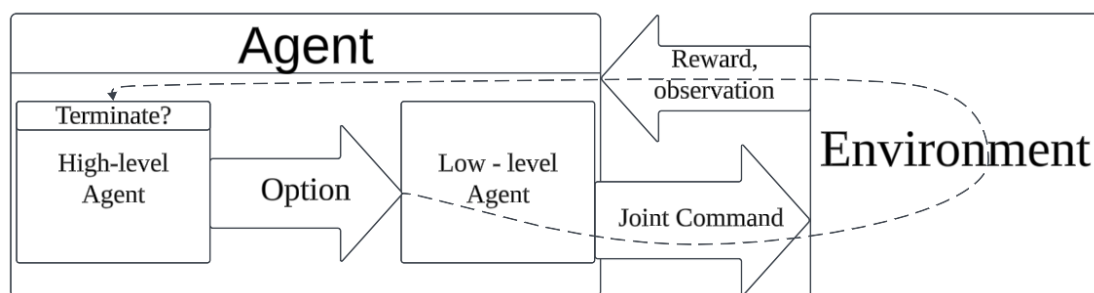


Figure 3.1: The structure of the agent built by our method

3.2 Low Level Agent

The low-level agent receives the option command of high-level agent and convert it to the joint command of the robot arm. It must be compatible with the task environment, which means it must meet the constraints. In this thesis, the robot arm’s generated trajectory must first be smooth and must not exceed the maximum and minimum limits of actual joint velocities and joint positions. Obviously, it should also avoid collisions with other joints or the table surface. Additionally, the end effector must always remain above the table.

ATACOM[25] and CNP-B[18] are built as low level agents in this thesis. CNP-B is trained by converting the violated parts of the generated trajectory into a loss function, while ATACOM adopts a simpler approach by optimizing the trajectory in real time through constraints. Their outputs are shown in Table 3.1.

Description	Methods	
	CNP-B	ATACOM
Output of high level/Input of low level	$[x, y, \rho, \eta]$	$[x, y]$
Output of low level	$\mathbf{u}_t \in \tau_{A \rightarrow E}$	$\mathbf{u}_t = f(s_t, o_t)$

Table 3.1: Comparison of Input and Output of Different Low-Level Agents. \mathbf{u} is the joint command for controlling. $\tau_{A \rightarrow E}$ means the trajectory from start point A to end point E. s means state and o means option. (x, y) represents the target point coordinates, ρ is the hit direction, and η is the speed scale.

3.2.1 Training CNP-B

The input of CNP-B is the joint positions and joint velocities $[\mathbf{q}, \dot{\mathbf{q}}]$ of both the start and end points. Its output would be the whole trajectory from start point to end point. To train CNP-B, we first define the constraints that the generated trajectories must satisfy. The constraints required for CNP-B in our thesis are shown in the following Table 3.2.

Once the constraints are set, we can prepare the training dataset for CNP-B. We employed four methods to construct the training dataset:

Constraints	Explanation
x_loss	The trajectory does not exceed the x-axis boundaries of the table
y_loss	The trajectory does not exceed the y-axis boundaries of the table
z_loss	The trajectory remains at a fixed height along the z-axis
q_loss	The joint positions do not exceed their limits
q_dot_loss	The joint velocities do not exceed their limits
t_loss	Minimize the execution time of the trajectory as much as possible
obstacle	The trajectory's intermediate points do not include the target point

Table 3.2: Constraints and Their Explanations

No.	Description
1	Set 100,000 random start and end point pairs of the task space.
2	Randomly generate 100,000 points of the task space and pair them arbitrarily during training.
3	Randomly generate 50,000 points of the task space, while the remaining points are randomly sampled from trajectories generated by the pre-trained CNP-B.
4	Continuously add data from planning failure points to the existing dataset for further training.

Table 3.3: Training Data Generation and Augmentation Strategies

When generating the data, we add noise to the optimal joint configurations of the start and end points to generate joint positions. We use linear programming to find the maximum velocity and multiply a random scaling factor $\zeta \in [0, 1]$ as hitting velocity, as indicated by the following optimize objective[24]:

$$\max_{\{v, \alpha\}} v, \quad \text{s.t.} \quad \dot{q}_l \leq J_{\text{pinv}} \cdot v + N \cdot \alpha \leq \dot{q}_u \quad (3.1)$$

where α is null space velocity, N is null space matrix, J_{pinv} is pseudo inverse of Jacobian matrix, \dot{q}_l and \dot{q}_u is the minimize and maximum limit of joint velocity.

3.2.2 ATACOM

We built an ATACOM agent for the air hockey environment, where the input is the 2D coordinates (x, y) of any point in the task space of the robotic arm, and the output is the control signal in the form of joint commands $[\mathbf{q}, \dot{\mathbf{q}}]$. The more details please refer to [25].

3.3 Learning in High Level

3.3.1 High Level Agent

The high-level agent is primarily used to issue decision-making instructions to the low-level agent for execution. Therefore, different low-level agents require different high-level instructions. In this thesis, there are mainly two types of action spaces, as shown in Table 3.1.

Based on the OCDA structure and TSAC algorithm, the high-level agent contains 7 neural networks to approximate different functions. They are two critic networks, two target critic networks, two actor and a termination function, the features of the networks are shown in Table 4.4.

After the warmup steps in Table 4.4, the high-level agent begins to learn the parameters every step. It extracts the data in the replay buffer and then calculates the gradient to update the parameters.

Every time the agent draws the option, it will first evaluate the previous option using the termination critic. Falls the previous option can be expected more reward, it will most likely to keep this option. Otherwise it may terminate the option and sample a new one.

3.3.2 Intra-option Q-Learning

The option framework is formalized on the basis of the theory of Semi-Markov Decision Process (SMDP). SMDP is an extension of the MDP where actions can take variable amounts of time to complete, and in the context of HRL, it refers to the temporary abstract option, and the low-level action follows the intra-option policy. When all intra-option policies are deterministic, we can implement effective Intra-option Q-Learning[39, 47]. The update equation 3.12 is applied to every option w satisfying $\pi_w(A_t|S_t) > 0$. We refer to this property as off-option.

3.3.3 Learning Options

Option-Value Function

The option-value function Q_Ω of OCA[3] is defined as :

$$Q_{\Omega}(s_t, \omega_t) = \sum_{a_t} \pi_{\omega}(a_t | s_t) Q_{\theta}(s_t, \omega_t, a_t) \quad (3.2)$$

Because of deterministic intra-option policies, $\pi_{\omega}(a_t | s_t) = 1$, the option-value function $Q_{\Omega}(s_t, \omega_t)$ is defined as

$$Q_{\Omega}(s_t, \omega_t) = Q_{\theta}(s_t, \omega_t, a_t) \quad (3.3)$$

$Q_{\theta}(s_t, \omega_t, a_t)$ is the value of executing an action a_t in the context of a state-option pair (s_t, ω_t) , we simply call it option critic. In our method, we additionally incorporated an entropy bonus in option critic Q_{θ} :

$$Q_{\theta}(s_t, \omega_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p(\cdot | s_t, a_t)} U(\omega_t, s_{t+1}) - \alpha \log \pi_{\psi}(\omega_t | s_t) \quad (3.4)$$

α is the trade-off coefficient the same as in SAC. $\pi_{\psi}(\omega_t | s_t)$ is the sample option policy over current state. Due to the deterministic intra-option policies, we will use the notation $Q_{\theta}(s, w)$ in the following sections and omit action a . $U(\omega_t, s_{t+1})$ denotes the option value upon arrival at the state option pair (s_{t+1}, ω_t) :

$$U(\omega_t, s_{t+1}) = (1 - \beta_{\phi}(s_{t+1}, \omega_t)) Q_{\theta}(s_{t+1}, \omega_t) + \beta_{\phi}(s_{t+1}, \omega_t) V(s_{t+1}) \quad (3.5)$$

Thus the complete expression of $Q_{\theta}(s_t, \omega_t)$

$$Q_{\theta}(s_t, \omega_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p(\cdot | s_t, a_t)} [(1 - \beta_{\phi}(s_{t+1}, \omega_t)) Q_{\theta}(s_{t+1}, \omega_t) + \beta_{\phi}(s_{t+1}, \omega_t) V(s_{t+1})] - \alpha \log \pi_{\psi}(\omega_t | s_t) \quad (3.6)$$

which is parameterized by θ in our implement.

