Doctoral Thesis

Optimizing Controllers of Swarm Robotic Systems with Deep Reinforcement Learning

深層強化学習によるスワームロボットのコントローラの最適化

D202275　NIE XIAOTONG

Graduate School of Advanced Science and Engineering
Hiroshima University
September 2023

Abstract

This thesis focuses on optimizing controllers of Swarm Robotics Systems (SRS) with Deep Reinforcement Learning (DRL). While DRL has demonstrated great potential in designing controllers for various static environment tasks, such as video games, it faces challenges in learning effective policies in dynamic environments due to incomplete observability and non-stationarity. SRS is a field that involves the coordination of multiple robots working in a decentralized manner. SRS present a highly dynamic environment since each robot only observes a partial view of the environment, and seen other robots as part of the environment. To address this challenge, one of the efficient ways is to integrate DRL with other algorithms, such as curriculum learning. On the other perspective, enhancing the understanding of DRL through explainable algorithms is also critical. Understanding decision-making processes can discover and improve the potential problems. In this thesis, three contributions are presented to the field of SRS and DRL.

Firstly, this thesis presents a novel automatic curriculum learning method called Self-Teaching Automatic Curriculum Learning. Curriculum learning is a promising solution to the issue that DRL is insufficient for training end-to-end controllers in a dynamic environment. However, traditional manually designed curriculums limit the pace of training and heavily rely on the designer's experience. Therefore, the proposed method integrates robot training with curriculum scheduling in one neural network, which can select the subtask to be trained automatically. The proposed method ensures the neural network learning in an optimal state, improves the controller's performance and efficiency.

Secondly, this thesis presents how DRL is utilized to address a decision-making problem in a multi-autonomous vehicle task, where autonomous vehicles are formulized as a SRS. Environmental vehicles are part of the environment. The positions and actions of environmental vehicles are unpredictable, which makes the environment more dynamic and dangerous. Therefore, it is necessary to equip autonomous vehicles with a security assurance mechanism. The proposed method utilizes time-to-collision (TTC) as the feature representation and proposes a TTC-based safety check system. The action output by the DRL would be replaced with a safer action chosen by the safety check system when an agent detects a potential collision, which can improve the safety of the system.

Thirdly, this thesis presents an Explainable Reinforcement Learning approach. Deconvnet and a saliency map method Grad-CAM are utilized to enhance comprehension of the control strategy for SRS trained by deep Q-network. The proposed approach is able to interpret the control strategy and evaluate the fault tolerance of the controller.

Overall, the proposed methods can optimizing controllers of SRS with DRL, provide promising solutions for addressing complex tasks in dynamic environments that traditional DRL approaches struggle with.

Contents

Chapter 1

Introduction

1.1 Background and Motivation

Swarm intelligence (SI) [1] is a branch of artificial intelligence that studies the collective behavior of a decentralized and self-organized system. SI algorithms draw inspiration from nature, such as ant colonies, bird flocks, fish schools, and bee hives. As shown in Fig. 1.1, SI refers to these highly intelligent activities displayed by these social insects. They engage collective behaviors to complete tasks that are are too difficult for a single individual.

The study of SI's application to multiple robot systems is known as swarm robotic syetsm. Swarm robotic system (SRS) is a field that focuses on the coordination of numerous robots that work decentralized from one another [2][3]. Robots in a SRS are relatively simple since the communication range is general local and sensor capabilities are limited. However, they can perform complex tasks collectively that a single robot cannot. A SRS operates in a self-organizing way, which means there is no supreme central robot dictating to the others, and the robots are not aware of global information. Each robot, on the other hand, just follows simple local rules and behaves autonomously based on its local perspective [4]. As a result, each robot only observes a partial view of the environment and other robots are treated as part of the environment. This characteristic results in a highly dynamic environment.

Swarm robotic system has many advantages over traditional robotics, such as fault tolarance, flexibility, and scalability [5][6]. The characteristics of swarm robotic system system were shown as follows:

- **Fault tolerance**

  Fault tolerance means that robots can cope with internal failures and continue performing their tasks without human intervention. The high redundancy of the swarm enables fault tolerance. Redundancy means that there are more robots than needed

(a) Flock of starlings (by Airwolfhound, licensed under CC BY-SA 2.0).

(b) School of fish (by Sam Howzit, licensed under CC BY 2.0).

Figure 1.1: Examples of collective behavior in biological swarms.

for a task, so if some of them fail, others can take over. Fault tolerance is important because it allows robots to operate in uncertain and dynamic environments where failures are inevitable and unpredictable.

- **Scalability**

  Scalability means that robots can perform well regardless of the number and size of the robots. Ideally, the addition or removal of individuals should not have a significant impact on the swarm's functioning. Scalability is achieved by using decentralized control and local communication. Scalability is important because it allows robots to handle large and variable tasks that may require different numbers of robots.

- **Flexibility**

  Flexibility means that robots can generate modularized solutions to the different tasks. Flexibility is achieved by the distributed and self-organized characteristics: in a SRS, robots flexibly assign themselves to various positions based on the demands of the surrounding environment and operational circumstances. Flexibility also enhances the creativity and diversity of SRS by enabling them to discover new strategies and behaviors that may not be designed or anticipated by human engineers.

Despite these advantages, designing controllers for SRS remains a challenging task due to the highly dynamic and non-stationary nature of the environment. Traditional deep reinforcement learning methods, which have demonstrated success in optimizing controllers for static environments [7], struggle with the incomplete observability and non-stationarity of SRS. To address this challenge, optimization algorithms of the deep

reinforcement learning controller are proposed in this thesis. This research aims to improve the performance of SRS in dynamic environments.

1.2 Thesis Objectives

This thesis focuses on optimizing controllers of SRS with Deep Reinforcement Learning (DRL). In recent years, DRL has demonstrated great potential in designing controllers to various static environment tasks, such as playing video games [7]. However, it is difficult for traditional DRL to learn effective policies in dynamic environments due to the lack of complete observability and the non-stationarity of the environment. SRS is a field that involves the coordination of multiple robots working in a decentralized manner. In SRSs, each robot only observes a partial view of the environment and other robots are treated as part of the environment. This characteristic results in a highly dynamic environment that poses significant challenges for DRL algorithms. To address this challenge, one of the efficient ways is to integrate with other algorithms, such as curriculum learning, to improve the performance of control algorithms. On the other perspective, enhancing the understanding of DRL through explainable algorithms is also critical. By understanding decision-making processes and policy characteristics, potential issues will be discovered and improved. In this thesis, three contributions are presented to the field of SRS and DRL:

The first objective of this thesis is to solve the issue that DRL is insufficient for directly training end-to-end controllers in a dynamic environment. To address this constraint, curriculum learning has emerged as a promising solution. However, a traditional manually designed curriculum limits the pace of training and heavily relies on the designer's experience. Therefore, a novel automatic curriculum learning method called Self-Teaching Automatic Curriculum Learning (STACL) is proposed in this thesis. The proposed algorithm integrates robot training with curriculum scheduling in one neural network. The reward function can calculate the learning rate for different curricula, then select the next subtask to be trained for the next episode. The proposed method is supposed to ensure that the neural network remains in an optimal state for learning, improve the controller's performance and accelerate its convergence speed.

The second objective of this thesis is to utilize DRL to address a decision-making problem in a multi-autonomous vehicle task. In this task, multi-autonomous vehicles will be formulized as a SRS controlled by DRL algorithm. Other traffic participants, i.e, environmental vehicles, are seen as part of the environment. The positions and actions of environmental vehicles are unpredictable, and their movements may affect the decisions of autonomous vehicles controlled by DRL, which makes the environment more dynamic and dangerous. Therefore, it is necessary to equip autonomous vehicles with a security as-

surance mechanism. The proposed method utilizes time-to-collision (TTC) as the feature representation and proposes a TTC-based safety check system. When an agent detects a potential collision, the action output by the DRL controller is replaced by a safer action selected by the safety check system. The proposed method is designed to reduce the collision rate even in a dense traffic situations.

The third objective of this thesis is to develop an explainable reinforcement learning approach. The lack of interpretability problem limits the understanding and optimization of the model's decision-making in dynamic environments. We applied a deep Q-learning algorithm to develop controllers for a SRS that take raw camera images as input. Three experiments are conducted to visualize the policies learned by deep Q-network. The first experiment proposes a network structure with several deconvolutional layers to view the neural network's feature map during various training stages. The second experiment employs a saliency map method Gradient-weighted Class Activation Mapping to determine which state variables the robot attends to during strategy execution. Lastly, the third experiment utilizes a perturbation-based visualization method to evaluate the fault tolerance of the controller. By understanding decision-making processes and policy characteristics, potential issues will be discovered and improved.

1.3 Structure of the Thesis

The overall structure of this thesis is illustrated in Fig. 1.2. In this thesis, all of the experiments are conducted in computer simulations. A summary of each chapter is described as follows.

The rest of this thesis is divided into five chapters:

- Chapter 2 first gives a review of SRS. Then, DRL and several related optimazation techniques are presented.
- Chapter 3 aims to develop a novel Automatic Curriculum Learning method called Self-Teaching Automatic Curriculum Learning. The proposed algorithm integrates robot training with curriculum scheduling in one neural network and make the network remains in an optimal state for learning.
- Chapter 4 describes how DRL is utilized to address a decision-making problem in a multi-autonomous vehicle task. In this task, multi-autonomous vehicles will be formulized as a SRS controlled by DRL algorithm. The proposed method is able to improve the arrival rate and reduce the collision rate effectively .
- Chapter 5 proposes an explainable reinforcement learning approach. Deconvnet and a saliency map method Grad-CAM are utilized to interpret the control strategy trained by deep Q-network.

Figure 1.2: Overview of the thesis structure

- Chapter 6 summarizes this thesis and addresses future research directions.

Chapter 2

Literature Review on Swarm Robotic System and Deep Reinforcement Learning

Swarm Robotics Systems (SRS) is a field that involves the coordination of multiple robots working in a decentralized manner, and can be applied in various domains, such as rescue and transportation. Deep Reinforcement Learning (DRL) is a subfield of machine learning that has demonstrated great promise in designing controllers for for a variety of tasks, including those in static environments such as playing video games. However, applying DRL to SRS presents significant challenges due to the highly dynamic environment where each robot only observes a partial view of the surroundings, and other robots are treated as part of the environment. This resulted in the development of numerous methods for optimizing SRS controllers utilizing DRL, such as curriculum learning, safety assurance mechanisms, and explainable reinforcement learning.

In this chapter, first, we give a review of SRS. Then, DRL and several related optimazation techniques are presented.

2.1 Swarm Robotic System (SRS)

2.1.1 Collective Bahivor of SRS

Robotic systems have been widely used in various applications, including industrial automation, transportation, and healthcare. With the advancement of artificial intelligence, SRS has emerged as a promising field for achieving collective behaviors in a group of robots. SRS consist of multiple robots that work together in a decentralized and self-organized manner. The collective behavior of the robots develops from simple interactions between individuals, requiring no central coordination. Collective behaviors that SRS can

perform are shown in Fig. 2.1 [6].

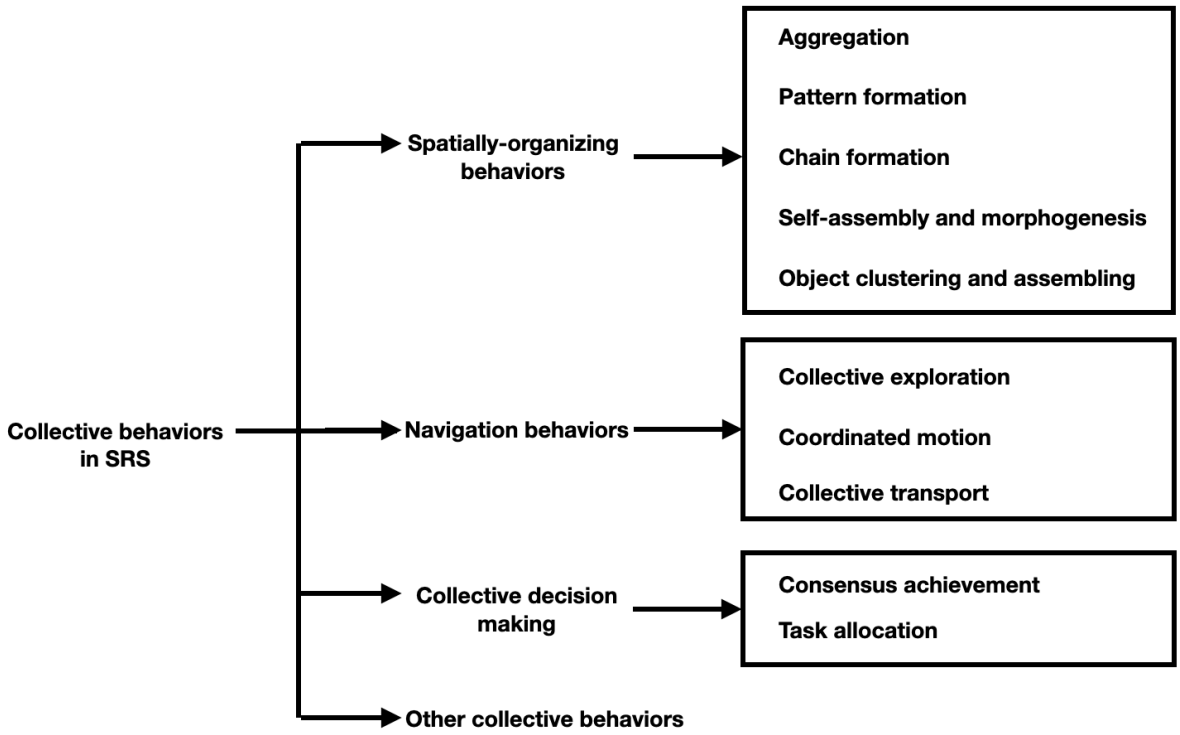


Figure 2.1: Collective behaviors in swarm robotics system

- **Spatially-organizing behaviors** involve the organization of robots in physical space to achieve a specific task or objective. Applications for spatially organizing behaviors include building, logistics, and environmental monitoring. For example, robots could be used to transport goods in a warehouse by coordinating their movements to avoid collisions and optimize their paths [8]. In Chapter 3, robots are required to complete a collective wall-jumping behavior, which belongs to this class. Overall, spatially-organizing behaviors are a key feature of SRS and provide a powerful tool for achieving complex tasks in physical space.
- **Navigation behaviors** enable robots to move and explore their environment in a coordinated and efficient manner. Navigation behavior is crucial in many applications of SRS. For example, robots could be used to search for survivors in a unknown zone [9], surveillance of dynamic areas [10], and transport of heavy objects collectively [11]. In Chapter 4, multi-autonomous vehicles will be formulized as a SRS. They need to move through an highway environment and reach a specific destination, which belongs to this class.

- **Collective decision making** is a key feature of SRS that allows a group of robots to make decisions collectively, such as consense achievement and task allocation.

2.1.2 Design Methods of SRS

As shown in Fig. 2.2, there are mainly two research directions of methods in the field of SRS, design and analysis methods [6].

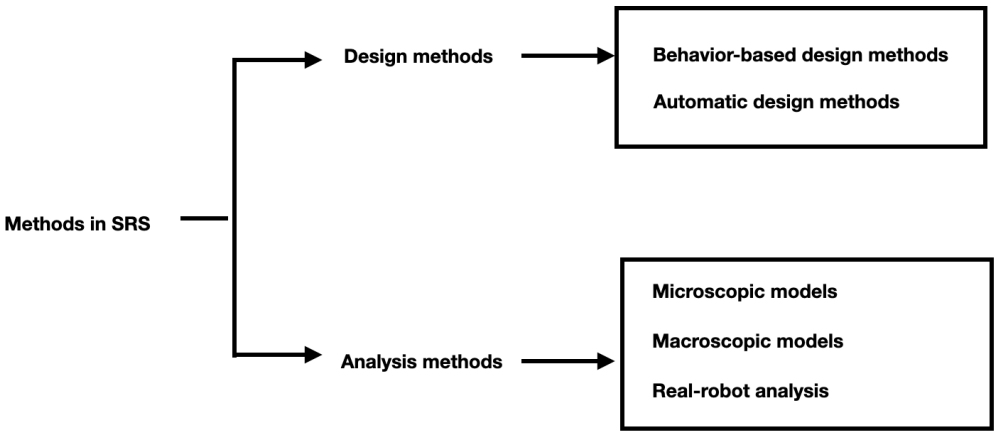


Figure 2.2: Methods in swarm robotics system

- **Design methods** of controllers for SRS could be classified into two categories, which are behavior-based design and automatic design methods [6]. A trial-and-error process is used in behavior-based design to develop, test, and refine each robot's individual behavior until the robots exhibit the desired collective behavior [12]. Behavior-based design methods have some advantages over other methods, such as simplicity, modularity, reusability, and adaptability. However, this method requires expert knowledge, and the system's performance is entirely dependent on the human designer.

  Another design method is automatic design [13], which means the controller is generated automatically by transforming the design problem into an optimization problem. Automatic design methods can be classified into two approaches: evolutionary algorithms [14] and reinforcement learning [15]. Evolutionary algorithms use a population of candidate solutions that are evaluated and modified according to a fitness function that measures their performance [16][17]. Reinforcement learning uses a trial and error process that rewards or penalizes the robots for their actions based on a reward function that measures their success [18]. Automatic design methods have some advantages over other methods, such as creativity, generality, and robustness.

In this thesis, we utilize reinforcement learning, one of the automatic design approaches, to train controllers for the SRS. Optimizing controllers by combining reinforcement learning with other algorithms to improve their performance.

- **Analysis methods** are determined by the specific task, the available resources, and the desired level of accuracy and complexity. Analyzing the behavior of a SRS is crucial to understanding its performance, identifying areas for improvement, and optimizing the system for a specific task.