Value Function

The relationship between $V(s)$ and $Q_{\theta}(s, \omega)$:

$$V(s_t) = \sum_{\omega_t} \pi_{\Omega}(\omega_t | s_t, \omega_{t-1}) Q_{\theta}(s_t, \omega_t) \quad (3.7)$$

where the option policy $\pi_\Omega(\omega_t | s_t, \omega_{t-1})$ includes the probability of keeping the previous option, or the probability of sampling a new option. Please note the difference between option policy π_Ω and sample option policy π_ψ .

$$\pi_\Omega(\omega_t | s_t, \omega_{t-1}) = (1 - \beta_\phi(s_t, \omega_{t-1})) I_{\omega_t = \omega_{t-1}} + \beta_\phi(s_t, \omega_{t-1}) \pi_\psi(\omega_t | s_t) \quad (3.8)$$

where $\beta_\phi(s_t, \omega_{t-1})$ is the termination function, which outputs the termination probability of the current option-state pair. It is parameterized by ϕ in our implementation. A higher β value indicates that the current option is more likely to be terminated.

In the actual implementation of the algorithm, we use a Monte Carlo estimator to approximate the value of the V function, as shown in Equation 3.21.

Policy Gradient

The policy gradient is the same as which used in SAC[13]. The policy tries to maximize the $V^\pi(s_t)$

$$\begin{aligned} V^\pi(s_t) &= \mathbb{E}_{\omega \sim \pi_o} [Q_\theta(s_t, \omega_t) + \alpha H(\pi_o(\cdot | s_t))] \\ &= \mathbb{E}_{\omega \sim \pi_o} [Q_\theta(s_t, \omega_t) - \alpha \log \pi_o(\omega | s_t)] \end{aligned} \quad (3.9)$$

The policy is thus optimized according to

$$\max_{\psi} \mathbb{E}_{s \sim \mathcal{D}} \left[\min_{j=1,2} Q_{\theta_j}(s, \tilde{\omega}_{\psi_j}(s)) - \alpha \log \pi_{\psi_j}(\tilde{\omega}_{\psi_j}(s) | s) \right] \quad (3.10)$$

where $\tilde{\omega}_{\psi}$ means the sampled options of the actor based on states in replay buffer, the actor is parameterized by τ .

Object

The option-value function $Q_\theta(w, s)$ parameterized by θ can be trained to minimize the Bellman residual:

$$J_Q(\theta) = \mathbb{E}_{(w_t, s_t) \sim \mathcal{D}} \left[\frac{1}{2} (Q_\theta(s_t, w_t) - (r(s_{t+1}, a_{t+1}) + \gamma \mathbb{E}_{s_{t+1} \sim p} [V_{\tilde{\theta}}(s_{t+1})]))^2 \right] \quad (3.11)$$

where $V_{\hat{\theta}}(s_{t+1})$ is derived from $Q_{\hat{\theta}}(s_t, w_t)$ according to Equation 3.7. The corresponding one-step off-policy updates target g_t :

$$g_t = r(s_{t+1}, a_{t+1}) + \gamma \left((1 - \beta_{\phi}(s_{t+1}, w_t)) Q_{\theta}(s_{t+1}, w_t) + \beta_{\phi}(s_{t+1}, w_{t+1}) \max_w Q_{\theta}(s_{t+1}, w) \right) \quad (3.12)$$

The policy parameters can be learned by directly minimizing the expected KL-divergence:

$$J_{\pi}(\psi) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[\mathbb{E}_{w_t \sim \pi_{\psi}} [\alpha \log \pi_{\psi}(w_t | s_t) - Q_{\theta}(w_t, s_t)] \right] \quad (3.13)$$

Reparameterize

The Q-function, which is represented by a neural network and can be differentiated, and it is thus convenient to apply the reparameterization trick instead, resulting in a lower variance estimator. To that end, we reparameterize the policy using a neural network transformation:

$$\omega_t = f_{\psi}(\epsilon_t; \mathbf{s}_t) \quad (3.14)$$

We use a squashed Gaussian policy, which means that samples are obtained according to

$$f_{\psi}(\epsilon_t; \mathbf{s}_t) = \tanh(\mu_{\psi}(s_t) + \sigma_{\theta}(s_t) \odot \epsilon_t), \epsilon_t \sim \mathcal{N}(0, I) \quad (3.15)$$

Update alpha

The optimal policy at time t is a function of the dual variable α_t :

$$\alpha_t^* = \arg \min_{\alpha_t} \mathbb{E}_{w_t \sim \pi_{\psi}^*} \left[-\alpha_t \log \pi_{\psi}^*(w_t | s_t) - \alpha_t \tilde{H} \right] \quad (3.16)$$

\tilde{H} is target entropy seen as a hyperparameter. The above equations are proved in the paper[13].

In our case, once the w_t is determined by high-level policy $\pi_{\Omega}(w_t | s_t, w_{t-1})$.

3.3.4 Learning Termination

Termination Gradient

According to the option-critic architecture[3], the termination gradient is

$$-\sum_{s_{t+1}, \omega_t} \mu_{\Omega}(s_{t+1}, \omega_t | s_1, \omega_0) \frac{\partial \beta_{\phi}(s_{t+1}, \omega_t)}{\partial \phi} A_{\Omega}(s_{t+1}, \omega_t) \quad (3.17)$$

In our implement, we consider only one step transition, it can be

$$-\sum_{s_{t+1}, \omega_t} \mu_{\Omega}(s_{t+1}, \omega_t | s_t, \omega_{t-1}) \frac{\partial \beta_{\phi}(s_{t+1}, \omega_t)}{\partial \phi} A_{\Omega}(s_{t+1}, \omega_t) \quad (3.18)$$

where $A_{\Omega}(s_{t+1}, \omega_t) = Q_{\theta}(s_{t+1}, \omega_t) - V(s_{t+1})$ denotes the advantage function. Falls $Q > V$, it means the current option is better than other options, the $\beta_{\phi}(s, \omega)$ would be decreased. Falls $Q < V$, it means the current option should be more likely to terminate and the $\beta_{\phi}(s, \omega)$ would be increased.

Implement Termination Gradient

We use a Monte Carlo estimator over the data in the replay buffer D to approximate the termination gradient in Equation 3.18. The extracted data $(s_t, w_t, r(s_t, a_t), s_{t+1})$ exactly consists of single-step transition. We used a mini-batch approach to update the gradients, with the batch size set as a hyperparameter, which can be found in Table 4.4.

$$-\mathbb{E}_{(s_{t+1}, w_t) \sim D} \left[\frac{\partial \beta_{\phi}(s_{t+1}, w_t)}{\partial \phi} A_{\Omega}(s_{t+1}, w_t) \right] \quad (3.19)$$

The advantage function A_{Ω} is defined as:

$$A_{\Omega}(s_{t+1}, w_t) = Q_{\theta}(s_{t+1}, w_t) - V(s_{t+1}) + v \quad (3.20)$$

where the $v > 0$ is the advantage bonus, which encourages the termination critic to keep the previous option ω_{t-1} rather than terminating it, when the Q_{θ} and V are close. The value of *advantage bonus* can be seen in Table 4.4.

The $V(s_{t+1})$ according to 3.7 can be approximated using Monte Carlo estimator

$$V(s_{t+1}) = \mathbb{E}_{\omega_{t+1} \sim \pi_{\Omega}(w_{t+1}|s_{t+1}, w_t)} [Q_{\theta}(\omega_{t+1}, s_{t+1})] \quad (3.21)$$

where the number of sampled ω set as a hyperparameter *number of MC samples* in Table 4.4. The policy π_{Ω} is determined according to 3.8, which can sample a new option or keep the previous option.

3.3.5 Pseudo Code of High Level Training

The following pseudo code provides a detailed illustration of the algorithm's process, where θ parameterizes the Q-function, ψ parameterizes the policy and ϕ parameterizes the termination critic.

Algorithm 1 Termination Soft Actor-Critic under OCAD

Input: $\theta_1, \theta_2, \psi, \phi$ ▷ Initial parameters
 $\tilde{\theta}_1 \leftarrow \theta_1, \tilde{\theta}_2 \leftarrow \theta_2$ ▷ Initialize target network weights
 $\mathcal{D} \leftarrow \emptyset$ ▷ Initialize an empty replay pool
for each iteration **do**
 for each environment step **do**
 $w_t \sim \pi_{\Omega}(w_t|s_t)$ ▷ Sample high-level option from the policy
 $a_t \sim \pi(a_t|w_t)$ ▷ Get deterministic low-level action
 $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$ ▷ Sample transition from the environment
 $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, w_t, r(s_t, a_t), s_{t+1})\}$ ▷ Store the transition in the replay pool
 end for
 for each gradient step **do**
 $\theta_i \leftarrow \theta_i - \lambda_Q \nabla_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$ ▷ Update the Q-function parameters
 $\psi \leftarrow \psi - \lambda_{\pi} \nabla_{\psi} J_{\pi}(\psi)$ ▷ Update policy weights
 $\alpha \leftarrow \alpha - \lambda_{\alpha} \nabla_{\alpha} J(\alpha)$ ▷ Adjust temperature
 $\phi \leftarrow \phi - \lambda_{\phi} \frac{\partial \beta_{\omega, \phi}(s')}{\partial \phi} A(s', w)$ ▷ Update termination function weights
 $\tilde{\theta}_i \leftarrow \tau \theta_i + (1 - \tau) \tilde{\theta}_i$ for $i \in \{1, 2\}$ ▷ Update target network weights
 end for
end for
Output: $\theta_1, \theta_2, \phi, \psi$ ▷ Optimized parameters

3.4 Curriculum Learning

In the CL experiment, we decomposed the final task into multiple simpler subtasks, gradually increasing the training difficulty to make the training process more efficient. Based on our ultimate goal of scoring, we adopted two approaches to design the curriculum experiments, namely the curriculum line and curriculum reward. Both curriculum tasks aim at hitting puck much closer to the goal. The curriculum steps are set to 5 in our thesis.

- **Curriculum Line:** This approach involves a dynamic line setting on the opponent's side above the table. Every time the puck across the line and its velocity towards the opponent, we give it an additional bonus. We also evaluate the success rate of crossing the line. If the success rate exceeds 0.7, the line would shift towards opponent for one shift step, until arriving at target line, which is very close to the table edge. A more intuitive explanation can be found in Fig. 3.2.
- **Curriculum reward:** This approach adjusts the reward as curriculum form:

$$r = \eta * r_{\text{hit}} + (1 - \eta) * r_{\text{cross line}}$$

where $r_{\text{cross line}}$ is the reward of crossing the fixed line in Fig. 3.2. The η is the curriculum coefficient, ranging in $\{0.9, 0.7, 0.5, 0.3, 0.1\}$. Every time the success rate of crossing the line exceeds a value, the value of η would decrease, leads to the weight of r_{hit} decrease, the weight of $r_{\text{cross line}}$ increase.

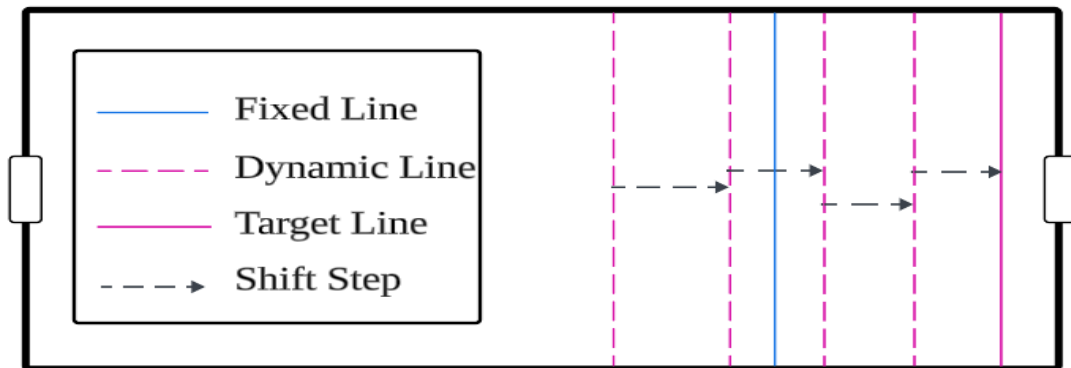


Figure 3.2: Setting of curriculum learning shown above table.

3.5 Self Learning

In the self-learning method, we first create an opponent list that includes all potential opponents. During training, each time a new episode begins, we randomly select an opponent from this list. At the end of each epoch, the agent trained in the previous epoch is added to the opponent list. This way, we gradually build a continuously evolving set of opponents. It is worth to notice that we keep the baseline agent forever, to maintain the diversity of opponents and the stability of training.

Another important point to note is that we need to pre-train a well-performing agent before continuously updating the opponents, otherwise, the opponents will not be sufficient to play games.

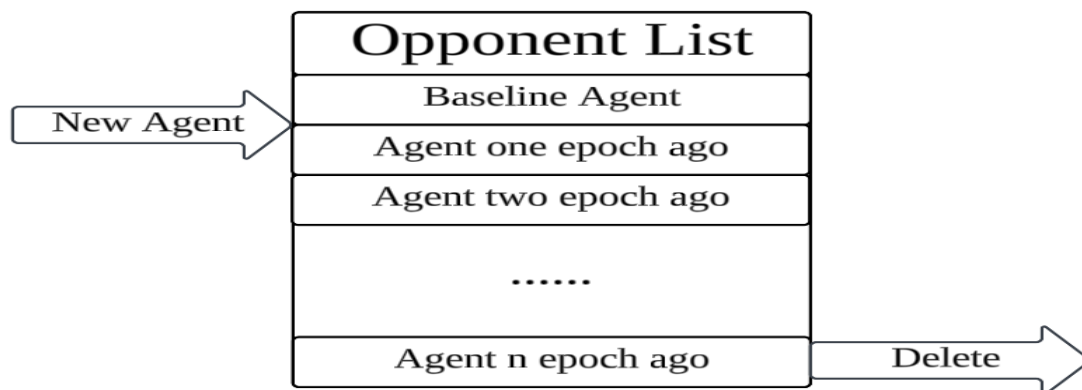


Figure 3.3: Opponent list and its update

4 Experiment

4.1 Settings

4.1.1 Experiment Overview

The names of all trained high-level agents evaluated in this chapter and their relevant information are shown in the following table 4.1, the relevant reward is shown in 4.2.

	Curriculum	Self learn	Opponent	Epoch	Pre-train
Origin	×	×	fixed two	200	×
SL-Origin	×	✓	updating	200	×
CL-Line	✓	×	fixed two	200	✓
SL-CL-Line	✓	✓	updating	200	✓
CL-R	✓	×	fixed two	200	✓
SL-CL-R	✓	✓	updating	200	✓

Table 4.1: Comparison of different agents across various attributes. SL stands for self learning and CL stands for curriculum learning.

where *pre-trained* in 4.1 means the first 100 epochs are trained using the same method as the origin agent. After the 100th epoch, training is conducted using the corresponding setting. Every epoch contains 20,000 steps. The opponent is also needed to be cleared, that *fixed two* is baseline and a TSAC trained agent. The baseline is provided by the *Robot Air Hockey Challenge 2023* competition[1]. It is a manually programmed agent with well-established and stable attack, defense, and preparation strategies. Therefore, it is regarded as the initial and most important training opponent.

Furthermore, different agents may utilize distinct reward structures during training. In

our implementation, we employ three types of reward systems, as presented in Table 4.2. One notable reward mechanism is the cross dynamic line bonus, where a line is set on the opponent’s side of the field, as referred to in section 3.4. A bonus is awarded whenever the puck crosses this line due to the training agent’s hitting. This line can be dynamically shifted as the agent’s success rate of crossing the line increases. This mechanism encourages the agent to focus on improving its offensive capabilities in a structured way.