  SRS can be modeled at two different scales: the microscopic level and the macroscopic level. In the microscopic level, also known as individual level, the characteristics of a single individual and their interactions with others is analyzed. In the macroscopic level, the characteristics of the entire system is analyzed [19]. Additionally, using actual robots to verify a collective behavior is a crucial tool. In [20], the authors propose an approach to optimize the odor localization behavior of SRS. They demonstrate that a set of real robots under fully distributed control are capable of cross an actual odor plume. Furthermore, these experiments can be reproduced and validated in the simulator.

2.1.3 Challenges of SRS Control

Due to the dynamic nature of the environment and the complexity of coordinating a large number of robots, the control of a SRS is a challenging problem. Some of the major challenges of SRS control include the following:

- Dynamic environments

  The dynamic nature of the environment makes it difficult to design a control algorithm that can adapt to changing conditions. For example, in the presence of obstacles or moving targets, the robots may need to change its behavior to achieve the desired objective. This challenge has been addressed in several studies, such as in the work by [21] where a SRS form different spatial structures in dynamic environments, to adapt to different task requirements.

- Control algorithm complexity

  The complexity of coordinating a large number of robots can be overwhelming, especially when dealing with nonlinear interactions between agents. The design of a control algorithm for SRS is a challenging task. This challenge has been tackled in several studies, such as in the work [6], where a decentralized control algorithm was proposed that can scale up to large SRS.

- Robustness and safety

  The safety of a SRS is a crucial aspect when designing a control algorithm. Due to the inherent uncertainty of the environment and the complexity of the SRS, there is a risk of unpredictable behavior that can cause harm to the robots. This challenge has been tackled in several studies, such as in the work by [22], where a SRS was able to follow a set of safety principles and verify their compliance.

There are other challenges such as communication issues or scalability. Overall, the challenges of SRS control are diverse and complex, and addressing them is critical to developing high qulity SRS.

2.2 Deep Learning (DL)

2.2.1 DL Techniques for SRS

Deep learning (DL) [23] has become a popular tool for developing control algorithms for SRS. By leveraging large datasets and powerful neural network architectures, DL algorithms can learn complex and sophisticated control policies that enable SRS to perform tasks with high efficiency and robustness.

One common approach is to use deep neural networks (DNNs) to model the dynamics of the robots and predict the future state of the SRS. This enables the development of model-based control algorithms to plan swarm's behavior in real-time. For example, [24] proposed a DNN-based control algorithm that can make a SRS cover an elliptical area and adapt to changing numbers of robots.

In addition to the use of DL for the design of control algorithms, it can also be used as a tool for processing inputs in SRS [25]. By using DNNs as a preprocessor for sensor data, the input data can be transformed into a more useful and informative format, which can enhance the SRS's perception capability and improve its overall performance.

Several studies have applied DNNs to the perception task of SRS, such as in the work by [26], where authors use graph neural networks (GNN) as a type of DNN to process neighbor information and achieve robots collaboration and self-organization.

Despite the potential of DL for SRS control, there are still many challenges need to be addressed, such as the difficulty of training DNNs with limited data and the challenge of developing interpretable control policies. Further research is needed to overcome these challenges and fully exploit the promise of DL for SRS control.

Overall, the use of DL techniques for SRS control represents a promising field to enable robots to perform complex tasks with high efficiency and robustness.

2.2.2 Deep Neural Network (DNN)

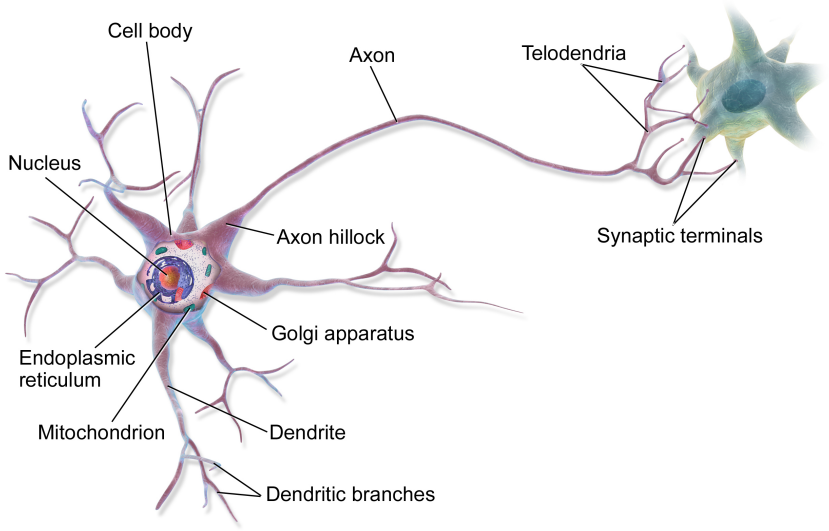


Figure 2.3: Anatomy of a multipolar neuron (licensed under CC BY-SA 3.0. https://en.wikipedia.org/wiki/Neuron)

Artificial neural networks (ANNs) are a type of machine learning algorithm that attempts to model the structure and function of the human brain. In a biological neural network, neurons communicate through synapses. Electrical impulses trigger neurotransmitter release, which bind to receptors on the receiving neuron's dendrites. If signals reach a threshold, the receiving neuron generates an action potential and transmits a signal downstream, as shown in Fig. 2.3.

There are at least three layers: an input layer, an output layer, and one or more hidden layers. The input layer receives information from outside world, such as images or text, then transmits it to the hidden layers for processing. The hidden layers perform a series of computations on the input data, gradually transforming it into a more abstract representation that captures relevant features or patterns. Finally, the output layer produces a response based on the transformed data, which could be a prediction, classification, or some other output.

The majority of modeling makes the assumption that each layer is completely connected. As shown in Fig. 2.4, all the units in one layer of a fully connected neural network are connected to all the units in the adjoining levels.

- **Activation function**

  Activation functions play a crucial role in DNN by introducing nonlinearity into the computations performed by the neurons [27]. Without activation functions, the

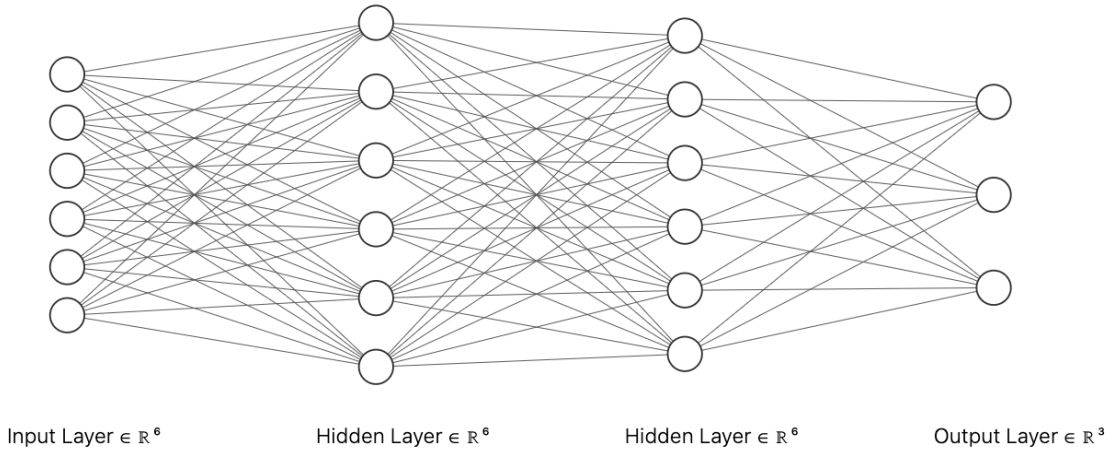


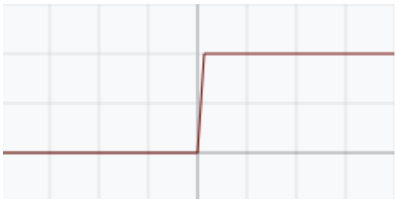
Figure 2.4: A fully connected neural network.

output of each neuron would be a linear function of its inputs, which would limit the complexity of the functions that the network could learn. Without activation functions, DNN would not be able to learn and model complex types of input such as photos, videos, audio, speech, etc.

There are many different types of activation functions used in DNNs, each with its own strengths and weaknesses. Several popular types of activation functions are shown in Fig. 2.5. The step function has a simple binary output of either 0 or 1, depending on whether the input is above or below a certain threshold. The sigmoid function is a classic activation function that maps the neuron's inputs to a value between 0 and 1. The tanh function is similar to the sigmoid function, but with a range of -1 to 1. The derivative of ReLU function is either 0 or 1 depending on whether the input is negative or positive, respectively. This allows for faster computation of gradients during backpropagation.

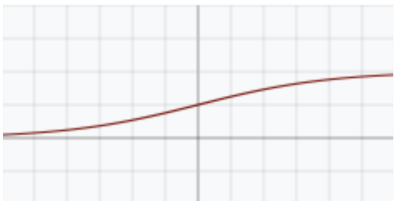
2.2.3 Convolutional Neural Network (CNN)

A Convolutional Neural Network (ConvNet/CNN) [28] is a type of DL algorithm. It is capable of analyzing input images and assigning significance, through learnable weights and biases, to different features or objects present within the image, ultimately enabling it to distinguish and classify objects with high accuracy. CNN makes the assumption that the input is a 3D tensor, which includes a 2D image with a depth of one or more than one (e.g., RGB images have a depth of three). The neurons in the CNN only have local connectivity to the previous layer. That is, each neuron always observes a sub-



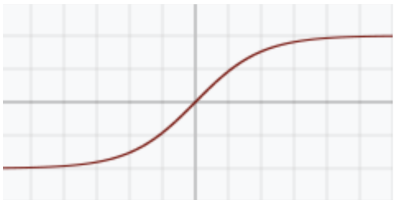
$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$

(a) Step function



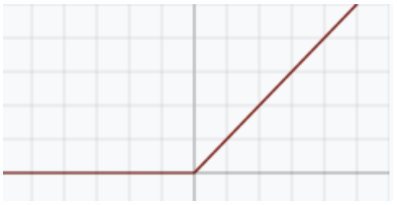
$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$

(b) Sigmoid function



$$f(x) = \tanh(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})}$$

(c) Tanh function



$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

(d) ReLU function

Figure 2.5: Common activation functions for artificial neurons. (licensed under CC BY-SA 4.0. https://en.wikipedia.org/wiki/Activation_function)

area of the 2D image but with full depth. The neurons in a convolutional layer are also different from the neurons of conventional structures, which have three hyper-parameters: receptive field, stride, and zero-padding. The receptive field determines the size of the area that the neuron can observe, and the stride determines the position to observe in the next moment.

Moreover, the zero-padding is used to extend the image to make sure that every element can be observed averagely by the neurons. Each neuron accepts a 3D tensor input and calculates a 2D output whose size is determined by the size of the input and the three hyper-parameters. Therefore the output of a convolutional layer is also a 3D tensor (with the depth of the number of neurons in the layer). Each neuron is also called

a filter or kernel.

2.3 Deep Reinforcement Learning (DRL)

2.3.1 Markov Decision Process (MDP)

A Markov chain is a stochastic process that has no memory and satisfies the Markov property, which means the probability of transitioning to a future state depends only on the current state and not on any previous state. It consists of a tuple (S, P), where S is a finite set of states and P is the transition probability distribution. However, when we add rewards to the model, we get a Markov Decision Process, which is a generalization of the Markov chain.

Markov Decision Processes (MDPs) [29] are a mathematical framework used to model decision-making problems. An MDP is a tuple (S, A, P, R, γ). The components can be summarized as follows:

- States S: is a set of possible states that the system can be in.
- Actions A: is a set of possible actions that can be taken in each state.
- Transition Probabilities P: is the probability of moving from one state to another state given an action.
- Rewards R: is a function that specifies the reward received for each state-action pair.
- Discount Factor γ: is a value between 0 and 1, determines the importance of future rewards.

Fig.2.6 [30] displays a diagram of a three-state MDP.

In a MDP, the reward in the S state is the reward expectation that can be obtained at the next moment t+1 at the state s at a certain time t, which is given by the equation below.

$$R_s = E[R_{t+1}|S_t = s] \tag{2.1}$$

G_t is the sum of the decays of all the rewards starting from time t on a MDP. There are also translations into "revenues" or "returns." The formula is as follows:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.2}$$

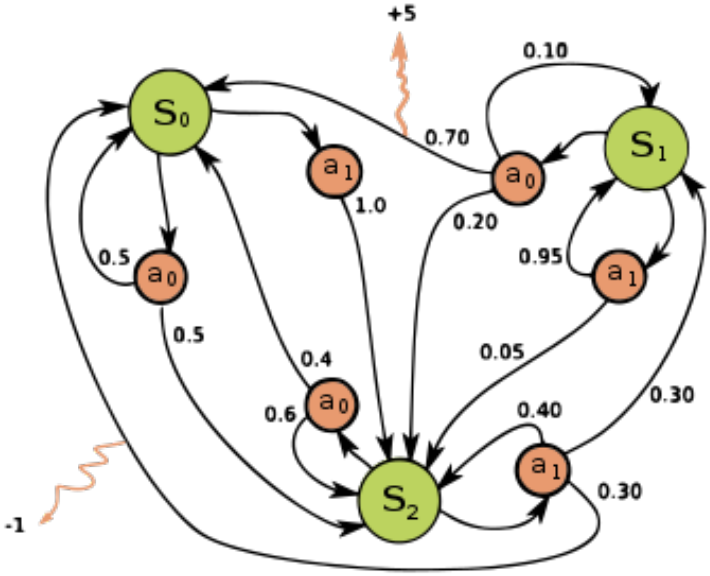


Figure 2.6: Diagram of a three-state MDP. (licensed under CC BY-SA 4.0. https://en.wikipedia.org/wiki/Markov_decision_process)

Discount factor γ is important to make the agent more focused on either short term reward or long term reward. If the value is 0, the agent will focus on the first reward that it receives, while if the value is 1, the agent will focus on all future rewards.

Giving a MDP and determining the best policy of action is the objectives of reinforcement learning (RL). The state-to-action mapping is referred to as the policy. The policy is commonly used by the symbol π. It refers to a distribution on the action set when a given state S is

$$\pi\left(a|s\right) = p\left[A_t = a|S_t = s\right] \tag{2.3}$$

In RL, a task is successfully accomplished if the agent manage to accumulate the as much rewards as possible over a given period of time. How successful a task is, is affected by how good a policy that being followed is. A certain policy π is better than other policy π' if the the expected return is greater or equal to that of π' for all states [31]. Knowing that expected return from a certain state is defined as state-value function $V, \pi \geq \pi'$ if and only if $V^{\pi}(s) \geq V^{\pi'}(s)$ for all states. The optimal state value function, denoted as

$V*$ is defined as

$$V*(s) = \max_{\pi} V^{\pi}(s) \tag{2.4}$$

This definition can also be enhanced and used for optimal action-value function $Q*$, defined as

$$Q*(s,a) = \max_{\pi} Q^{\pi}(s,a) \tag{2.5}$$

Because of this framework, RL is usually illustrated as two components, agent and environment as shown in Fig. 2.7. These are the process of reinforcement learning. Other way to explain this is that agent observes the environment. Agent also received reward from being in this certain state. Based on the state that is observed, the agent will select an action that maximize the cumulative reward.

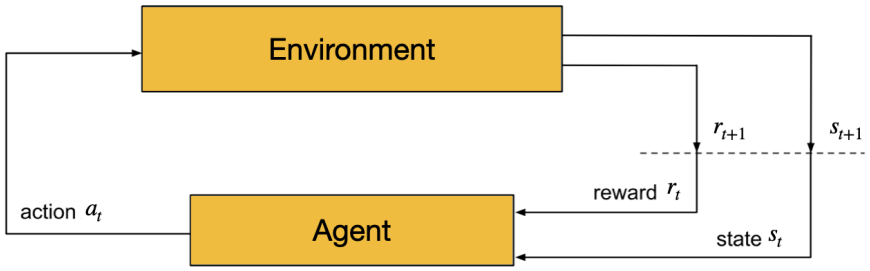


Figure 2.7: Relationship between agent and environment

Summarize the above, there are six components that make up an RL system: an agent, an environment, a policy, a reward function, a value function, and a model of the environment. Additionally, depending on the circumstance, the environment model is optional. Reward function is the most important. The expected rewards, as known as values, are the secondary. If there were no rewards,there could be no values. Because the sole point of calculating values is to obtain more rewards.

2.3.2 Exploring Model-free DRL for SRS

There are a variety of existing DRL algorithms, which can be categorized based on the following standards. The classifications and characteristics are shown in Fig.2.8.