The hyperparameter η , as defined in Table 4.2, has a range of $\{0.9, 0.7, 0.5, 0.3, 0.1\}$ and serves as a curriculum learning parameter. The more details are shown in Section 3.4.

	Reward
Origin	Base reward + goal – lose
SL-Origin	Base reward + goal – lose
CL-Line	Base reward + Cross dynamic line bonus + goal – lose
SL-CL-Line	Base reward + Cross dynamic line bonus + goal – lose
CL-R	η * Base reward + $(1 - \eta)$ * Cross fixed line bonus + goal – lose
SL-CL-R	η * Base reward + $(1 - \eta)$ * Cross fixed line bonus + goal – lose
Base reward	Hit puck bonus + Initial velocity bonus of hit puck

Table 4.2: Compare the reward of different agent

4.1.2 Simulation Environment

The observation space is 20-dimensions, as shown in Table 4.3.

Observation	Description
Puck’s X-Y Position, Yaw Angle	$[x, y, \theta]$
Puck’s Velocity	$[\dot{x}, \dot{y}, \dot{\theta}]$
Joint Position / Velocity	$[q_1, q_2, q_3, q_4, q_5, q_6, q_7, \dot{q}_1, \dot{q}_2, \dot{q}_3, \dot{q}_4, \dot{q}_5, \dot{q}_6, \dot{q}_7]$

Table 4.3: Observation space

The simulation environment is set up strictly following the real-world arrangement to facilitate future migration. The dimensions of the table are shown in Fig. 4.1, and the placement coordinates of the iiwa are given in Fig. 4.2.

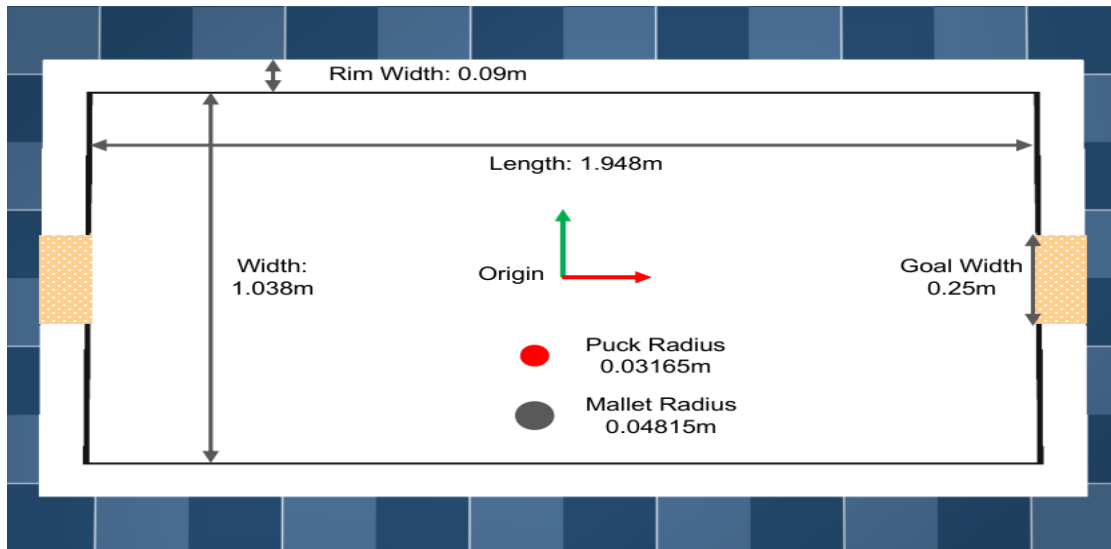


Figure 4.1: Air hockey table

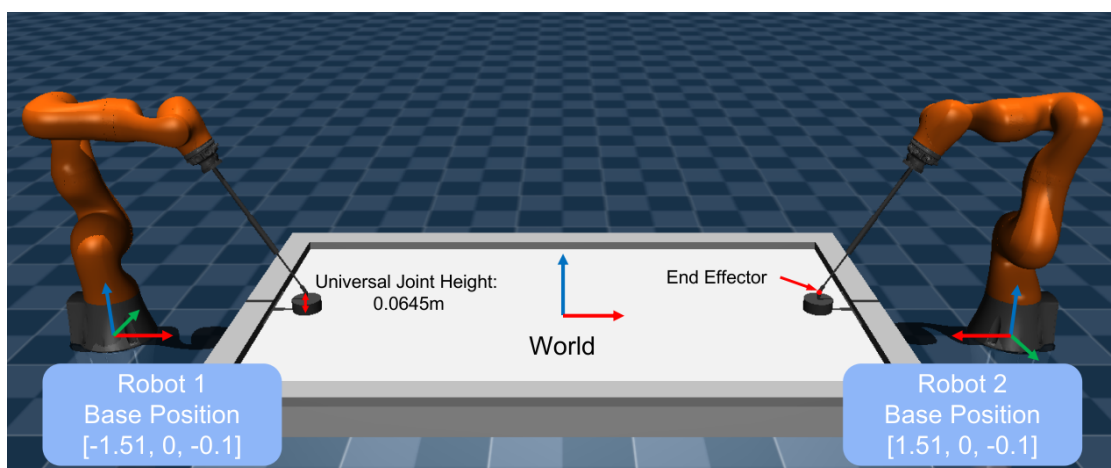


Figure 4.2: Kuka iiwa environment

4.1.3 Hyperparameters

The hyperparameters are shown in Table 4.4. The *adv bonus* is an additional term in the advantage function A_Ω in Equation 3.18, designed to ensure that when Q and V are close, the value of A_Ω remains greater than zero. This encourages the agent to continue using the existing option rather than sampling a new one. The *number of MC samples* represents the number of sampled w_{t+1} when applying the Monte Carlo estimator in Equation 3.21. *Update target network weight* and *target entropy* are parameters used in the SAC algorithm. The *warmup* settings mean parameter updates begin only after accumulating a certain number of warmup steps in the replay buffer, which enhances learning stability.

Table 4.4: Hyperparameters in experiment

Algorithm-related			
Adv bonus	Number of MC samples	Update target network weight	Target entropy
0.1	50	0.003	-2
Replay Buffer Size and Warm-up Settings			
Max replay size	Critic warmup	Actor warmup	Termination warmup
1000000	20000	20000	20000
Gradient Updating			
Batch size	Alpha learning rate	Actor learning rate	Critic learning rate
256	0.00001	0.0003	0.0003
Termination learning rate		0.00001	
MDP Settings			
Maximum horizon of one episode	Gamma		
600	0.995		
Features of Neural Network			
Critic	Actor	Termination critic	
256 256 256	256 256 256	256 256 256	
Others			
Curriculum step	Size of opponent buffer in SL		
5	4		

4.2 Evaluation

4.2.1 CNP-B

Setting constraint of CNP-B

The training method refers to Section 3.2.1. Here we want to show two training curves of total loss in Fig. 4.3. The total loss is the sum of the losses from all the constraints during training. Both experiments use the same dataset. The difference between Experiment 1 and Experiment 2 is the strictness of the constraints.

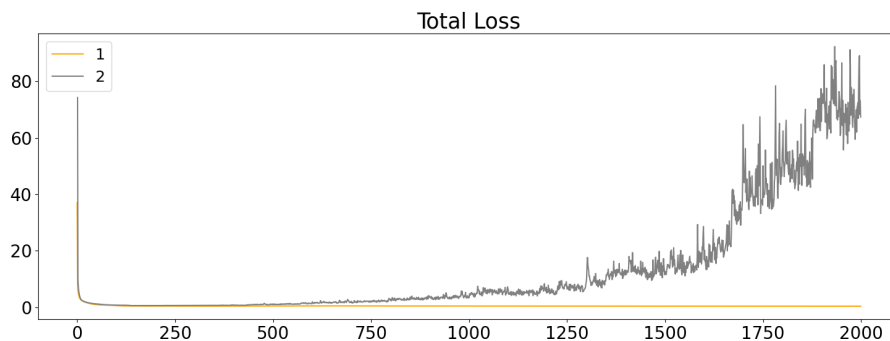


Figure 4.3: Total loss of CNP-B in training

In both experiments, the loss initially drops sharply to near zero. However, in Experiment 2, the total loss quickly rises again during subsequent training.

The reason for the difference is that Experiment 2 failed to meet the stricter constraint settings, resulting in the inability to reduce the loss, which was continuously amplified as the weight coefficients increased. There was no significant difference in the actual performance of the agents from Experiment 1 and Experiment 2. Therefore, we believe that overly strict constraint settings in the CNP-B training method can cause the loss to fail to converge but may not have a significant impact on actual performance.

Implement CNP-B in Our Method

In the OCAD framework, we can arbitrarily replace the low-level agent as long as it can generate smooth trajectory actions based on high-level commands. We first used a CNP-B agent as the low-level agent for a simple serve task, the agent needs to serve a stationary puck from our half side to the opponent's half.

We first run a normal experiment, see 4.4 for origin curve. The β values are updated but the reward value is very unstable, and inversely proportional to the number of terminations. Then we limit the terminations, the reward appears normal, as shown by the green curve in 4.4.

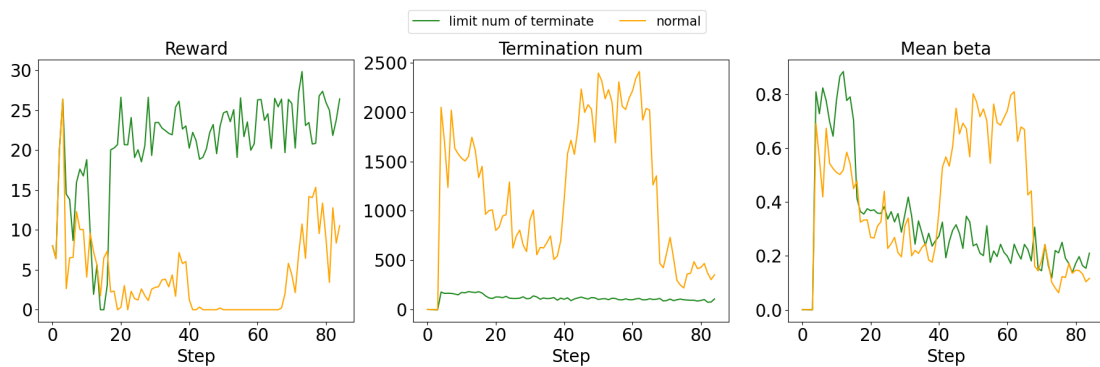


Figure 4.4: Serve task using CNP-B

The algorithm is working but the performance is highly dependent on termination numbers. The higher β leads to more terminations and the performance becomes poor. The reason is that the CNP-B agent doesn't have the ability to generate smooth trajectory during lots of terminations. That will cause the robotic arm to stop working.

Improve CNP-B

Our approach is to improve the dataset. The corresponding method No. 2, No. 3 and No. 4 can be found in Section 3.2.1. The final trained agent showed some improvement but still couldn't be reliably applied to a high-level agent with frequent terminations. We

observed that in certain termination cases, the joint velocity of the robot in the simulation exceeded the limit, which could be a key reason for the abnormal trajectory planning.

Conclusion

The CNP-B can generate trajectories that meet the requirements. However, within the framework of our method, it cannot adapt well to situations with frequent terminations. Thus in the subsequent experiments, we employ the ATACOM agent as low-level agent.

4.2.2 Hierarchical Learning

Here we aim to compare the differences between reinforcement learning training using the OCAD structure of HRL and reinforcement learning training without the HRL. The main differences between the two are shown as Fig. 4.5.

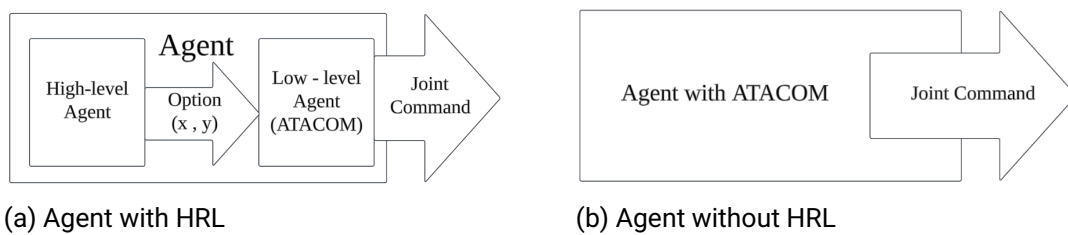


Figure 4.5: Comparison between HRL and non-HRL reinforcement learning

The reward graph shows that the hierarchical model consistently outperforms the non-hierarchical model, achieving significantly higher rewards and maintaining a more stable performance. In terms of scores, the hierarchical model also displays a faster and more consistent increase, indicating a more efficient learning process. Meanwhile, the losses graph reveals that the non-hierarchical model accumulates losses at a faster rate, while the hierarchical model keeps losses relatively low, reflecting its better ability to optimize and learn from the environment.

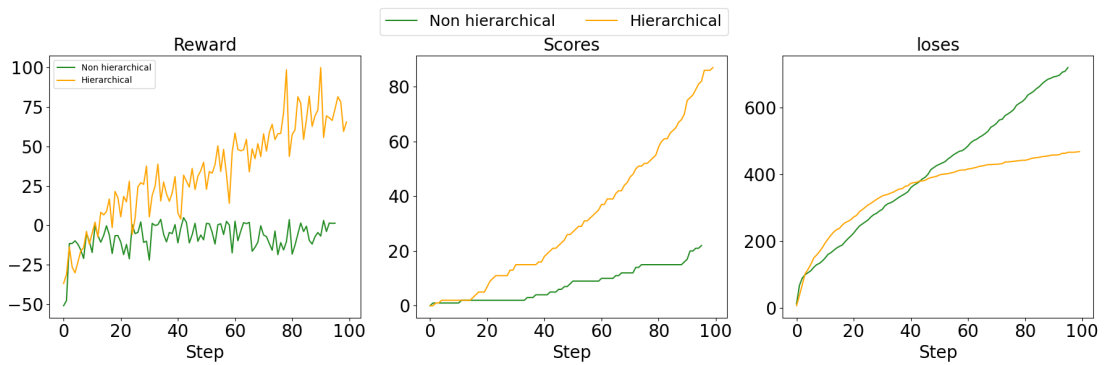


Figure 4.6: With or without hierarchical structure

Conclusion

The comparison in the Fig. 4.6 clearly demonstrates the superiority of the hierarchical reinforcement learning approach. This is because the dimension of action space is highly reduced. The action space of the high level agent in OCAD has only two dimensions, compared with 7 dimensions of joint space without HRL structure. This greatly reduces the difficulty of the learning task, improves sampling efficiency, and makes the result more explainable.

4.2.3 Different Termination Probability

Setting

Termination condition is a crucial part of our algorithm, a perfect termination condition makes the agent to change its current option to a better option immediately. We ran four experiments with different settings of termination condition:

Experiments	exp 1	exp 2	exp 3	exp 4
Beta Values(β)	0.1	0.5	0.9	updating beta

Table 4.5: Beta settings

The β of Exp 4 is not fixed, updated using the termination gradient 3.18. The beta values β means the probability of terminating the current option. Lower β means the lower terminate probability at current option-state pair.

Analysis

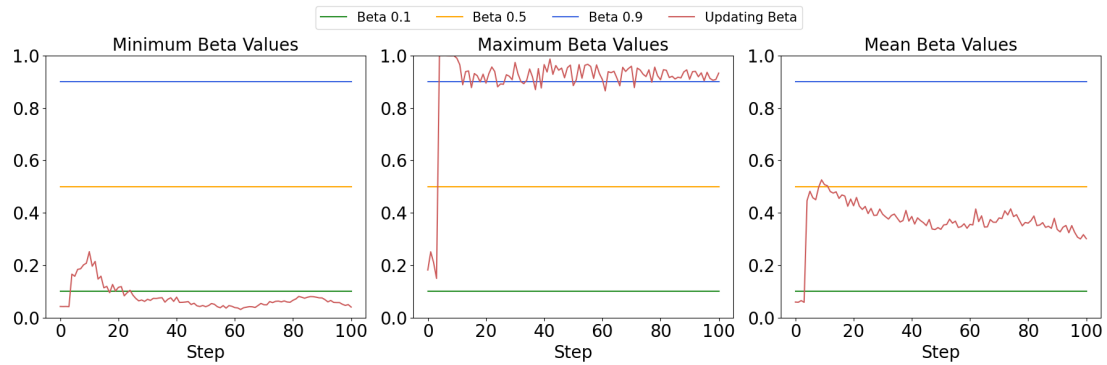


Figure 4.7: Different Beta Values in different experiments

From figure 4.7 we observe the updating beta values, compared with other fixed β . At different state-option pair, the value of beta exhibits a polarization phenomenon, with its mean close to 0.4.