According to whether the algorithm depends on the model, it can be divided into a model-based RL algorithm [32] and model-free RL algorithm [33]. The commonality of

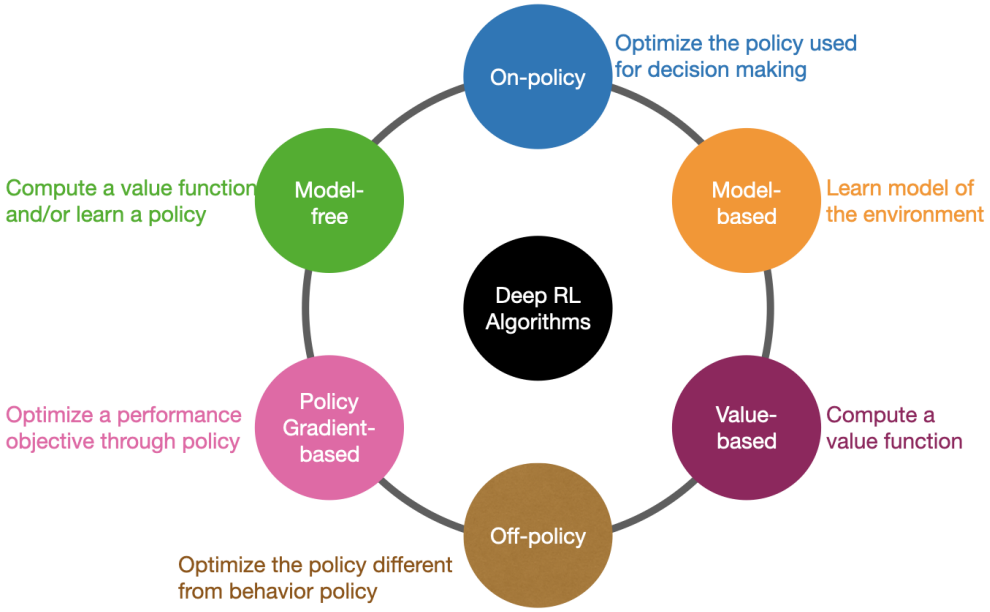


Figure 2.8: Characteristics of different DRL algorithms.

these two types of algorithms is to obtain data by interacting with the environment, the difference is that the way to use the data. The model-based RL algorithm utilizes a data learning system or an environmental model obtained by interacting with the environment and then performs the sequential decision based on the model. The model-free RL algorithm directly uses the data obtained by interacting with the environment to improve its behavior.

Since the environment for SRS is too dynamic to be modeled, the model-free RL algorithm is utilized in this thesis. Moreover, according to the strategy update and learning methods, the model-free RL algorithm can be divided into RL algorithm based on value function, RL algorithm based on direct strategy search, and actor-critic method. The so-called value-based RL method refers to the learning value function, and the final strategy is greedy according to the value function. In this case, in any state, the action with the most significant value function is the current optimal strategy. The RL algorithm based on direct strategy generally searches the strategy's parameters and learns the target's optimal parameters. The actor-critic approach [34] uses a combination of value functions and direct strategy searches. In [35], an actor-critic methodology is applied to make the swarm robotic system complete the task of locating the target.

Next, three popular model-free DRL algorithms are introduced: (i) value-based methods, (ii) policy gradient-based methods, and (iii) actor-critic methods.

2.3.3 Value-based Methods

Value-based methods frequently need to alternate between (i) estimation of the value function based on the present policy and (ii) policy optimization based on the estimated value function. However, it can be challenging to estimate a complex value function.

Q-Learning [36] is a tipical value-based optimization methods. Algorithm 1 summarizes the algorithm of Q-learning.

Algorithm 1 Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Parameters: step size $\alpha \in (0, 1]$, $\epsilon > 0$;
Initialize $Q(s, a)$, for every $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, except $Q(\text{terminal}, \cdot) = 0$;
for e= 1,...,E episodes **do**
    Initialize the starting state S
    for t=1,...,T timesteps **do**
        Select an action A with state S utilizing a policy obtained from Q
        Execute the action A, observe the reward R and the next state S'
        Update the action-value function utilizing the observed reward and the maximum expected value of the next state: $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
        Update the state: $S \leftarrow S'$
    end for
end for

The key aspect of Q-learning is to calculate the Q-value of the current state-action pair, which involves taking the maximum Q-value of the next state and factoring in the current reward. This allows the algorithm to learn from the best action to take in any given state, based on its previous experience.

A randomly perform actions instead of the optimal one a fraction of the time. The strategy is known as the $\epsilon -$ greedy approach. The ϵ denotes the randomness. Initially, the value of ϵ is high. High randomness in the actions chosen initially ensures that the agent explores its available options more often. Then as time progresses, the value of ϵ is decayed. This ensures that the agent explores and ultimately chooses the optimal set of actions and learns the optimal policy.

The formula of Q-Learning is defined as below

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_t) - Q(s_t, a_t)] \tag{2.6}$$

where α is the learning rate, which determines the extent to which the Q-value is updated based on the new information. The discount factor γ determines the relative weight of immediate and future rewards.

In some situations, reward to use is from N-time step. This derivation of Q-Learning is commonly called N-step Q-Learning, formula as below

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[G_t + \gamma Q(s_{t+1}, a_t) - Q(s_t, a_t)] \tag{2.7}$$

where G_t is defined as equation 2.2.

Q-learning can effectively solve some simple reinforcement learning problems where the actions are discrete. However, in practical applications, many actions are continuous, and the state-action spaces can be much larger and more complex. For example, in the game of Go, there are approximately 10^{170} possible states, and for each state-action pair, a Q(s, a) value needs to be recorded. The traditional tabular approach in Q-learning, which uses a lookup table to store all the Q-values, is not practical for such situations due to its high memory and computing resource requirements.

To improve scalability, more advanced techniques are needed. One such technique is function approximation, which uses a parameterized function to estimate the Q-values instead of explicitly storing them in a lookup table. This approach significantly reduces the memory requirements, and updating the function parameters can be done efficiently in a single step. However, function approximation introduces additional challenges, such as the possibility of overfitting and the need for careful selection and tuning of the function class and parameters. Furthermore, in real-world applications, state representations must be carefully designed and supplied as input to the Q-learning algorithm. This can be challenging, as the state representation should capture relevant features of the environment while also being computationally efficient to process.

Next, some optimization algorithms for Q-learning will be introduced.

- **Deep Q-network (DQN)**

  Deep Q-network (DQN), developed by Mnih et al. [37], is one of the most well-known value-based DRL algorithms. It is an advancement of Q-learning which calculates the Q-values utilizing a deep neural network. This approach can directly learn and extracting features from high-dimensional state spaces. For instance, image-based inputs used for playing ATARI games. DQN uses experience replay to randomly sample past experiences to break the correlations between consecutive state-action pairs and stabilize the learning process.

DQN employed a predictive DNN for executing Q-learning. Loss (L) is the amount that separates the approximated value from the actual value, as shown below.

$$L_t(\theta_t) = E_{(s,a,r,s')}\left[(target - Q(s,a,;\theta_t))^2\right] \tag{2.8}$$

where $target$ is a close approximation to the ideal action-value function Q, s, a, r, s' are experiences of randomly sampled agents that are stored in the replay memory, and θ are the parameters of the neural network:

$$target = r + \gamma \max_{a'} Q(s', a'; \theta_t) \tag{2.9}$$

- **Dueling DQN**

  The Dueling DQN architecture has shown promising results in addressing the overestimation of action values and slow convergence issues of the traditional DQN. By decomposing the Q-function into state value and advantage functions, the Dueling DQN is able to provide more accurate estimates of the action values while allowing for faster convergence [38].

  The architecture of the Dueling DQN consists of an input layer, a dueling layer, and an output layer. The dueling layer takes the feature vector as input and splits it into two streams, one representing the state value function and the other representing the advantage function. The state value stream estimates the value of being in the current state, while the advantage stream corresponding to each possible action. The output nodes of the advantage stream represent how advantageous each action is compared to the others in the current state.

  The final Q-value estimate for each action is obtained by combining the state value and advantage streams. This is done by adding the average of the state value stream to the difference between the advantage stream and its maximum value. By using this architecture, the Dueling DQN is able to learn more efficiently and accurately, leading to improved performance in various RL tasks.

- **Prioritized Experience Replay**

  Prioritize experience replay (PER) [39] change the sampling process in the learning process. Typically, at every update, the controller will sample the experience transitions uniformly. This means that every bit of experience has the chance to be sampled with the same probability. However, this technique tries to replay critical transitions more frequently, thereby learning faster and more effectively. The key idea is to make the experience with higher loss (L) or TD-error have a higher probability of being sampled. This is achieved by assigning a priority value to each

transition based on its TD-error or loss, and then using these priority values to sample transitions from the replay memory. The probability of sampling transition i is then given by the priority value of the transition divided by the sum of all priority values.

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{2.10}$$

where k is the minibatch, α is the prioritization and p_i is defined as follow.

$$p_i = \frac{1}{rank(i)} \tag{2.11}$$

where $rank(i)$ is the rank of transition i sorted according to the TD-error.

2.3.4 Policy Gradient-based Methods

The goal of policy gradient-based RL algorithms is to discover the best course of action that maximizes performance goals like predicted cumulative reward. The purpose is to improve the policy in order to maximize the expected reward, which is represented by the equation 2.13. In order to arrive at the best possible policy settings, this class of algorithms uses gradient theorems. The estimation of a value function based on the present policy is often necessary for policy gradient.

$$J(\theta) = E_\pi[r(\tau)] \tag{2.12}$$

where $r(\tau)$ is the reward for following the trajectory (τ). The performance objective is rewritten as follow [40].

$$J(\theta) = \int \pi(\tau) r(\tau) d\tau \tag{2.13}$$

To improve this value we need to find the gradient of the expectation, shown in equation 2.14

$$\nabla E_\pi = \int \pi(\tau) \nabla \log(\pi(\tau)) r(\tau) d\tau \tag{2.14}$$

We previously defined policy as probability of doing an action s for a given state s.

However, this is a simplification. The policy of a trajectory τ is defined in equation 2.15

$$\pi_\theta(\tau) = P(s_0)\prod_{t=1}^{T}\pi_\theta(a_t|s_t)p(s_{t+1}, r_{t+1}|s_t, a_t) \tag{2.15}$$

where P represents stationary ergodic distribution, π_θ is the policy to choose the action, and p is the dynamic of the environment. However, if we want to find the gradient of $\pi_\theta(\tau)$, we can remove the first and last part because their values are zero and we can get the gradient of J as follow.

$$\nabla E_{\pi_\theta} = E_{\pi_\theta}[r(\tau)(\sum_{t=1}^{T}\nabla \log \pi_\theta(a_t|s_t))] \tag{2.16}$$

This equation is the basis for all of any policy gradient based reinforcement learning. We will mention some of the algorithms in this category.

- **REINFORCE**

  This algorithm changes the term $r(\tau)$ with G_t shown in equation 2.17. In real world application, we cannot know the whole trajectory. This means that we will need to sample the trajectories. However, with enough iteration, the policy should converge.

$$\nabla E_{\pi_\theta} = E_{\pi_\theta}[(\sum_{t=1}^{T}G_t\nabla \log \pi_\theta(a_t|s_t))] \tag{2.17}$$

- **REINFORCE with baseline**

  Without baseline, the value of the cumulative future reward will vary depend on the sampling data. If the data has a high variance, the result of G_t will be biased.This algorithm introduce a bias b to improve the performance of the algorithm shown in equation 2.18.

$$\nabla E_{\pi_\theta} = E_{\pi_\theta}[(\sum_{t=1}^{T}(G_t - b)\nabla \log \pi_\theta(a_t|s_t))] \tag{2.18}$$

2.3.5 Actor-Critic Methods

In RL, policy gradient algorithms are often used to learn an optimal policy. However, in order to update the policy, we need to know whether the current policy is good. The value function can be used here, which estimates the expected return of following the current policy from a given state. By knowing the value of each state, we can estimate how good the current policy is and use this information to update the policy in the direction that leads to higher expected return [41][42].

The actor-critic methods consist of two components: a critic network and an actor network. The critic network estimates the state value function, which represents the expected total reward starting from a particular state. The state value function is updated using the time difference (TD) error, which is the difference between the estimated value of the current state and the estimated value of the next state, plus the reward obtained in the transition. The equation for the critic in equation 2.19

$$\nabla J(\omega) = r_{t+1} + \gamma V^{\omega}(s_{t+1}) - V^{\omega}(s_t) \tag{2.19}$$

where γ is the discount factor that determines the importance of future rewards. $V^{\omega}(s_t$ and $V^{\omega}(s_{t+1}$ are the estimated values of the current and next states, respectively. ω are the parameters of the critic network.

The actor network is implemented as a neural network that takes the current state as input and outputs a probability distribution over actions. The actor network is updated using the policy gradient, which is shown in last subsction. The equation for the actor in equation 2.20.

$$\nabla E_{\pi_\theta} = E_{\pi_\theta}[(\sum_{t=1}^{T} r_{t+1} + \gamma V^{\omega}(s_{t+1}) \nabla \log \pi_\theta(a_t|s_t))] \tag{2.20}$$

where the actor network updates the policy parameters θ for $\pi_\theta(a_t \mid s_t)$, in the direction suggested by the critic.

- **Proximal Policy Optimization (PPO)**

  The standard policy gradient-based methods have a significant drawback in that it can only be executed once for each set of data sampled by the policy π_θ. This is because updating the policy parameter θ to a new value θ_{new} requires resampling the training data from the perspective of the new policy $\pi_{\theta_{new}}$, which is computationally expensive. To address this issue, researchers are attempting to create policy gradient-based methods to update parameters θ_{new} using old data collected from previous policies $\pi_{\theta_{old}}$. An effective solution is Proximal Policy Optimization (PPO) methods, which offer a simple and effective way to update the policy parameters using the current and previous data [43]. To avoid excessive updates to old data, PPO

clips the importance sampling ratio. Specifically, the clipped importance sampling ratio is defined as:

$$r_t(\theta) = \frac{\pi_\theta \left(a_t \mid s_t\right)}{\pi_{\theta_{\text{old}}} \left(a_t \mid s_t\right)} \tag{2.21}$$

where $r_t(\theta)$ is the probability ratio between the two policies π_θ and $\pi_{\theta_{old}}$, which is a quantitative tool for comparing two policies.

One crucial component of PPO is the use of a clipped surrogate objective function that constrains the policy update to be within a certain range. This constraint prevents the policy from changing too much in each iteration, which can lead to instability. Furthermore, the clipping term serves as a form of regularization, which improves the stability and performance of the algorithm. The function is defined as follows:

$$L^{CLIP}(\theta) = E_{\pi_\theta}[min(r_t(\theta)A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t] \tag{2.22}$$

where A_t is the function estimator for the advantage function at time step t. ϵ is an arbitrary hyperparameter with a typical value of 0.1 or 0.2, which controls the maximum allowed change in the probability ratio between the new policy and the old policy.

2.4 Advanced Techniques for Optimizing DRL

2.4.1 Curriculum Learning for Reinforcement Learning

Curriculum Learning (CL) [44] is a technique which can used in RL to optimize the order in which an agent acquires experience to improve performance and training speed on a set of final tasks. By leveraging the generalization capabilities of the agent, CL allows knowledge acquired from simpler tasks to aid in solving more complex ones. By using this strategy, the amount of labeled data needed for training may be decreased, the model's robustness and generalizability can be increased, and the training process is able to speed up.

In traditional DRL settings, the agent begins with a random strategy and learn an optimal policy for the final task. However, when the final task is complex or the reward setting is sparse, it is tricky to train a model. CL addresses this issue by enabling the agent to train on one or several source tasks and transfer the knowledge to the final target task. The knowledge may be presented in the form of samples [45], options, policies, models [46], or value functions.

Some approaches assume that state and action spaces are the same in source and target task, or that they can be passed through a mapping [47]. These mappings may be manually specified or automatically learned [48]. In addition, there are also model-agnostic CL methods that can be applied to any RL algorithm without relying on any specific assumptions about the source and target tasks [49]. Model-agnostic methods typically involve pre-training a policy or value function on the source task and fine-tuning the learned parameters on the target task.

Algorithm 2 illustrates a generic formulation of CL.

Algorithm 2 Algorithm for General Curriculum Learning

1: **for** $t \in 1, 2, \ldots, n$ **do**
2: Compute the current performance of the model, $p \leftarrow P(M)$.
3: Update the curriculum by selecting examples or adjusting their weights according to the criterion/difficulty measure and the current curriculum level, $M, E, P \leftarrow C(l, M, E, P)$.
4: Select a subset of examples or mini-batches from the training dataset, $E^* \leftarrow \text{select}(E)$.
5: Train the model on the selected examples using the performance measure, $M \leftarrow \text{train}(M, E^*, P)$.
6: **end for**

where M is a machine learning model. E is a training dataset used to train the model. P is a performance measure used to evaluate the model's performance on the training dataset. n is the number of iterations/epochs in the training process. C is a criterion to determine the difficulty of the training examples. l is the current level of difficulty in the training process. And S is a scheduler used to adjust the difficulty level of the training examples.

In general, there are three essential components, where the agent can learn from numerous intermediate tasks until the final task is soled.

- **Task Generation**

  Task generation is a critical component of CL, as the quality of the training tasks can greatly impact the agent's performance and generalization ability. While manual task generation by domain experts can provide representative tasks and valuable domain knowledge, it can also be time-consuming and subject to human biases. Meanwhile, automatic task generation is a promising approach that can avoid the time cost and biases associated with manual design, while providing diversity and difficulty guarantees. By combining the strengths of both approaches, CL can benefit from both the domain expertise of manual task design and the efficiency and

diversity of automatic task generation. Narvekar et al. [50] proposed several methods to generate intermediate tasks for a specific final task using automatic task generation. Their methods involve defining the task domain and creating tasks by adjusting the task descriptors, enabling the generation of diverse and challenging training tasks.

- **Sequencing**

  Sequencing refers to the process of creating an order of experience samples or tasks to be used in CL. While traditionally curriculum have been manually designed by researchers, recent work has explored automated methods for curriculum sequencing. One such approach is Prioritized Experience Replay (PER) proposed by Schauul et al. [39]. Unlike methods that alter the domain or task, PER reorders samples within the same domain based on their expected learning progress, as measured by the TD error. In other words, samples with higher TD error are prioritized and replayed more frequently, allowing the agent to make stronger updates and implicitly learn from the samples. By prioritizing important transitions with high TD errors, PER enables the agent focuses on the most challenging parts of the task and acquire more complex skills.