Based on a comprehensive observation of figures 4.8, the performance of $\beta = 0.1$ is relatively poor, while the other three groups are close to each other. The reward curve shows a convergence trend. The slope of the score curve increases as the number of iterations increases, whereas the slope of the concede curve decreases. In the case of target entropy is -2, the entropy values of all four experimental groups tend to approach -2, ranging between -1 and -2.

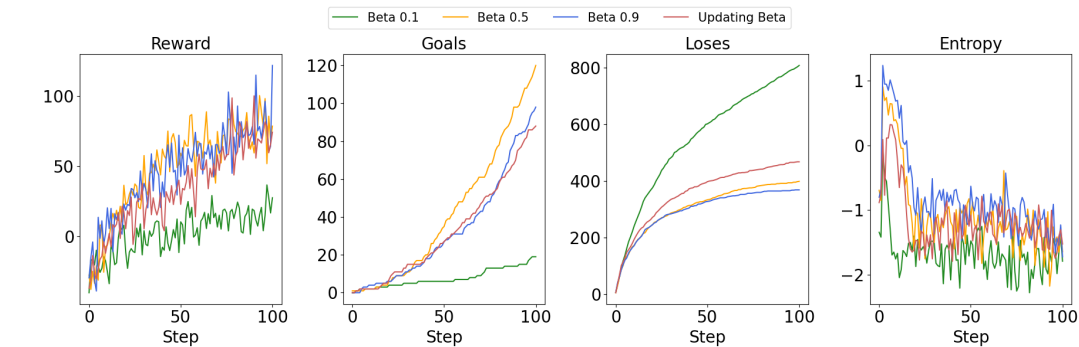


Figure 4.8: Reward, scores, concedes and entropy

Conclusion

In context of air hockey games. It is necessary to have termination condition while the option contains multiple action steps in our structure, we need termination to adapt dynamic environment. With termination gradient we truly result a good termination critic.

4.2.4 Curriculum Learning

Curriculum learning is an approach in which the learning process is structured by progressively increasing the complexity of the tasks the agent faces. By starting with simpler tasks and gradually introducing more challenging ones, the agent can build foundational skills before confronting more complex scenarios.

Here we run two experiment as the section 3.4 shows. In Fig. 4.9, the origin model has the most goals, but the slope in scores of all three agents are fast the same. The slope and the total number of loses also don't have much difference. That means the performance on goals and loses might be no significant difference by using the curriculum algorithm. It is important to note that the both curriculum task don't arrive at the final curriculum step, they are both stuck at the second step.

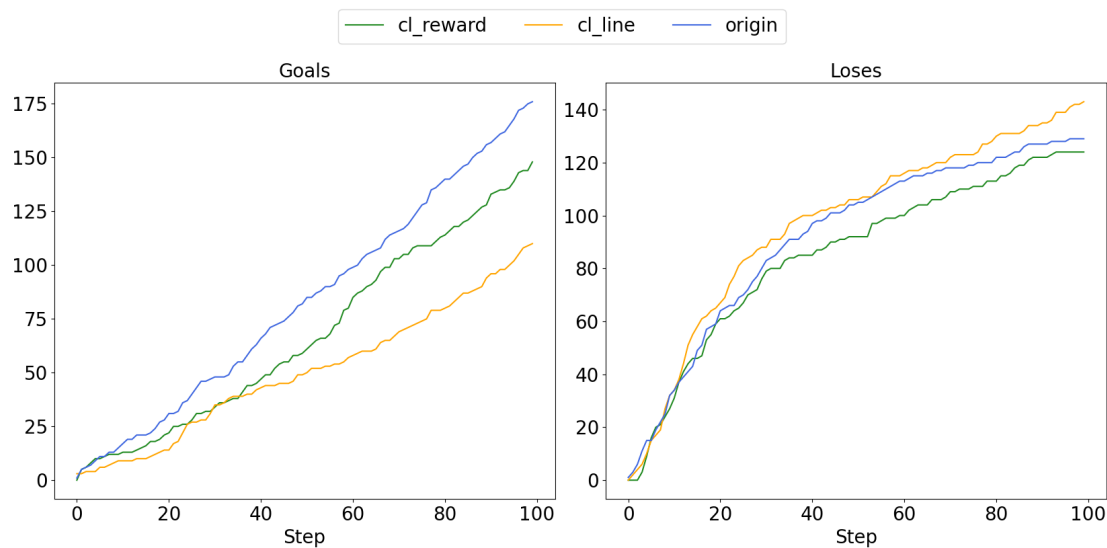


Figure 4.9: Scores and concedes in curriculum experiment

Conclusion

Based on the training curves, using the CL method did not show a significant improvement in performance. Moreover, since the curriculum tasks did not progress to the end, this indicates that there is room for improvement in the design of the CL experiment. It is necessary to identify more appropriate subtasks or progression criteria.

4.2.5 Self Learning

In our agent training approach, we utilize a unique self-learning method by setting up an opponent list to store different versions of the agent and periodically update it, the setting details refer to section 3.5.

In Fig. 4.10 shows the results of self-learning (SL) methods versus non-self-learning methods. We see that the self-learning agents (denoted with sl prefixes) generally perform better in goals. On the total number of losses, we observe that the sl-origin agent accumulates losses at a much faster rate compared to the other models. Based on the figure alone, the agent in training implementing the self-learning method performs slightly worse in

terms of losses, with a higher number of losses and a steeper slope.

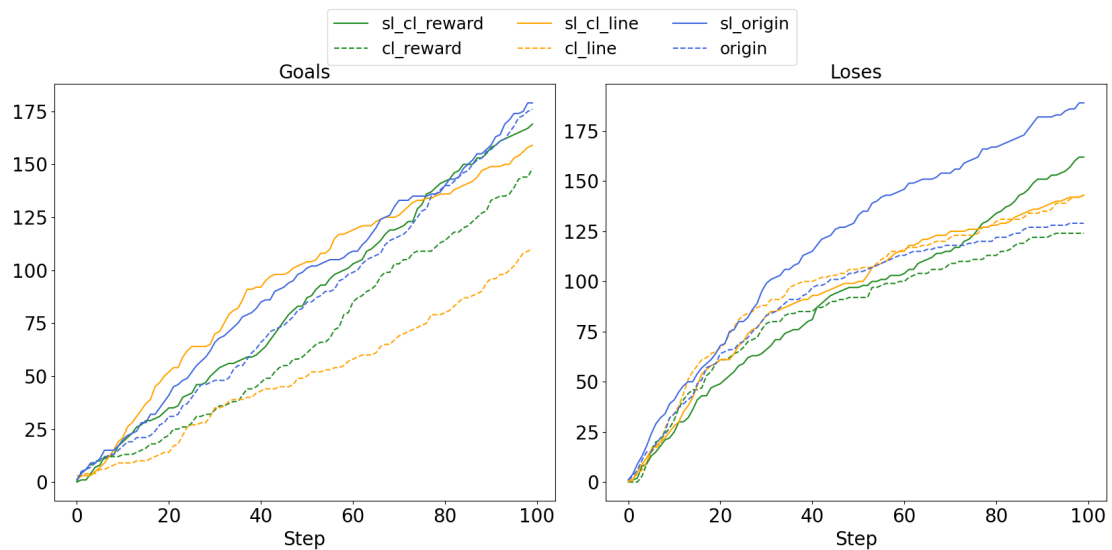


Figure 4.10: Goals and loses in self learn training

Conclusion

The agent that used the SL method performed slightly better than the one that did not use the SL method. Considering that during the training process the opponents are also continuously updating and becoming stronger, we infer that the self learning agent would be more powerful in actual competition. Therefore, the SL method does indeed help improve the agent's performance.