- **Knowledge transfer**

  Knowledge transfer plays a vital role in CL that allows the agent to leverage knowledge gained from simpler tasks to solve more complex ones. Value function transfer is a technique that can be used to transfer low-level knowledge between tasks. The parameters of the value function obtained in the previous task is used to initializing the value function of subsequent subtasks. Liu et al. [51] proposed two methods: direct value function transfer and N-step returns-based value function transfer. The proposed methods have demonstrated the effectiveness in grid world environment, multi-agent particle environment, and Pac-Man game. Both learning effectiveness and asymptotic performance are significantly improved by the proposed methods.

2.4.2 Automatic Curriculum Learning for Reinforcement Learning

Automatic Curriculum Learning (ACL) is an approach in RL field that aims to improve the efficiency and speed of learning by dynamically adjusting the difficulty of the learning task [52]. It can automatically adapt the distribution of training data by learning to match a variety of learning situations for DRL agents.

ACL mechanisms can be employed for a variety of situations.

- **Enhancing performance on hard tasks.**

  In some cases, the target tasks in RL may be too difficult or have sparse rewards, making it challenging for the agent to learn directly. In this cases, ACL can be used to assign additional tasks to the agent, which will gradually lead its learning curve from simple tasks to challenging tasks until the target tasks are completed. Recently, ACL has been utilized to schedule DRL agents through difficult mazes, such as in work [53].

  Another way in which ACL can improve performance is by providing a more structured learning environment that encourages the agent to explore and learn in a more efficient manner. By breaking down a complex task into a series of smaller subtasks, ACL can help the agent to build up its knowledge and skills in a more incremental and structured way. This can lead to faster learning and better overall performance on the task set.

- **Improve generalization.**

  Agents should be able to complete tasks that they haven't encountered before in order to effectively enhance generalization. This is particularly important in scenarios where the task space is continuous or where there are distinct training and testing sets. In such cases, ACL can be used to shape the learning trajectories of the agent, improving its ability to generalize from simulated environments to the real world [54]. By gradually increasing the realism of the simulation and introducing more complex and realistic scenarios, agents can learn a strtegy that is more robust and transferable to the real world with the help of ACL. ACL can also be used to maximize performance in multi-agent settings via self-play [55], help the agents to learn more effective and robust strategies.

- **Training multi-goal agents.**

  Agents are trained and evaluated on tasks with multiple objectives in multi-goal RL. Recently, ACL has shown outstanding performance in multi-goal RL tasks. It can assist agents in more effectively learning the multiple objectives within a task by gradually adjusting the distribution of training data. For instance, in a multi-goal robotic arm manipulation task [56], ACL can help the agent to learn a diverse set of manipulation skills, such as grasping, placing, and lifting objects, that can be applied to a wide range of different goals.

While ACL mechanisms have shown promise in improving performance in complex learning scenarios such as RL, there are still some challenges and limitations that need to be addressed. One of the major issues is the absence of consistent benchmark for

evaluating the effectiveness of various ACL approaches. Another challenge is that most of the existing research on ACL has focused on supervised learning settings, where the input-output mappings are well-defined and easily measurable. It is not clear whether the same considerations and approaches apply to the more complex and dynamic settings of DRL, where agents must interact with a complex and uncertain environment to learn optimal policies. To address these challenges, there is a need for more research to develop standardized benchmark environments for evaluating ACL methods in DRL settings.

2.4.3 Explainable Reinforcement Learning (XRL)

Explainable Reinforcement Learning (XRL) is a promising area of research that has the ability to overcome many of RL's current challenges. One of the main challenges of RL is its lack of interpretability and explainability, which can hinder its adoption in applications where human trust and understanding are essential. This is because RL agents are typically trained using complex models, that are difficult to understand and analyze. Furthermore, RL agents often learn policies that are suboptimal or unexpected, which can make it difficult to trust or deploy these models in real-world applications.

As a result, it can be quite beneficial to observe and fully understand the decision-making process in order to identify and address issues with the trained model. Table 2.1 shows the classification of interpretability methods.

Table 2.1: Interpretability methods classification.

Method Classification	Time of Obtaining Explanation	Scope
Visual and Language-Assisted Explanation	Post-hoc	Local
Behavior Cloning	Post-hoc	Global
Interpretable Models	Intrinsic	Global
Logical Relationship Extraction	Post-hoc	Local
Policy Decomposition	Intrinsic	Local

In the topic of interpretability, classification typically exists on two factors: the time it takes to receive the explanation and the scope of the explanation. Interpretability approaches are divided into intrinsic and post-hoc explanations especially based on the time that the explanation was obtained. Intrinsic explanations limit the model's expression to generate interpretable outputs at runtime. For example, constructing a model based on strong interpretability principles and components (such as decision trees and linear models), or adding specific processes to generate interpretable outputs. Post-hoc explanations achieve the goal of explanation by analyzing the model's behavior and summarizing its behavior patterns. In general, intrinsic explanations are explanations during the strategy

generation process, specific to a particular model, while post-hoc explanations are explanations after the strategy generation, independent of the model. According to the scope of the explanation, interpretability methods are classified into global and local explanations. Global explanations provide macro-level explanations of the model by ignoring the model's microscopic structures (such as parameters and layers), while local explanations start from the microscopic level and obtain explanations of the model by analyzing its microscopic structures. In this thesis, the proposed apporach belongs to the category of local explanations.

There is an increasing amount of literature on local explanations algorithms. Grün et al. [57] outlined three different types of methods: saliency methods, input reconstruction, and perturbation-based methods.

- **Saliency Methods**

  A group of approaches known as saliency methods are used to draw attention to areas of an input image that are crucial to the model's decision-making. The aim of these methods is to provide insights into how the model processes visual information and makes decisions.

  Gradient magnitude heatmaps [58] and class activation mapping [59] were followed by more advanced techniques like guided backpropagation [60], excitation backpropagation [61], GradCAM [62], and GradCAM++ [63]. Visualizing distinct regions that are in support of the current prediction and those that are against it was done by Zintgraf et al. [64]. Sundararajan et al. differentiate between sensitivity and implementation invariance in their article [65]. It's important to note that these techniques seem to provide accurate saliency maps even for networks with unpredictable weights, according to [66]. As is demonstrated by Kindermans et al., [67], saliency methods do not result in analytically rational justifications for linear models.

- **Perturbation Methods**

  Perturbation methods are a class of techniques used in the field of machine learning to understand how changes in input data affect the output of a model. These methods are often used to provide insight into the decision-making process of neural networks, by measuring the sensitivity of the model to small perturbations in the input data.

  By placing an occluding rectangle across the image and tracking how the prediction changes, Zeiler and Fergus [68] created a heatmap of importance for each area that was occluded. While Dabkowski and Gal [69] develop a neural network for masking salient regions, Fong and Vedaldi [70] examine this approach by introducing noise or blurring to the image and repeatedly finding a minimal perturbation mask that

lowers the classifier's performance. These methods can be used to identify potential holes or weaknesses in these models. They are also important for developing robust and reliable machine learning systems that can perform well in real-world scenarios.

- **Input Reconstruction**

  Image reconstruction is a technique used in the field of computer vision to generate new images that are similar to existing ones. In the context of deep learning, image reconstruction can be used to visualize the features learned by a neural network, by synthesizing an image that maximally activates certain neurons in the network.

  Input reconstruction approaches are the most closely linked research since our method synthesizes inputs for the agent. Based on nearest neighbors in feature space, Long et al. [71] employed the average of image patches to rebuild an image. Dosovitskiy and Brox develop a CNN to reconstruct the input from its encoding [72], meanwhile Mahendran and Vedaldi [73] suggest to rebuild images by inverting representations learned by CNNs.

  The applications of image reconstruction techniques are numerous, including art generation, content creation, and feature visualization. They can be used to better understand the behavior of DL models and to generate new content that is similar to existing data. These techniques are also important for developing more advanced and sophisticated machine learning models that can perform complex tasks in a variety of domains.

In order to make RL more interpretable and explainable, several well-known algorithms have been proposed. Zahavy et al. implement t-SNE on the final layer of a DQN to cluster the agent's behavioral states [74]. As Greydanus et al. [75] investigate how the current state influences the strategy in a vision-based method employing saliency methods, Mnih et al. [37] utilize t-SNE embeddings for visualization. To illustrate the value and advantage function of a dueling Q-network, Wang et al. [38] apply saliency approaches from the work of Simonyan et al. [58].

XRL has a wide range of applications in domains where human trust and understanding are essential. For example, in healthcare, XRL can be used to develop decision support systems that provide physicians with transparent and interpretable recommendations for treatment. In finance, XRL can be used to develop trading systems that are transparent and understandable to human traders. In robotics, XRL can be used to develop robots that are transparent and understandable to humans, which can help to build trust and acceptance of these technologies.

Chapter 3

Generating Collective Wall-Jumping Behavior for a Swarm Robotic System with Self-Teaching Automatic Curriculum Learning

3.1 Introduction

Swarm robotics (SR) [2][3] studies how systems composed of a large number of robots can be used to accomplish collective tasks that are beyond the capability of a single robot. There is no centralized control, and robots can only have partial information about the environment. It takes inspiration mainly from social animals such as bees, ants, and birds. In particular, a robotic swarm exhibits the following advantages; Fault-tolerance, Flexibility, and Scalability. Fault tolerance means the system can still achieve the task while some individuals cannot work. Flexibility means the robotic swarm can cope with similar environments and tasks. Scalability is the ability to achieve the given task with different group sizes.

The design methods in SRS can be mainly divided into two approaches, i.e., behavior-based design method and automatic design method [6]. A trial-and-error process is used in behavior-based design to develop, test, and refine each robot's individual behavior until the robots exhibit the desired collective behavior [12] Behavior-based design methods have some advantages over other methods, such as simplicity, modularity, reusability, and adaptability. However, this method requires expert knowledge, and the system's performance is entirely dependent on the human designer. In automatic design, the design problem is transformed into an optimization problem [13], and then optimization algorithms are adopted to develop the controller. Two representative automatic design

methods are: Evolutionary Robotics (ER) and Reinforcement Learning (RL) [15].

The majority of the current literature on automatic design belongs to evolutionary robotics, in which the control policies are optimized by artificial evolution. ER has been used to accomplish a variety of tasks, such as aggregation, collective transport, and path formation. However, ER approaches require a large number of computation resources when the dimensionality of parameters is enormous. An alternative is RL. Recently, RL has made a remarkable achievement in addressing a variety of robotic problems. The design of the reward function plays a critical role in the success of RL algorithms. The reward function defines the goal of the RL agent and provides feedback on its behavior, guiding it towards the desired behavior. In particular, the sparse reward problem is a common challenge in RL where the agent receives little or no feedback for its actions, resulting in training failure.

In the case of sparse rewards, the agent may have difficulty learning the optimal policy since it receives little feedback on its actions, resulting in a slow or failed learning process. To address this problem, various techniques have been proposed, such as Curriculum Learning.

A promising approach to solve this problem is Curriculum Learning (CL) [44]. CL is a training approach that imitates the meaningful learning sequence in human curricula. The central idea is to train the agent in numerous tasks in a meaningful sequence, until the agent can solve the target task. However, the curriculum is frequently designed manually by researchers. Manual CL limits the pace of training and heavily relies on the designer's experience. Recently, many researchers have been investigating how to automatically generate such curricula for RL [52]. Sequencing subtasks is the most common way. More fundamentally, a curriculum can also be defined as a series of experience samples.

In this chapter, we propose a novel method called Self-Teaching Automatic Curriculum Learning (STACL). The central idea is to automatically schedule the lessons that the agents should learn based on the learning progress of previous lessons. This approach is intended to be more effective than manual CL or random CL approaches. In order to illustrate the effectiveness of STACL, the chapter uses a collective wall-jumping task where robots must jump over a high wall and reach the goal as quickly as possible. The experiments are conducted in computer simulations, and the results are compared to those of manual CL, random CL, and a conventional RL algorithm. Results show that the STACL method has the fastest convergence speed and the most stable performance. All CL methods can train agents to generate collective wall-jumping behavior, while the conventional RL approach fails. In addition, the flexibility of the developed controllers is also examined.

3.2 Related Work

A jumping robot can pass through obstacles several times its height and has an excellent ability to avoid danger. There are some studies about single jumping robot [76]. For instance, Noh et al. proposed flea-inspired jumping robot is capable of jumping up to 30 times farther than it is height [77]. Another research direction is to use multiple modular robots to overcome the obstacle together. M-block is a kind of cubic modular jumping robot. It is a novel self-assembling, self-reconfiguring robot that uses pivoting motions to change its intended geometry [78]. M-blocks achieve the jumping motion by quickly applying a brake to an internal flywheel to transfer angular momentum to the body of the robot. A single module is incompetent but together they can achieve tasks beyond an individual's ability.

In the Atari game of Pong and Breakout, the reward function is relatively simple, and the agent can receive a positive reward for every successful action. Therefore, RL has achieved better performance than human players. However, in more complex games like Montezuma's Revenge [79], the reward function is often sparse and only provides feedback in certain rare situations, making it difficult for the agent to learn effective strategies. In Montezuma's Revenge, the player must perform a series of actions to complete a subtask and obtain a reward. This requires the agent to explore the environment and discover the sequence of actions that lead to a positive reward. However, if the agent receives little or no feedback for its actions, it may have difficulty learning the optimal policy, resulting in a slow or failed learning process.

Similarly, in the wall-jumping task, robots have to conduct a series of behaviors. They have to aggregate, form a staircase-like structure, jump over the wall and finally reach the goal. Therefore, it is hard for the controller trained by RL to complete this task when the wall is relatively high. In that case,CL can be used to direct the learning trajectory from easy to challenging tasks until the goal tasks are completed. The performance of CL depends on how we design the curriculum for specific applications and datasets. Therefore, research into automatically designing the data subset or curriculum, also known as Automatic Curriculum Learning (ACL) is growing rapidly. ACL is an approach in RL field that aims to improve the efficiency and speed of learning by dynamically adjusting the difficulty of the learning task [52]. By learning to match a range of learning scenarios for DRL agents, it can change the distribution of training data automatically. Recently, ACL has been utilized to schedule DRL agents through difficult mazes, such as in work [53].

3.3 Research Methodology

In this section, we describe how RL can be applied with CL. The approach we proposed will be described in detail.

3.3.1 Curriculum Learning with Reinforcement Learning

CL [44] is a technique which can used in Reinforcement Learning (RL) to optimize the order in which an agent acquires experience to improve performance and training speed on a set of final tasks. It is like how humans learn new things, usually from easy to difficult. By leveraging the generalization capabilities of the agent, CL allows knowledge acquired from simple tasks to aid in solving more complex ones. By using this strategy, the amount of labeled data needed for training may be decreased, the model's robustness and generalizability can be increased, and the training process is able to speed up.

In RL algorithms, the knowledge transfer from the source tasks to the target task can be realized by the transfer of the value function. The value function represents the expected return from a given state. It is equal to the expected total reward R_t for an agent starting from state s. For MDPs, $V^{\pi}(s)$ can be defined as:

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\{R_t \mid s_t = s\} \\ &= E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\} \end{aligned} \tag{3.1}$$

In general, a value function depends on the policy π. Therefore, the policy learned from the source task π_{source} can be used in the target task. The value function in the target task $V_target(s)$ is initialized using the parameters of the value function $V_source(s)$ learned in the source task. An initialization bias can be introduced to allow the agent to investigate the goal task more effectively.

3.3.2 Self-Teaching Automatic Curriculum Learning

A proper lesson sequence is an essential part of CL. The majority of previous research has employed manually constructed curricula, in which a researcher chooses the sequencing of samples or courses. However, research into automated methods for curriculum sequencing has been emerged recently. In this study, we proposed a novel sequencing algorithm named Self-Teaching Automatic Curriculum Learning.

In each episode, agents choose lessons to practice for the following episode. Agents train on those lessons and return the episode total reward. Changes in the episode total reward represent the slope of the learning curve, i.e., the training progress of agents.

Initially, lessons are sampled randomly. From the second episode, lessons with a higher learning curve will have a higher probability of being sampled. Then other lessons will be selected more, and training will repeat until all lessons have been learned. The central concept is that agents should practice the lessons with the highest learning curve slope, where they will improve the most. Algorithm 1 shows the pseudocode of the proposed method.

Algorithm 3 Self-Teaching Automatic Curriculum Learning

for e = 1,...,E episodes **do**
  if e == 1 **then**
    Select lesson k randomly
  else
    Select lesson k using Boltzmann policy based on L: $P(a) = \frac{\exp(L[a]/\tau)}{\sum_{i=1}^{I} \exp(L[i]/\tau)}$
  end if
  Initialize a list L to store the number of lesson selections
  Initialize environment with lesson k
  for t = 1,...,T=5000 timesteps **do**
    if The task is finished **then**
      Break
    end if
    for j = 1,...,N=24 agents **do**
      Agent j selects lesson n
      L[n] = L[n] + 1
      Add reward to agent j based on reward settings R_{STACL}
    end for
  end for
  for j = 1,...,N=24 agents **do**
    Add $r_s = R_i^k - R_{i-1}^k$ to agent j.
  end for
end for

A list L is initialized to store the number of times each lesson was selected by agents, the length of the list is the number of lessons. List L will be reset at the beginning of an episode. At each timestep, each agent selects a lesson. The corresponding number in the list $L[n]$ will be increased by one, where n is the index of the lesson. At the beginning of an episode, the Boltzmann policy is used to select the lesson based on the list L. The Boltzmann policy calculates the probability that each lesson will be selected. Then, the lesson for the next episode is decided by sampling the probability distribution. If the lesson k is selected, each agent will receive a reward $r_s = R_i^k - R_{i-1}^k$ at the last

time step of an episode, which is the change in total reward between the ith training and $i-1$th training. The total episode reward will increase more when r_s is high. Therefore, according to the principle of maximizing the reward of the RL algorithm, the probability of agents selecting lesson k will be increased. In the Boltzmann policy, the probability of each action is calculated by a softmax over the value of each action. Then the action is sampled from this distribution. The equation is shown in Eq. 3.2.

$$P(a) = \frac{\exp\left(L[a]/\tau\right)}{\sum_{i=1}^{I} \exp\left(L[i]/\tau\right)} \tag{3.2}$$

Where $P(a)$ is the probability of selecting lesson a, I is the number of lessons, τ is a temperature parameter, and the number of times the lesson a is selected in one episode is represented by $L[a]$. A higher τ value makes the policy more random, whereas a lower τ value makes the policy greedier. During training in this research, τ is set to 1.