4.2.6 Play Games

Against baseline

Here, we compared the number of goals and loses between different training agents and the baseline. The data is averaged based on simulations over 120,000 steps with three

different seeds, which is equivalent to the average result of three 40-minute matches in the real world. The result shows in Fig. 4.11.

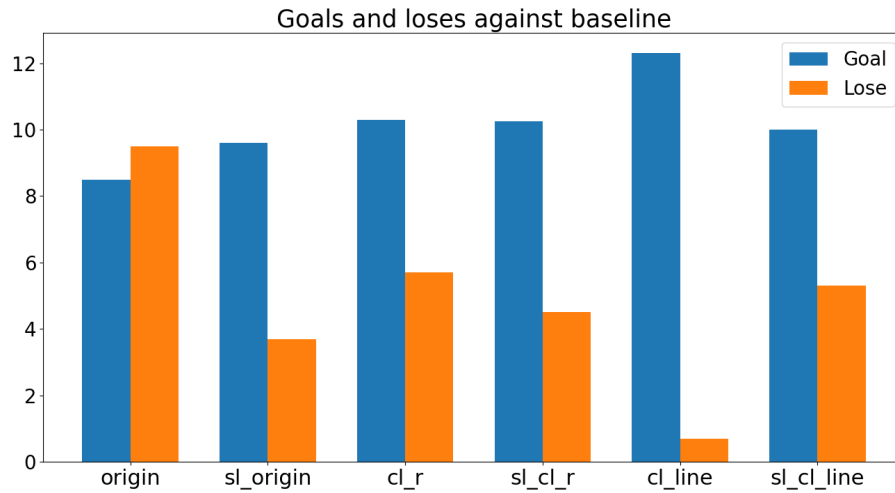


Figure 4.11: Goals and loses against baseline

From the results, it can be seen that the sl-origin method, which uses the SL approach, shows a significant improvement in defensive performance. Similarly, the sl-cl-r method also enhances the defensive ability of cl-r. The overall performance of cl-line is the most outstanding, indicating that the related experimental configuration is most suitable for training against a fixed opponent.

Against each other

The data in the Fig. 4.12 and 4.13 represents the average values obtained after running three matches, each equivalent to 40 minutes in the real world. From the matches among the six agents, it is evident that the sl-cl-line agent stands out, based on the scores alone, achieving a decisive victory against all five other agents. Agents using the SL method were consistently able to defeat those that did not employ SL. Agents using only the CL method did not show a clear advantage over those without CL. The cl-line agent seemed to perform worse than the cl-r agent, but the sl-cl-line agent outperformed the sl-cl-r agent.

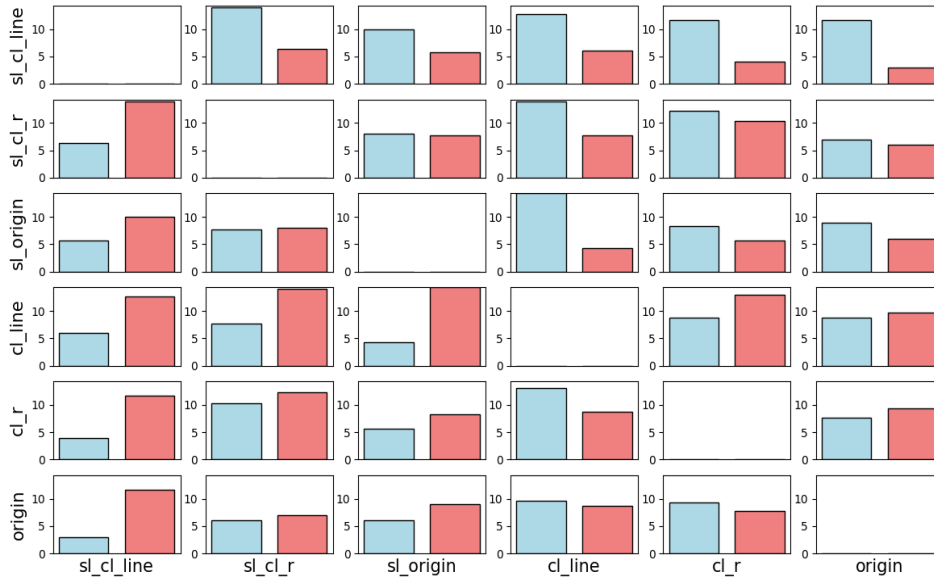


Figure 4.12: Agents against each other. The blue bars represent the goals scored by the y-axis agent, while the red bars represent the goals scored by the x-axis agent.

If the agent fails to hit the puck from its own half to the opponent's half within 5 seconds, it is recorded as a fault. A higher number of faults indicates poorer performance in the match. The proportion of faults for each agent relative to the total number of episodes is shown in the Fig. 4.13. The agent sl-cl-line and sl-cl-r have the minimum rate, while the cl-line has the maximum rate. The performance of the other agents does not differ significantly.

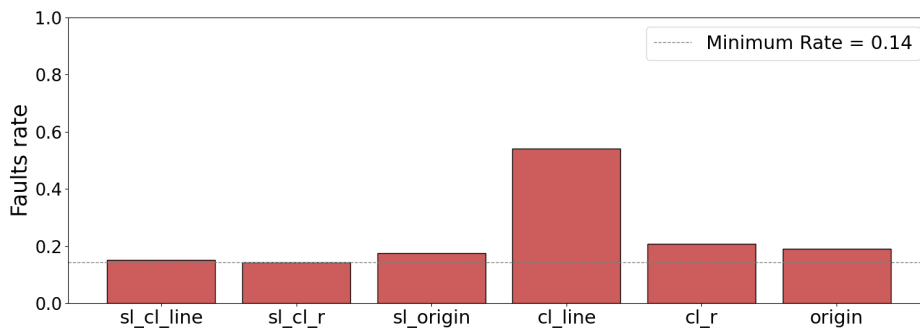


Figure 4.13: Agents' faults rate

Conclusion

The sl-cl-line agent exhibited top-tier performance, significantly outperforming the other agents in the matches. The sl-cl-r agent also showed outstanding performance compared to the others. Agents trained using SL, CL, or a combination of both methods demonstrated superior capabilities compared to those that did not employ these methods. Therefore, the use of SL and CL methods can substantially enhance the agent's performance.

Compared to their matches against the baseline, agents that did not use the SL method experienced a significant drop in performance when competing against unseen opponents. This further demonstrates that the SL method can significantly enhance generalization.

5 Conclusion and Future work

Combining all the experiments, we arrived at the following conclusions:

- **1:** Our proposed TSAC algorithm within the OCAD framework has been successfully validated in the air hockey environment. The OCAD structure truly simplifies the learning task and the TSAC efficiently trains the continuous options and performs well with the termination critic.
- **2:** CNP-B can be trained to generate trajectories that meet the constraints. However, it fails to function properly in our method when multiple terminations occur. This failure is due to the abnormal joint velocities observed at the termination points.
- **3:** The well-performing ATACOM agent meets our requirements.
- **4:** Curriculum learning can significantly improve training outcomes. However, the CL experiment we designed did not complete all the curriculum steps.
- **5:** Self-learning not only significantly improves the agent’s performance but also enhances generalization, allowing it to perform well even when facing previously unseen opponents.

Our future work can be carried out as follows:

- **1:** Compare our method with other existing HRL approaches.
- **2:** Validate our approach on more platforms.
- **3:** Further explore new CL experiment designs.
- **4:** Continue to explore new self-learning methods, such as using multiple agents for parallel training to diversify the opponents in the opponent list.

Bibliography

- [1] *Air Hockey Challenge*. <https://air-hockey-challenge.robot-learning.net/home>. Accessed: 2024-09-21.
- [2] Ahmad AlAttar, Louis Rouillard, and Petar Kormushev. “Autonomous Air-Hockey Playing Cobot Using Optimal Control and Vision-Based Bayesian Tracking”. In: *Towards Autonomous Robotic Systems*. Ed. by Kaspar Althoefer, Jelizaveta Konstantinova, and Ketao Zhang. Cham: Springer International Publishing, 2019, pp. 358–369. ISBN: 978-3-030-25332-5.
- [3] Pierre-Luc Bacon, Jean Harb, and Doina Precup. “The Option-Critic Architecture”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. AAAI Press. 2017, pp. 1726–1734.
- [4] Andrew G Barto and Sridhar Mahadevan. “Recent advances in hierarchical reinforcement learning”. In: *Discrete Event Dynamic Systems* 13.4 (2003), pp. 341–379.
- [5] Marc Bellemare et al. “Unifying count-based exploration and intrinsic motivation”. In: *Advances in neural information processing systems* 29 (2016).
- [6] Bradley E Bishop and Mark W Spong. “Vision-based control of an air hockey playing robot”. In: *IEEE Control Systems Magazine* 19.3 (1999), pp. 23–32.
- [7] Petros Christodoulou, Curtis Twomey, and Doina Precup. “Soac: The soft option actor-critic architecture”. In: *Proceedings of the 38th International Conference on Machine Learning*. 2021.
- [8] Thomas G Dietterich. “Hierarchical reinforcement learning with the MAXQ value function decomposition”. In: *Proceedings of the 17th International Conference on Machine Learning (ICML)*. 2000, pp. 232–240.
- [9] Jeffrey L Elman. “Learning and development in neural networks: The importance of starting small”. In: *Cognition* 48.1 (1993), pp. 71–99.

-
-
- [10] Carlos Florensa, Yan Duan, and Pieter Abbeel. “Stochastic neural networks for hierarchical reinforcement learning”. In: *arXiv preprint arXiv:1704.03012* (2017).
- [11] Roy Fox et al. “Multi-level discovery of deep options”. In: *arXiv preprint arXiv:1703.08294* (2017).
- [12] Abhishek Gupta et al. “Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning”. In: *arXiv preprint arXiv:1910.11956* (2019).
- [13] Tuomas Haarnoja et al. “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor”. In: *Proceedings of the 35th International Conference on Machine Learning (ICML)*. 2018, pp. 1861–1870.
- [14] Jean Harb et al. “Waiting for the right time: When to commit to action in model-based reinforcement learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.
- [15] Anna Harutyunyan, Nicolas Vieillard, Doina Precup, et al. “The natural option critic”. In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2019, pp. 3093–3102.
- [16] Kazuki Igeta and Akio Namiki. “Algorithm for optimizing attack motions for air-hockey robot by two-step look ahead prediction”. In: *2016 IEEE/SICE International Symposium on System Integration (SII)*. IEEE. 2016, pp. 465–470.
- [17] Lydia E Kavraki et al. “Probabilistic roadmaps for path planning in high-dimensional configuration spaces”. In: *IEEE transactions on Robotics and Automation* 12.4 (1996), pp. 566–580.
- [18] Piotr Kicki et al. “Fast kinodynamic planning on the constraint manifold with deep neural networks”. In: *IEEE Transactions on Robotics* (2023).
- [19] Vijay R. Konda and John N. Tsitsiklis. “Actor-Critic Algorithms”. In: *Advances in Neural Information Processing Systems*. Vol. 13. MIT Press, 2000, pp. 1008–1014.
- [20] Tejas D Kulkarni et al. “Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation”. In: *Advances in neural information processing systems* 29 (2016).
- [21] Steven M LaValle and James J Kuffner. “Rapidly-exploring random trees: Progress and prospects: Steven m. lavalley, iowa state university, a james j. kuffner, jr., university of tokyo, tokyo, japan”. In: *Algorithmic and computational robotics* (2001), pp. 303–307.

-
-
- [22] Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. “Transfer in reinforcement learning: a framework and a survey”. In: *Proceedings of the 9th European Conference on Machine Learning and Knowledge Discovery in Databases*. 2008, pp. 19–24.
- [23] Andrew Levy et al. “Learning multi-level hierarchies with hindsight”. In: *arXiv preprint arXiv:1712.00948* (2017).
- [24] Puze Liu et al. “Efficient and reactive planning for high speed robot air hockey”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 586–593.
- [25] Puze Liu et al. “Safe Reinforcement Learning on the Constraint Manifold: Theory and Applications”. In: *arXiv preprint arXiv:2404.09080* (2024).
- [26] Tamber Matiiisen et al. “Teacher-student curriculum learning”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 2017, pp. 4930–4940.
- [27] Ofir Nachum et al. “Data-efficient hierarchical reinforcement learning”. In: *Advances in neural information processing systems* 31 (2018).
- [28] Ofir Nachum et al. “Why does hierarchy (sometimes) work so well in reinforcement learning?” In: *arXiv preprint arXiv:1909.10618* (2019).
- [29] Akio Namiki et al. “Hierarchical processing architecture for an air-hockey robot system”. In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 1187–1192.
- [30] Sanmit Narvekar and Matthew E Taylor. “Source task creation for curriculum learning”. In: *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*. 2016, pp. 566–574.
- [31] Sanmit Narvekar et al. “Learning curriculum policies for reinforcement learning”. In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. 2019, pp. 25–33.
- [32] Shubham Pateria et al. “Hierarchical reinforcement learning: A comprehensive survey”. In: *ACM Computing Surveys (CSUR)* 54.5 (2021), pp. 1–35.
- [33] Deepak Pathak et al. “Curiosity-driven exploration by self-supervised prediction”. In: *International conference on machine learning*. PMLR. 2017, pp. 2778–2787.
- [34] Ahmed Hussain Qureshi et al. “Motion planning networks: Bridging the gap between learning-based and classical motion planners”. In: *IEEE Transactions on Robotics* 37.1 (2020), pp. 48–66.

-
-
- [35] Nathan Ratliff et al. “CHOMP: Gradient optimization techniques for efficient motion planning”. In: *2009 IEEE international conference on robotics and automation*. IEEE. 2009, pp. 489–494.
- [36] Terence D Sanger. “Neural network learning control of robot manipulators using gradually increasing task difficulty”. In: *IEEE International Conference on Robotics and Automation*. IEEE. 1994, pp. 1766–1771.
- [37] John Schulman et al. “Finding locally optimal, collision-free trajectories with sequential convex optimization.” In: *Robotics: science and systems*. Vol. 9. 1. Berlin, Germany. 2013, pp. 1–10.
- [38] Hideaki Shimada et al. “A two-layer tactical system for an air-hockey-playing robot”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 6985–6990.
- [39] Richard S Sutton, Doina Precup, and Satinder P Singh. “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning”. In: *Artificial Intelligence* 112.1-2 (1999), pp. 181–211.
- [40] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: <http://incompleteideas.net/book/the-book-2nd.html>.
- [41] Richard S. Sutton et al. “Policy Gradient Methods for Reinforcement Learning with Function Approximation”. In: *Advances in Neural Information Processing Systems*. Vol. 13. MIT Press, 2000, pp. 1057–1063.
- [42] Max Svetlik et al. “Automatic curriculum learning through value disagreement”. In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. 2017, pp. 4333–4340.
- [43] Ayal Taitler and Nahum Shimkin. “Learning control for air hockey striking using deep reinforcement learning”. In: *2017 International Conference on Control, Artificial Intelligence, Robotics & Optimization (ICCAIRO)*. IEEE. 2017, pp. 22–27.
- [44] Matthew E Taylor and Peter Stone. “Transfer learning for reinforcement learning domains: A survey”. In: *Journal of Machine Learning Research*. Vol. 10. 2009, pp. 1633–1685.
- [45] Philip S. Thomas. “Bias in Natural Actor-Critic Algorithms”. In: *Proceedings of the International Conference on Machine Learning (ICML)*. PMLR, 2014, pp. 441–448.
- [46] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, et al. “Feudal networks for hierarchical reinforcement learning”. In: *Proceedings of the 34th International Conference on Machine Learning*. 2017, pp. 3540–3549.

-
- [47] Shangtong Zhang and Shimon Whiteson. “DAC: The double actor-critic architecture for learning options”. In: *Advances in Neural Information Processing Systems 32* (2019).