3.4 Collective Wall-jumping Task

3.4.1 Environment Settings

In this study, computer simulations are conducted to perform a collective wall-jumping task, where the robots have to reach the goal area as soon as possible. The environment is conducted in three-dimensional space using Unity3D game engine. The top view of the environment is shown in Fig. 3.1. The environment is a rectangle area surrounded by walls. It is 80 meters long and 60 meters wide. The goal area (green-colored area) is 48 meters long and 12 meters wide. In the beginning, 24 robots are placed on the field. A 5-meter-high wall is located in the central area, which is an obstacle for robots to overcome.

3.4.2 Robot Settings

Fig. 3.2 shows the design of the robots. As shown in Fig. 3.2(b), for robots to observe the environment, seven IR sensors are attached to detect the distance between the robot and other objects within 20 meters. With IR sensors, a robot can recognize walls, the goal, and other robots. Robots have four kinds of actions which are forward movement, side movement, rotating, and jumping. The robot is 1 meter high, and its maximal jumping height is 1.2 meters. Robot's jumping ability is a little higher than its height so that it can jump on the top of another. In order to maintain the stability of the staircase-like structure, robots perform the overhead check. The overhead check means that a robot detects whether another robot exists above itself. If another robot exists, the robot will immediately be motionless to prevent the upper robot from falling.

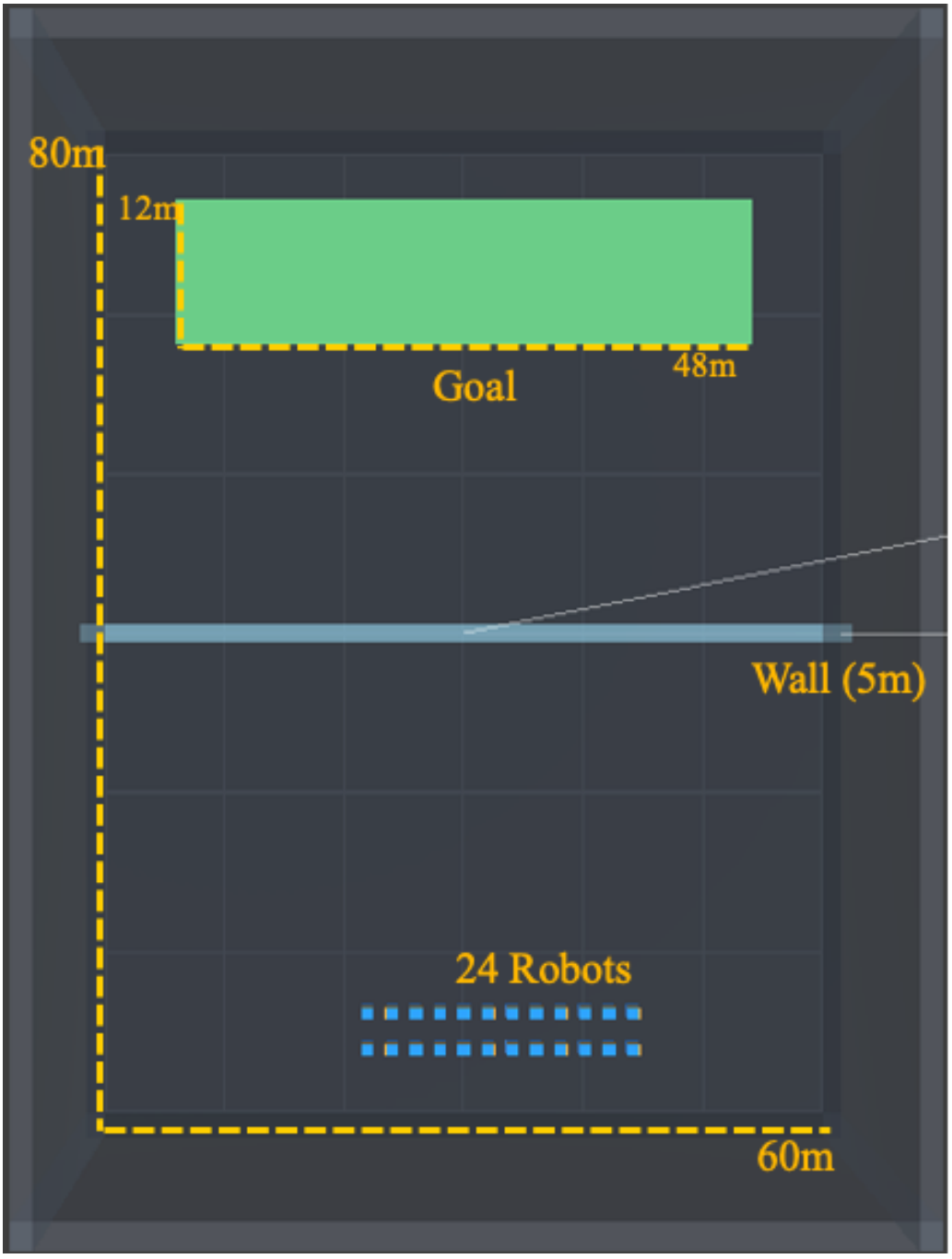
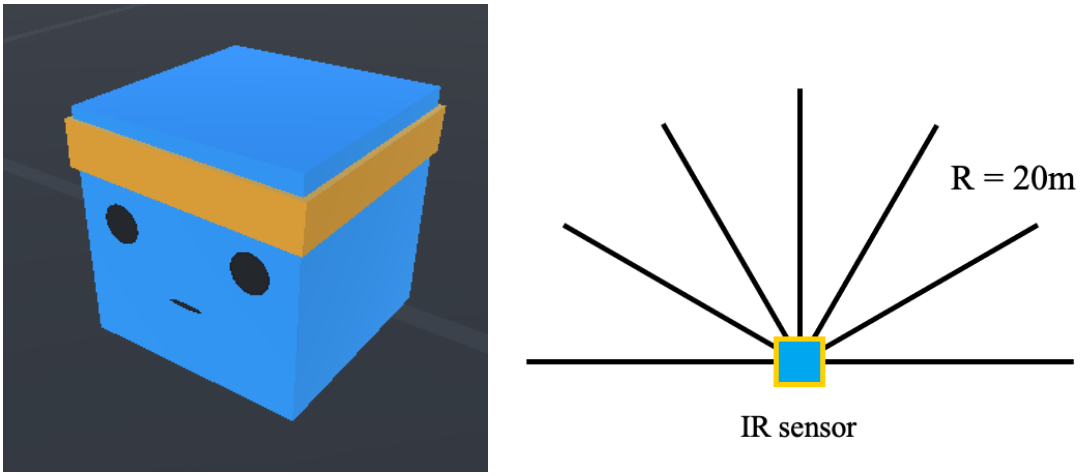


Figure 3.1: Top view of the environment.

3.4.3 Task Settings

There are 10 million timesteps in one training process. Each episode consists of 5000 timesteps. As shown in Fig. 3.3, robots have to learn to aggregate, form a staircase-like structure, jump over the wall and finally reach the goal. When 12 out of the 24 robots reach the goal, the task should be complete, and the episode will end. Note that the maximum jumping height of a robot is 1.2 meters, while the target wall height is 5



(a) Robot overview.

(b) Arrangement of IR sensors.

Figure 3.2: The robot design.

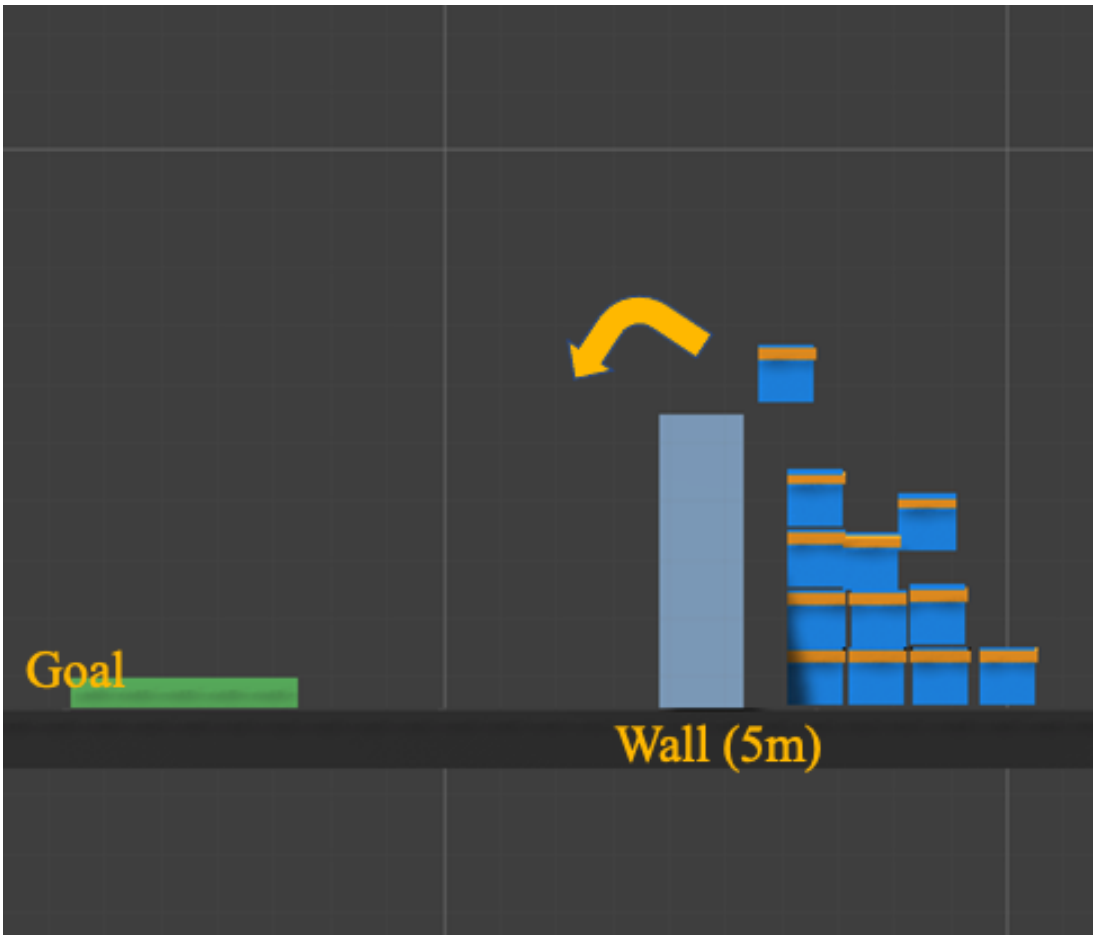


Figure 3.3: The desired structure to complete wall jumping task.

meters. Thus one robot can never complete the task. Robots need to form a four layers structure to pass the wall collectively.

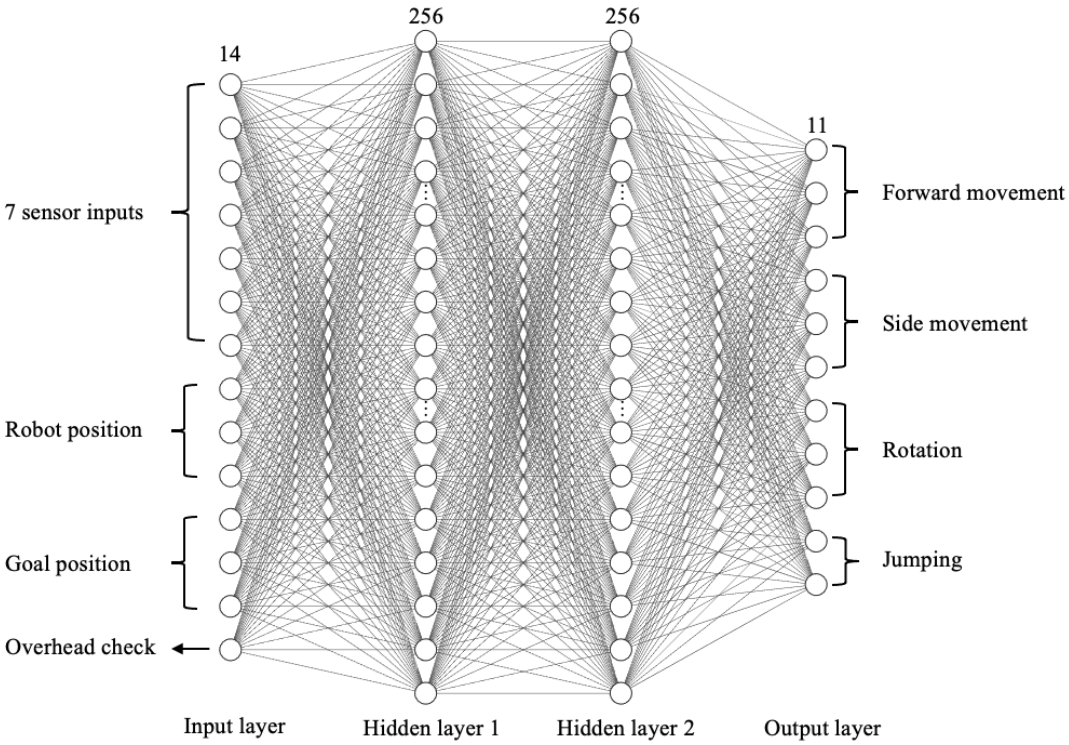


Figure 3.4: Architecture of the vanilla neural network.

3.4.4 Network Structure and Reward Settings

Fig. 3.4 shows the vanilla network architecture. This network trains controllers for manual CL, random CL, and conventional RL. It contains two hidden layers; each layer has 256 nodes. The first layer receives robot position, goal position, overhead checks, and 7 sensor information as inputs. The outputs of the neural network are four robot actions: forward movement, side movement, rotating, and jumping. Robots obtain a sparse reward based on the following equations:

$$
\begin{aligned}
R &= \sum_{j=1}^{N}\sum_{t=1}^{T} r_{g,j,t} + \sum_{j=1}^{N}\sum_{t=1}^{T} r_p \\
r_{g,j,t} &= \begin{cases} 1 & \text{if robot } j \text{ reaches the goal at time } t \\ 0 & \text{otherwise} \end{cases} \\
r_p &= -\frac{1}{5000} \quad \text{at each timestep}
\end{aligned}
\tag{3.3}
$$

where episode reward R is the sum of the rewards received by all agents in one episode, N is the number of robots, and T is the length of the episode. The environment gives robots both positive and negative rewards based on their states and actions. The robot j will receive the reward $r_{g,j,t}$ =1 if it reaches the goal at timestep t. The tiny negative

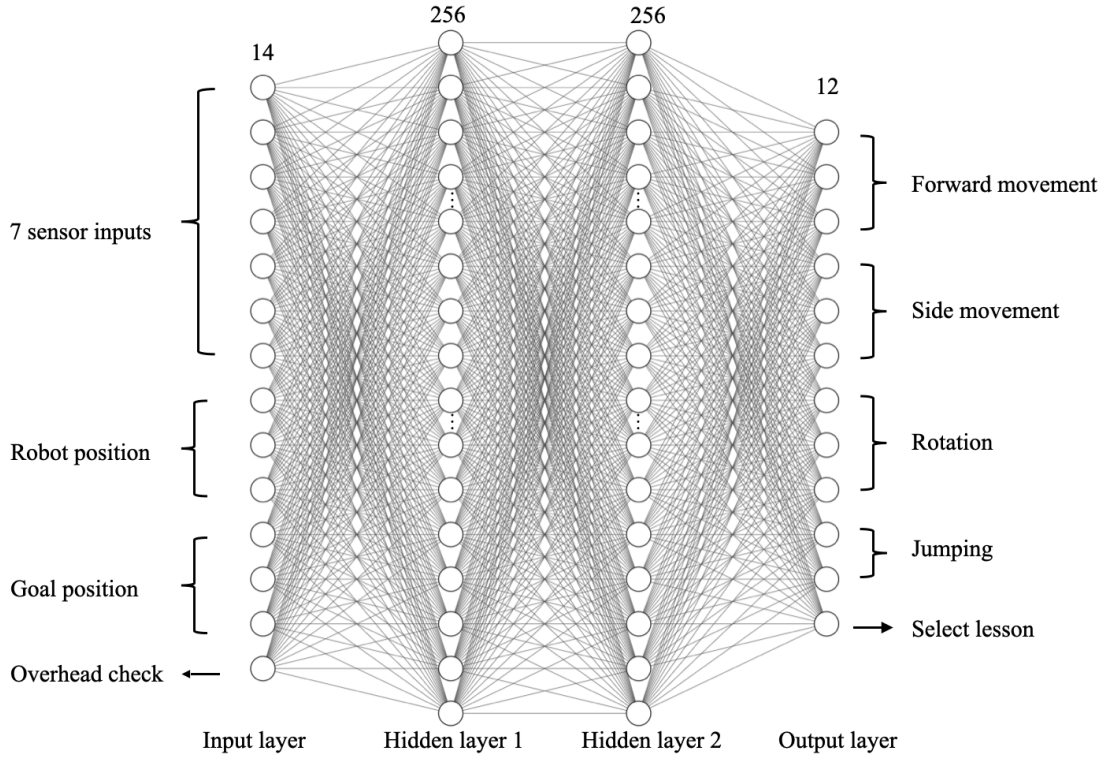


Figure 3.5: Architecture for self-teaching automatic curriculum learning.

reward r_p in each timestep is the time penalty that is used to encourage the robot to finish the task as soon as possible.

The network architecture for STACL is shown in Fig. 3.5, which can train controllers to not only complete the wall-jumping task but also to schedule lessons automatically. Inputs and hidden layers are identical to the vanilla network architecture. In addition to the four robot actions, a lesson selection output has been added. Each agent selects a lesson and records it at each timestep. When an episode ends, a lesson for the following episode is chosen using the Boltzmann policy.

For STACL, the reward function is based on the following equations:

$$\begin{aligned} R_{STACL} &= R + \sum_{j=1}^{N} r_s \\ r_s &= {R^k}_i - {R^k}_{i-1} \quad \text{at end of the episode} \end{aligned} \tag{3.4}$$

Where R_{STACL} is the sum of the original episode reward R and the lesson selection reward r_s. If the lesson k is selected, the robot j will receive a lesson selection reward r_s at the last time step of an episode. The number of training in the lesson k is represented by i. Changes in $R_i^k - R_{i-1}^k$ represent the progress between the ith training and $i-1$th training. As a result, the lesson selection reward r_s is used to motivate robot j to select

the lesson that maximizes the episode reward change.

Table 3.1: Hyper-parameters

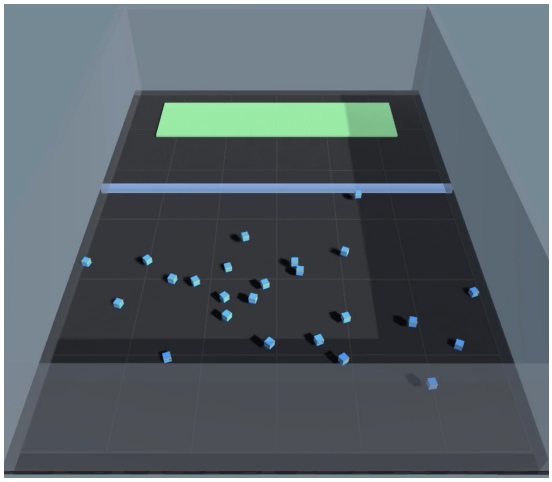
Hyper-parameter	Value
Batch size	256
Buffer size	20480
Optimizer	Adam
Learning rate	0.0003
Number of timesteps	1e7
ϵ in PPO	0.2

Some important hyper-parameter values are shown in Table 3.1.

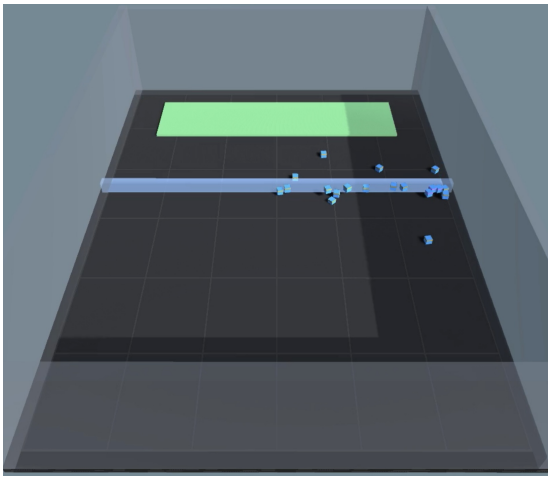
3.5 Results

Simulation snapshots of the controller trained with the proposed method are shown in Fig. 3.6. There are four lessons. The wall in lesson 1 is 1 meter high. In Fig. 3.6(a), robots have a sparse spatial distribution because a single individual can jump over the wall. It is relatively simple for robots to find the goal and get rewards when they explore the environment. Robots learn the right direction to reach the goal in lesson 1. The wall height in lesson 2 is 2 meters. A robot can only jump over the wall only from the top of another robot; they need to cooperate as a group. As can be observed in Fig. 3.6(b), robots learn to aggregate with other robots. In lesson 3, The wall height is 3 meters. A bigger group is needed. From Fig. 3.6(c), we can find the robots form a staircase-like structure and generate the collective jumping behavior. The wall height in lesson 4 is 5 meters, which is also the final target task. From Fig. 3.6(d), robots form the most aggregate pattern and biggest staircase-like structure.

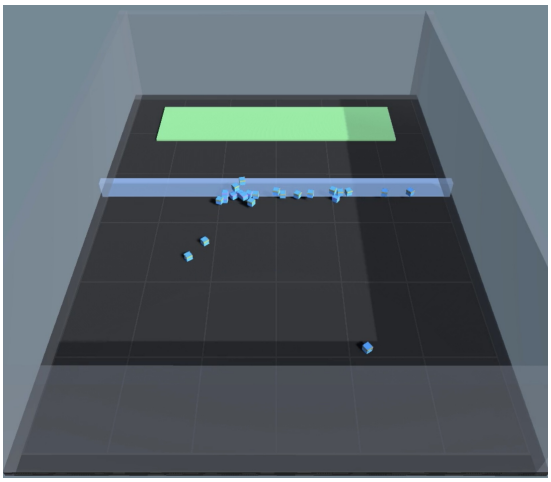
Fig. 3.7 shows the episode reward for the comparison experiment, which is an average of 10 trials. Manual CL training takes 1 million timesteps for lesson 1, 2 million timesteps for lessons 2 and 3, and 5 million timesteps for lesson 4. In manual CL, cumulative rewards are reduced when lessons and wall heights change. It increases again as the training progresses. However, since the difficulty gap between lessons 2 and 3 is small, the cumulative reward around the 3M timesteps changes slightly. The cumulative reward converges to around 0.36. In STACL, since the training progress is not limited by manual settings, the convergence speed is faster. From 5M timesteps, the controller maintains a



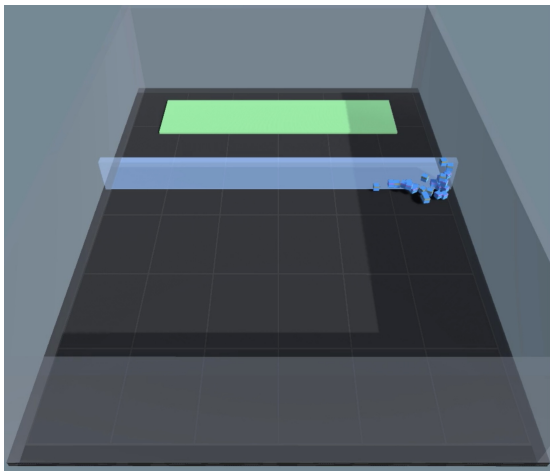
(a) Lesson 1



(b) Lesson 2



(c) Lesson 3



(d) Lesson 4

Figure 3.6: Simulation snapshots of the controller trained with STACL in the training process.

high-level performance. It is important to note that the lesson selection reward r_s is not included in the trajectory in order to compare with other approaches. The cumulative reward converges to around 0.43. Random CL means that lessons are randomly selected for each episode. In random CL, training progress is slow and unstable. Then it eventually converges to roughly the same level as the manual CL. In a conventional PPO, cumulative rewards are always -1; controllers fail to learn to complete the task without CL.

The episode length to accomplish the task is shown in Fig. 3.8, which is the average of 10 trials. The episode length refers to the time it takes to complete the task, which is the time it takes 12 out of the 24 robots to reach the goal. When the episode length is less than 5000 timesteps, the task has been completed within the time limitation. Furthermore, the

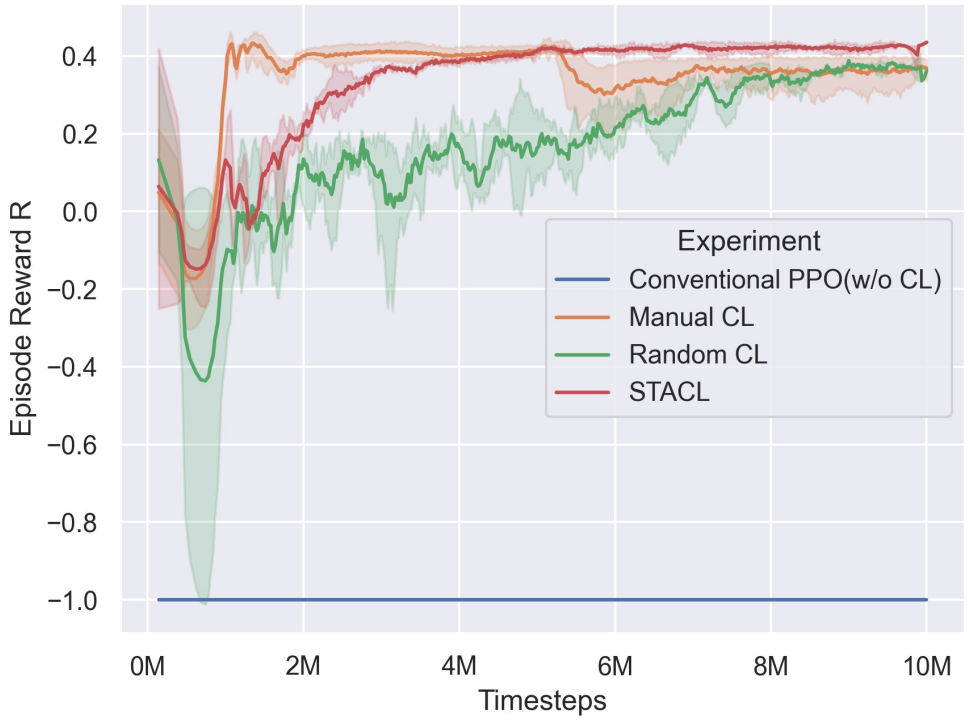


Figure 3.7: Trajectories of episode reward, where each data point is the average of 10 trials (with standard deviation).

shorter the episode length means the controller can complete the task more efficiently. In manual CL, the trained controller is able to complete the task around 700 timesteps. Also, the episode length increases at 1M, 3M, and 5M timesteps because of the same reason mentioned above. In STACL, the controller converges at approximately 5M timesteps with a slight standard deviation and completes the task for about 350 timesteps. As a result, the controller trained with STACL can accomplish the task more efficiently than the manual CL. The training process is slower and less stable with random CL because it lacks an appropriate method for choosing lessons. Conventional RL cannot complete the task throughout the entire training process; episodes end only when the time runs out.

Fig. 3.9 shows an example of lesson distribution during the training process. The experiment consists of 11635 episodes, of which 50 episodes are sampled once to examine the trend in lesson selection. As can be seen, lesson 1 is selected the least frequently, and the majority of them are distributed at the beginning of the experiment. Lessons 2 and 3 are selected more frequently as the difficulty increases. Additionally, they are sometimes chosen in the latter stage of training. This is because agents need retraining on tasks that the

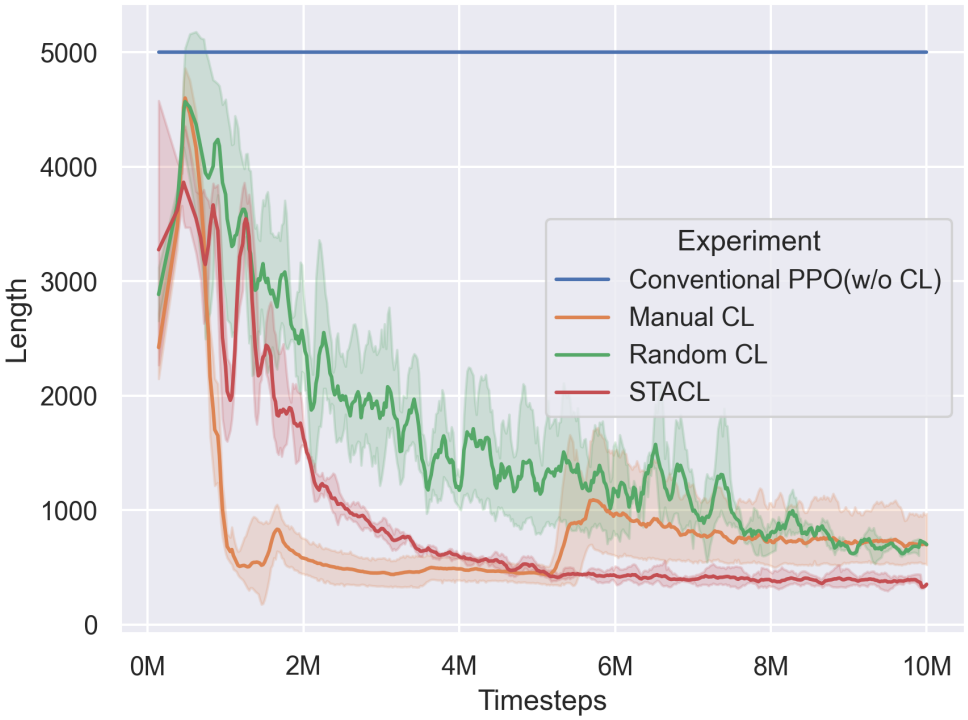


Figure 3.8: Comparison of the episode length required to accomplish the task, where each data point is the average of 10 trials (with standard deviation).

network is starting to forget to counter unlearning. Lesson 4 was selected the most, with a selection rate of over 70%. Consequently, for the collective wall-jumping task, spending a large amount of time training the hardest lesson and intermittently training the easier intermediate lessons to counter unlearning is an appropriate lesson sequence. It should be mentioned that the lesson sequence for each trial is slightly different. For instance, the fluctuations in the episode length across multiple trials will alter the number of episodes, which implies a change in the length of the lesson order. However, the proposed model can ensure that the lesson with the greatest progress is chosen under different training states, enabling robots to complete the task effectively and with high performance.

Additionally, to examine the flexibility of the swarm, evaluation experiments are conducted. Flexibility is the property that a robotic swarm should still be able to cope with changes in environments and tasks. Therefore, we saved the controllers with the highest reward trained in previous experiments by the STACL method, the manual CL method, and the random CL method, respectively, to address new tasks. First, the wall heights were changed to 3m, 4m, 5m, 6m, and 7m, respectively. The controller is evaluated 20 times in each new environment. The results of the test experiments are shown in Fig.

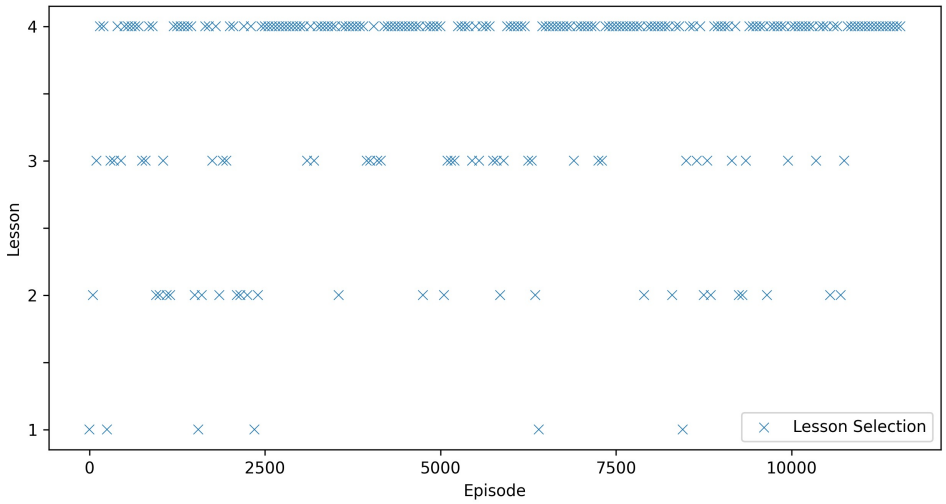


Figure 3.9: An example of lesson distribution during the training process.

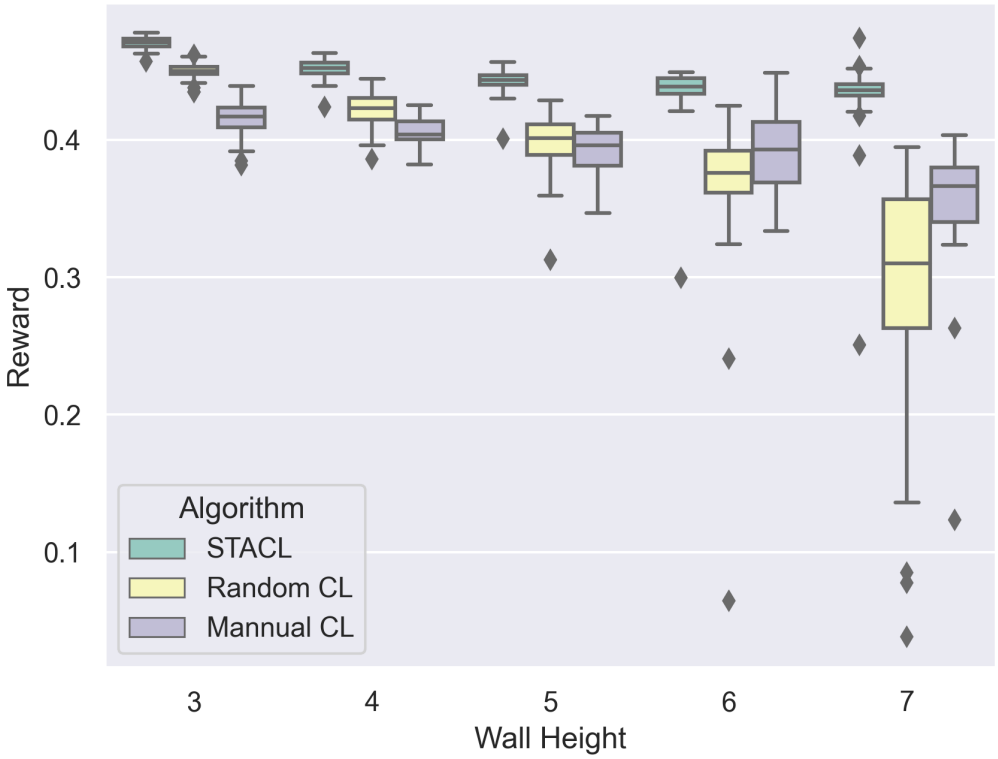


Figure 3.10: Flexibility testing with different wall heights.

3.10. As it can be observed, the proposed controller can achieve the highest reward with

the smallest variance in every new environment. For manual CL and random CL, as the wall height increases, the cumulative reward decreases, and the variance of the cumulative reward also increases. Especially in the environment where the wall height is 7m, there is a huge performance drop for random CL. In general, it can be said that the controller developed with STACL exhibits great flexibility.

3.6 Conclusion

In this study, we proposed STACL, in which agents automatically select which lessons to train next from a given set. In the proposed approach, agents practice more of the subtasks with the highest slope of the learning curve so that they can make the most progress. In order to illustrate the effectiveness of STACL, this study uses a collective wall-jumping task. The manual CL algorithm, random CL algorithm, and traditional RL algorithm are compared with the proposed method. Simulation resluts demonstrate that the suggested method has the quickest convergence speed since it can automatically schedule lessons and is unaffected by manual settings. All CL methods can complete the given task. In contrast, the conventional RL method failed because of the sparse reward problem. Additionally, we also performed experiments to examine the flexibility of the proposed approach. One limitation of our approach is that the intermediate lessons are generated manually, which may not produce optimal experiment results. We believe that careful fine-tuning could result to better performance.

Chapter 4

Autonomous Highway Driving Using Reinforcement Learning with Safety Check System based on Time-to-Collision

4.1 Introduction

In recent years, the development of autonomous vehicle (AV) technologies are greatly promoted by advances in the field of artificial intelligence (AI) and machine learning (ML). However, there are still many issues in high interactive traffic scenarios such as ramp merging[80] unprotected left turns [81] [82], and narrow street passing [83] [84]. Autonomous vehicles need to interact with other traffic participants, react to road objects, and select an appropriate strategy.

Most autonomous vehicles have a modular hierarchical structure and can be divided into four components [85], which are perception, prediction, decision-making, and control. Decision-making is an essential component and received significant attention from academic and industry organizations. The majority of current approaches for decision-making methods can be divided into the rule-based method and the data-driven method. The rule-based methods employ heuristics and hard-coded rules to guide the behaviors, such as the Intelligent Driver Model (IDM) [86] and the MOBIL model [87]. For instance, if an autonomous vehicle with a rule-based model observes a stop sign while driving, rules enforce the model to set the acceleration to negative until the vehicle stop. It is feasible to design a strategy hand-crafted for simple traffic scenarios. However, the number of rules increases exponentially in complex scenarios, and there may be conflicts between the rules. Furthermore, the strategies are designed case-to-case, which lacks robustness

and generalization ability to new scenarios.

An alternative is data-driven method such as Reinforcement Learning (RL). Recently, RL has made a remarkable achievement in addressing a variety of robotic problems and autonomous driving tasks. The decision-making problem for autonomous navigation can be formalized as a Markov Decision Process (MDP) [29]. An agent (autonomous vehicle) attempts to adopt the optimal policy to maximize rewards while taking into account the influences of its behaviors through dynamic interaction with the environment. Most of the previous works are limited to single-agent tasks and cannot be directly introduced to multi-agent tasks. Multi-agent RL algorithms need to maximize the sum of the rewards of all agents, which makes it more difficult to optimize the network. Furthermore, as the number of agents increases, the complexity of the environment rises as well, which leads to a dramatic increase in the variance of optimization methods that estimate gradients by sampling.

In order to apply a reinforcement learning algorithm to an autonomous driving problem, a feature representation of the state must first be chosen. The state should at least contain a description of nearby vehicles and the environment. The spatial-temporal state feature is the most commonly-used representation [88]. A vehicle driving on the road can be described in a kinematic way by its continuous position, velocity, and heading. This representation is efficient. However, it has two limitations. First, the environment might change across time and space, which presents a challenge for learning strategies that depend on inputs with a fixed size. Second, this type of feature representation is permutation-variant, which means that it is impacted by the order in which the interacting agents are listed. For example, a different feature representation might emerge from simply changing the feature entries of agents i and j.

In this chapter, we utilize Time-to-Collision (TTC) as the feature representation to train a controller for multiple autonomous vehicles. TTC is the time needed for a vehicle to collide if it continues driving on the same route and at the same speed. TTC focuses on the potential risk posed by other vehicles and static obstacles rather than a specific agent. Therefore, even if the number or order of the surrounding vehicles changes, the feature representation will not be affected.

We also propose a safety check system (SCS) based on TTC as shown in Fig.4.1. First, the SCS needs to determine whether a vehicle is unsafe. TTC is used as a threat assessment in several approaches [89] [90]. However, the general definition of TTC is calculated from relative distance and relative velocity with constant relative acceleration. When two vehicles are moving with approximately the same velocity, even if the distance between them is very close, the general TTC-based SCS will not detect a potential collision. Therefore, in the proposed method, the TTC under the three driving circumstances of uniform speed, acceleration, and deceleration will be calculated to improve the safety

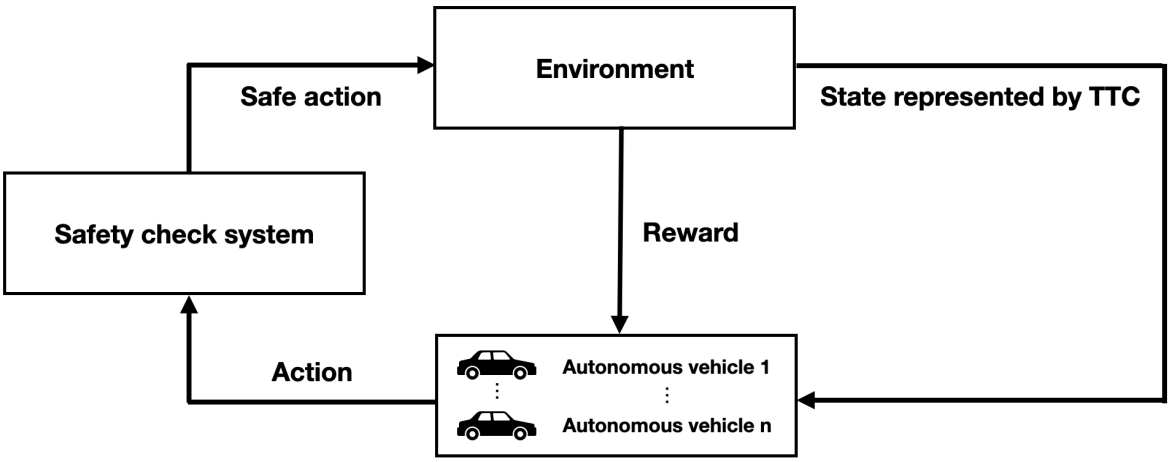


Figure 4.1: The framework of the proposed approach.

of the system. Second, the SCS needs to replace dangerous action output by RL with a safer action. For instance, the safety system proposed in [91] [92] includes a dynamically learned safety module in addition to handcrafted safety rules. However, the majority of current safety enhancement methods were created for single autonomous vehicle tasks. For tasks involving multiple autonomous vehicles, the purpose of SCS is to guarantee the safety of the entire system. In this case, relying on communication between vehicles, the proposed method can satisfy comprehensive requirements including safety and order rationality.

The main contributions of this study are summarized as follows.

1. We use the modified TTC as the state representation to train an RL controller for multiple AVs. It performs better than the conventional approach utilizing kinematics, demonstrating the reliability of this state representation.

2. We propose a safety check system for multiple AVs to enhance the safety of the system and improve the learning efficiency of RL. The results of the simulation experiments demonstrate that, even in cases of heavy traffic, the proposed approach can successfully increase the arrival rate and decrease the collision rate. Furthermore, evaluation experiments are conducted to examine the performance of the safety check system with different time thresholds.

3. A ramp merging task in the computer simulation is used to examine the effects of the proposed method. Autonomous vehicles and environmental vehicles co-exist in the merge lane and the main lane. Vehicles on the ramp need to merge into the main lane efficiently without collision. After passing the main lane, autonomous vehicles need to divert to the ramp and reach the goal.

4.2 Research Methodology

In this study, Proximal Policy Optimization (PPO), one of the most well-known DRL algorithms, is used to train a decision-making controller. However, DRL algorithms are not safe enough since the agent is encouraged to explore a wide range of states to find the best strategy. Therefore, it is necessary to equip autonomous vehicles with a security assurance mechanism when collisions are about to occur. In this study, we propose a safety check system for reducing collisions by utilizing Time-To-Collision (TTC).

4.2.1 Time-To-Collision (TTC)

In research on traffic conflicts techniques, Time-To-Collision (TTC) has proven to be an effective measure for rating the severity of traffic conflicts [93]. According to Hayward's definition, TTC [94] is the amount of time needed for two vehicles to collide if they continue driving in the same route and at the same speed. It stands for the danger posed by the vehicle at the current lane and speed. There is a higher chance of a collision when the TTC is low. The TTC of two vehicles can be approximated by Equation 4.1.

$$TTC = \frac{|R_i|}{|V_i| \cdot \cos < R_i, V_i >} \tag{4.1}$$

Where R_i is the relative position vector of vehicle i, V_i is the relative velocity vector of vehicle i, $|\cdot|$ is the 2-norm of a vector.

4.2.2 PPO with Safety Check

In this study, we propose a safety check system based on TTC. The central concept is that the action output by the DRL controller should be replaced with a safer action chosen by the safety check system when an autonomous vehicle detects a potential collision, i.e., the TTC is below the threshold. The overall algorithm is shown in Algorithm 4.

The detection of collision is achieved by calculating the TTC under the three driving circumstances of uniform speed, acceleration, and deceleration. Autonomous vehicles will communicate with each other, when a potential collision is detected, a safer action using Algorithm 5 will replace the action output by the RL controller. First, all the AVs will calculate their TTC according to the action output by the RL controller. Then, each AV broadcasts its TTC to other AVs and sequences them based on the level of risk. The action will be replaced if the TTC is below the time threshold. The high-risk vehicle will be given top priority for action replacement. The TTC of every possible action will be recalculated and the action with the highest TTC will be chosen as the new action, indicating maximum safety. When the new action is selected, the high-priority AV will

Algorithm 4 PPO with safety check

```
Inistialize replay buffer D.
for m = 0 to M episodes do
    for t = 0 to T timesteps do
        for v = 0 to V vehicles do
            Observe s_t^v.
            Select an action a_t^v using the neural network.
        end for
        Safety Check
        for v = 0 to V vehicles do
            Execute new action a_t^v' in environment.
            Get reward r_i^v.
            Add to replay buffer D with (s_t, a_t^v', r_i^v)
        end for
    end for
    Update the networks using PPO.
end for
```

broadcast its latest target lane and speed to others. The process will then be repeated by vehicles with lower priority until the entire system is in a safe situation.

4.3 Experiment Settings

This study conducted experiments in highway-env simulators [95]. The code is implemented in Python using Pytorch framework. A computer with NVIDIA RTX 3070 GPU, AMD Ryzen 9 3950x CPU, and 128GB memory is utilized for the experiments.

4.3.1 Task Settings

In this study, a ramp merging task is used to evaluate our method. The environment is shown in Fig. 4.2. The simple task consists of 4 autonomous vehicles and 6 environmental vehicles. There are 300,000 timesteps in one training process. The hard task consists of 8 autonomous vehicles and 8 environmental vehicles. There are 500,000 timesteps in one training process. In order to increase the complexity of the hard task, an obstacle is added to the main lane. In the beginning, the vehicles in each of the three lanes are generated at a random position. Autonomous vehicles and environmental vehicles co-exist in the merge lane and the main lane. Vehicles on the ramp need to merge into the main lane efficiently without collision. After passing the main lane, autonomous vehicles need to divert to the

Algorithm 5 Safety check

for each autonomous vehicle v **do**

Calculate the TTC if excuate action a_t^v .

Broadcast its TTC to all neighboring autonomous vehicles.

v.decided = False.

end for

Within each autonomous vehicle, sequence the vehicles by TTC from small to large.

for each autonomous vehicle v with lowest TTC **do**

v.decided = True.

Calculate TTCs with different actions using the target lane and speed of other vehicles whose decided is True, and ignoring the autonomous vehicles whose decided is False.

if TTC of the current speed < Threshold **then**

Select the action with highest TTC.

end if

Calculate the target lane and speed with the new action.

Send the target lane and speed to all other autonomous vehicles.

end for

ramp and reach the goal. For environmental vehicles, we utilize the Intelligent Driver Model (IDM) [86] and MOBIL model [87] for longitudinal acceleration and lateral lane change decisions.

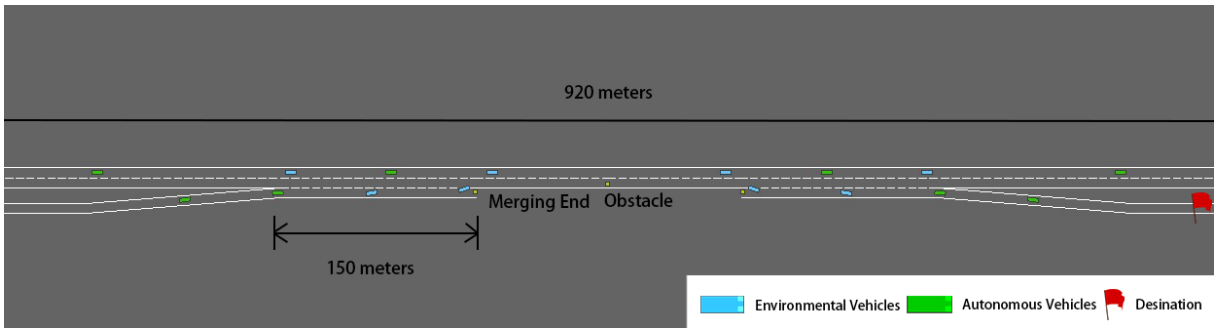


Figure 4.2: The ramp merging task conducted using the highway-env simulator. In the left lanes, the vehicles are generated at random. Environmental vehicles are depicted in blue, whereas autonomous vehicles are depicted in green. The on-ramp's vehicles need to merge into the main lane. After a section of main lane, autonomous vehicles need to exit the highway by the off-ramp.

4.3.2 Neural Network Settings

The state space of the baseline method is the features of other vehicles, including *ispresent*, x, y, v_x, v_y, where *ispresent* is a variable denoting whether a vehicle is observable, x and y are the longitudinal and lateral position of the observed vehicle, v_x and v_y are the longitudinal and lateral speed of the observed vehicle. The state is represented by a 16 *times* 5 matrix, and the maximum of 16 cars (including 8 autonomous and 8 environmental vehicles) can be observed.

The proposed approach uses TTC as the state representation. At each timestep, each vehicle will calculate TTCs with different speeds. In this study, the state of the autonomous vehicle is represented by a $3 \times 3 \times 10$ matrix. The first dimension represents the TTC if the vehicle at the current speed, slows down, or speeds up. The second dimension represents the left, current, and right lane of the vehicle. The third dimension represents the TTC time in one-hot encoding.

The set of high-level control decisions, including turn left, turn right, accelerate, decelerate, and idle are referred to as the autonomous vehicle's action space. Following the high-level decision, the low-level controller will generate the corresponding steering and throttle control signals to control the vehicle.

There are two neural networks in PPO, i.e., the actor and the critic. Fully connected layers are used in both neural networks. The outputs of the actor are the probabilities of the actions that the vehicle may execute. The critic's output is the value function. The actor and critic networks each have two fully connected layers with 256 hidden units.

The architecture of the actor and critic networks are shown in Fig. 4.3 and Fig. 4.4.

Table 4.1 lists the hyper-parameters utilized in this investigation. Ten times each experiment was carried out using different random seeds.

4.3.3 Reward Settings

The performance of DRL algorithms is highly dependent on the design of the reward functions. At each timestep, each vehicle gets a reward using the Equation 4.2.

$$r_t = r_c + r_a + r_s + r_l \tag{4.2}$$

Where r_t is the total reward that the vehicle can receive at each timestep, r_c is the collision penalty when the vehicle is involved in a collision, r_a is the arrived reward when the vehicle reaches the goal, r_s is the reward when the vehicle speeds up. The penalty for changing lanes is r_l, which is intended to discourage lane switching by the vehicle. The values of each reward setting are shown in Table 4.2.

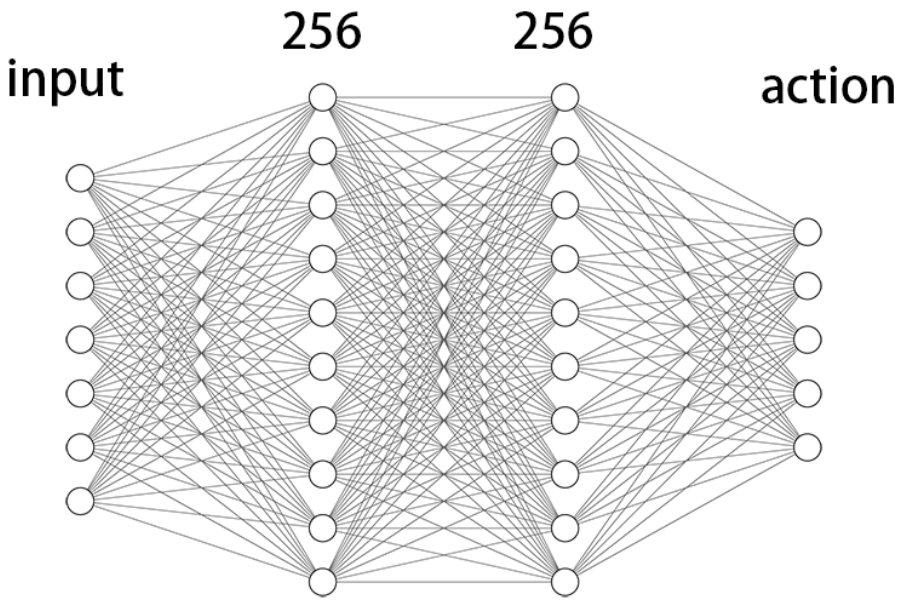


Figure 4.3: The network structure of the actor.

Table 4.1: Hyper-parameters

Hyper-parameter	Value
Batch size	64
Buffer size	240
Parallel environments	32
Optimizer	Adam
Learning rate	0.0005
Number of timesteps	3e5
ϵ in PPO	0.2
TTC time threshold	3
Input size of kinematic representation	16×5
Input size of TTC representation	$3 \times 3 \times 10$
Output size of actor network	5

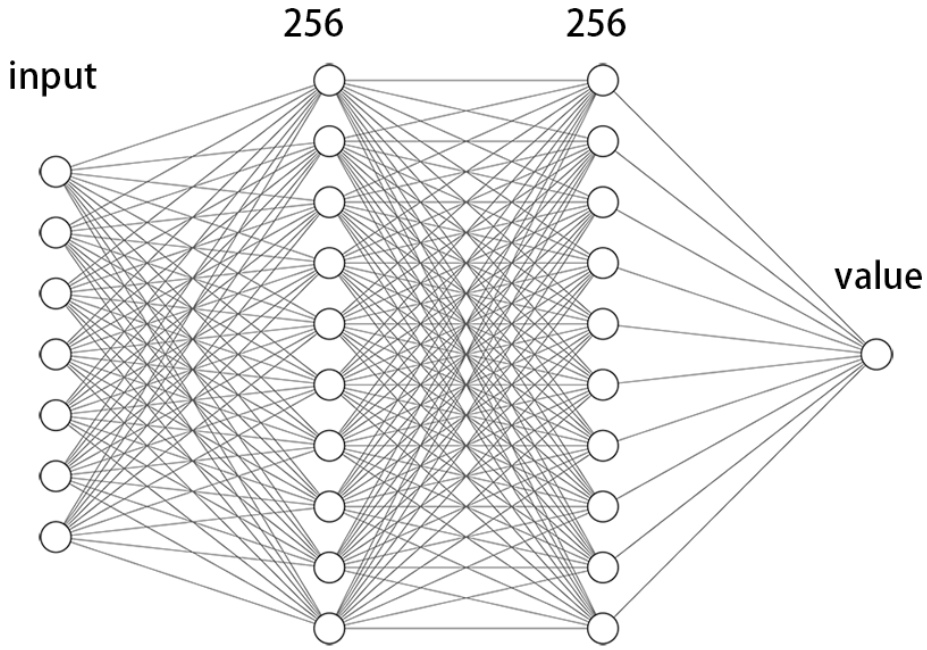


Figure 4.4: The network structure of the critic.

Table 4.2: Reward settings

Reward	Value
Collision Reward r_c	-50
Arrived Reward r_a	100
High Speed Reward r_s	0.2
Lane Change Reward r_l	-0.05

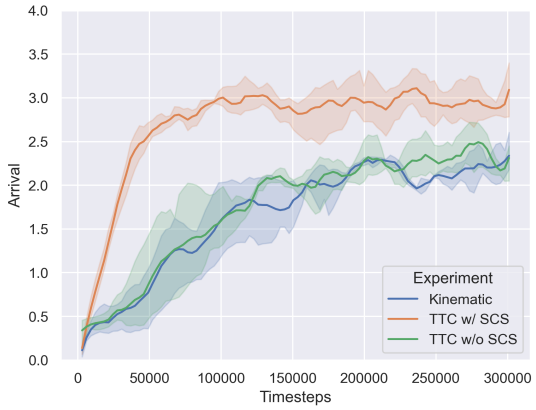
4.4 Results

4.4.1 Simple Task

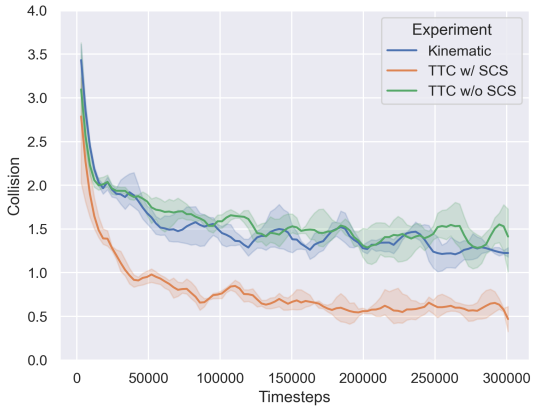
Ablation experiments are conducted to investigate the impact of the state representation and the safety check system. We employ the PPO method uses kinematic representation as the baseline method, which does not employ the safety check system. It needs to be noted that the baseline method cannot equip the safety check system due to a lack of TTC information. Fig. 4.5 shows the performance trajectories on the simple task, which is an average of 10 trials. The simple task consists of 4 autonomous vehicles and 6 environmental vehicles. Fig. 4.5(a) shows the number of arrivals for the conventional method, the method using TTC representation without the safety check system, and the

proposed method. Arrivals converge to 2.31, 2.33, and 3.1 respectively. When the safety check system is not provided, the conventional method and the method using the TTC representation have similar performance. Fig. 4.5(b) shows the number of collisions for the conventional method, the method using TTC representation without the safety check system, and the proposed method. Collisions converge to 1.22, 1.41, and 0.47 respectively. The best collision avoidance ability can be obtained by the proposed method. Fig. 4.5(c) shows the velocity for the conventional method, the method using TTC representation without the safety check system, and the proposed method. The velocity converges to 23.9m/s, 28.1m/s, and 27.8m/s respectively.

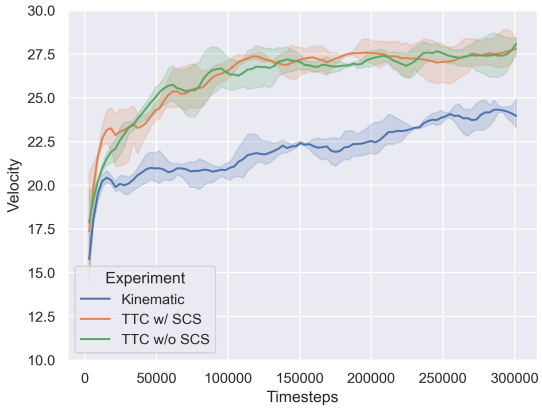
Simulation results show that the proposed method can effectively improve the arrival rate and reduce the collision rate without reducing the efficiency of autonomous vehicles.



(a) The number of arrivals.



(b) The number of collisions.

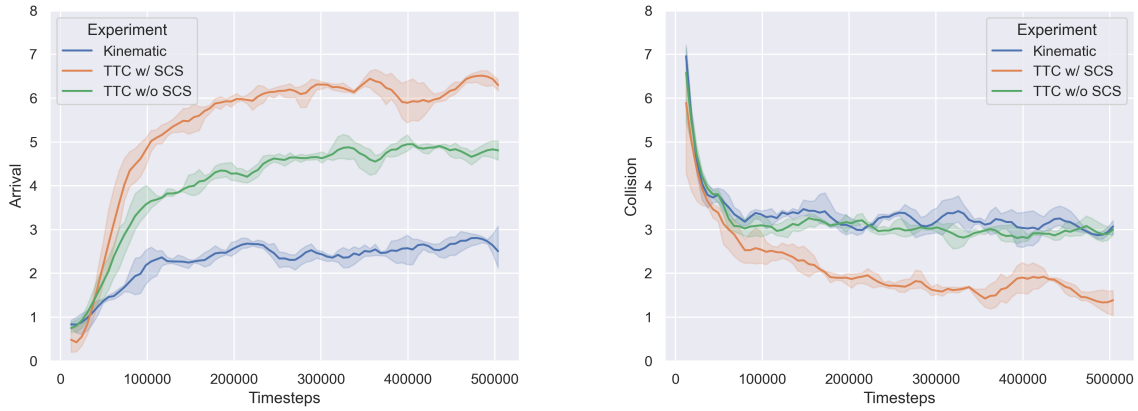


(c) Velocity of autonomous vehicles.

Figure 4.5: The performance trajectories in the simple task, where each data point is the average of 10 trials (with standard deviation).

4.4.2 Hard Task

Fig. 4.6 shows the performance trajectories on the hard task, which is an average of 10 trials. The hard task consists of 8 autonomous vehicles and 8 environmental vehicles. An obstacle is in the middle of the main lane.



(a) The number of arrivals.

(b) The number of collisions.

(c) Velocity of autonomous vehicles.

Figure 4.6: The performance trajectories in the hard task, where each data point is the average of 10 trials (with standard deviation).

Fig. 4.6(a) shows the number of arrivals for the conventional method with kinematic representation, the method using TTC representation without the safety check system, and the proposed method. Arrivals converge to 2.5, 4.8, and 6.3 respectively. As mentioned earlier, kinematic state representation would be affected by the number or order of surrounding vehicles. The effect would be magnified in a dense traffic situation. As a result, the conventional method performs worse than methods using TTC representation, even without the safety check system. Fig. 4.6(b) shows the number of collisions for the

conventional method, the method using TTC representation without the safety check system, and the proposed method. Collisions converge to 3.0, 3.0, and 1.4 respectively. The proposed method has the best collision avoidance ability. Fig. 4.6(c) shows the velocity for the conventional method, the method using TTC representation without the safety check system, and the proposed method. The velocity converges to 23.7m/s, 23.0m/s, and 27.1m/s respectively. In a dense traffic environment, the speed of each algorithm decreases slightly more than in the simple task. Generally, simulation results show that the proposed method can deal with a dense traffic scenario more than other approaches.

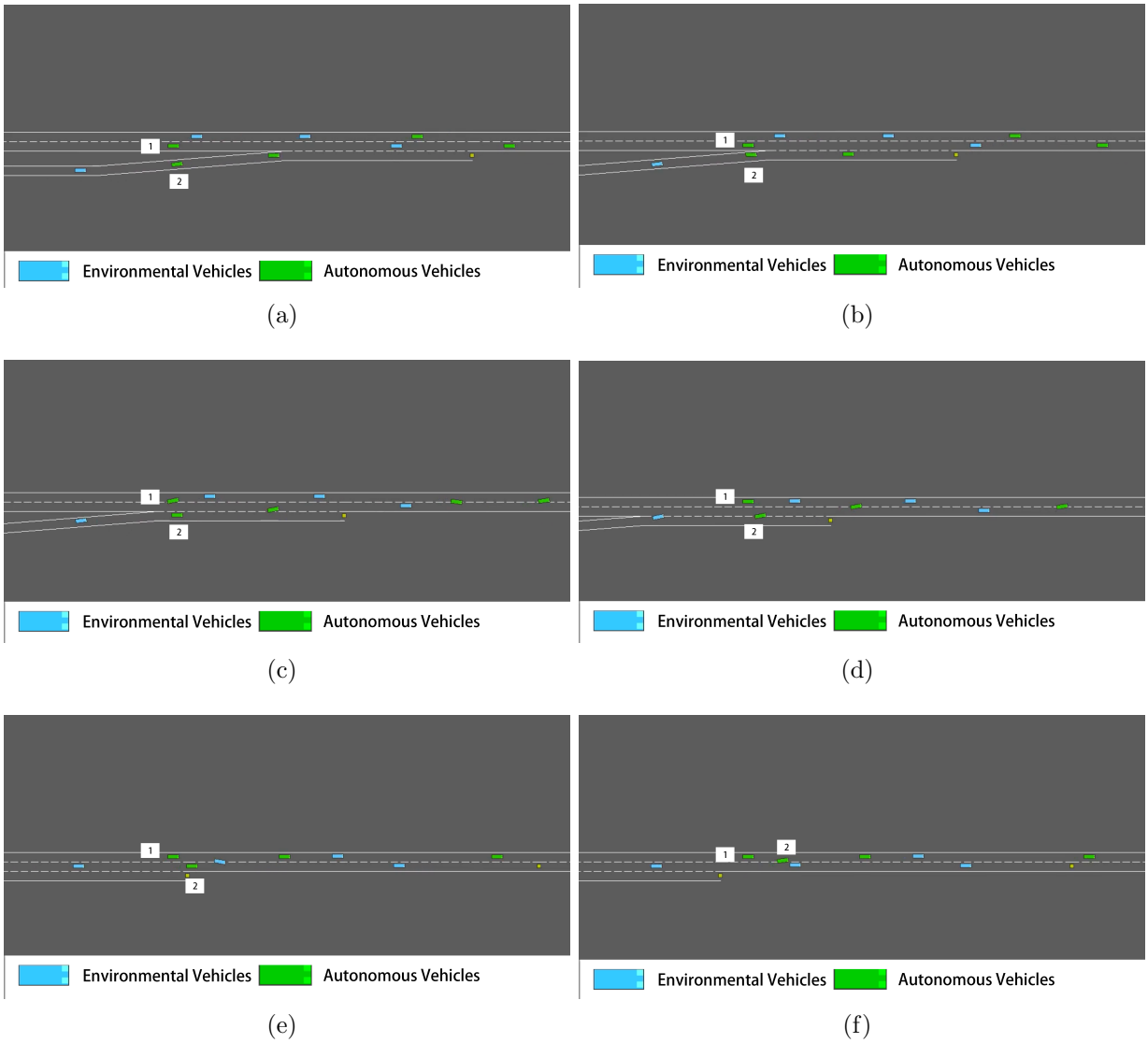


Figure 4.7: Simulation snapshots of the controller trained by the proposed method.

Simulation snapshots of the controller trained with the proposed method are shown in Fig. 4.7. It can be observed in Fig. 4.7(a) and (b), that the first autonomous vehicle is driving on the main road. The second autonomous vehicle is driving on the merging

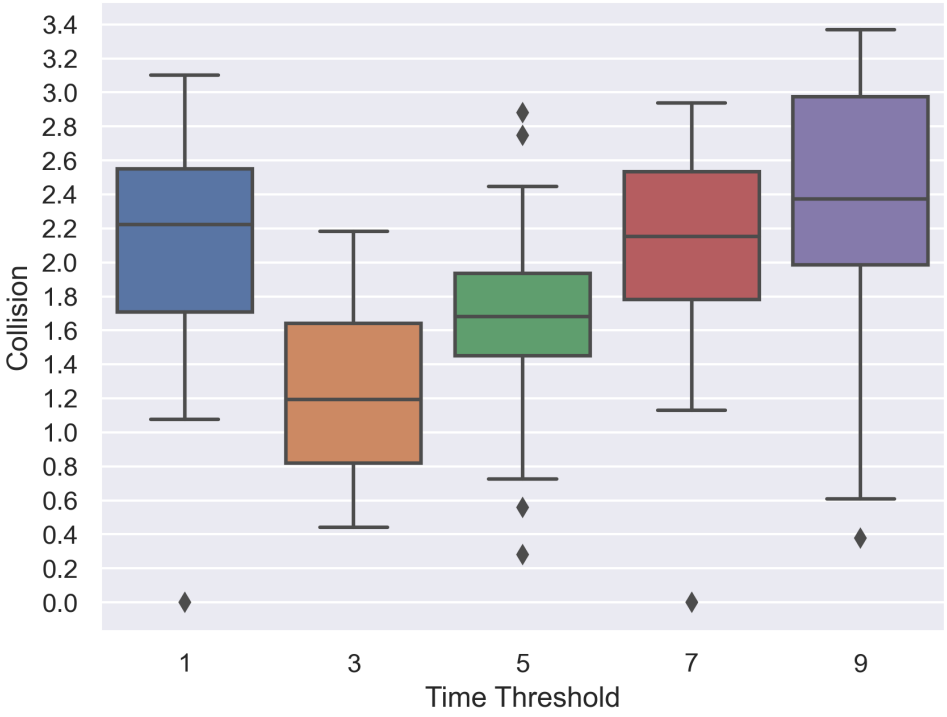


Figure 4.8: Collision evaluation experiments with different safety time thresholds.

road. As shown in Fig. 4.7(c), when the distance between the two vehicles is close, which means the TTC is less than the safe time threshold, the action output by DRL will be replaced by a safer action. Therefore, the first autonomous vehicle chooses to turn left into another lane to avoid the potential collision, rather than go straight to maximize the expected return. In Fig. 4.7(d) and (e), the second vehicle successfully merges into the main lane. As seen in Fig. 4.7(f), the second vehicle makes a lane change to avoid a probable collision with an environmental vehicle.

Furthermore, to examine the effect of different safety time thresholds, evaluation experiments were performed. The safety time threshold is an indicator for evaluating the degree of danger of a state. When TTC is less than the safety time threshold, the actions output by the RL method will be replaced with safer actions by the safety check system. Therefore, we changed the safety time thresholds to 1s, 3s, 5s, 7s, and 9s, respectively. The corresponding controllers are trained in the hard task. Each controller is evaluated 50 times. The results of the evaluation experiments are shown in Fig. 4.8. As it can be observed, when the time threshold is 3s, the lowest collision can be obtained. As the safety time threshold increases, the number of collisions increases accordingly. Since it is still relatively safe when TTC is 7s or 9s, it is not ideal to replace actions too early. Especially there is a great fluctuation when the time threshold is 1s. It is difficult to

completely avoid a collision since the emergency response time is insufficient. As a result, the best performance and safety can be obtained by setting the safety time threshold to 3s.

4.5 Conclusion

In this study, we propose a safety check system based on time-to-collision. The central concept is that the action output by the DRL controller should be replaced with a safer action chosen by the safety check system when an agent detects a potential collision, i.e., the TTC is below the time threshold. A ramp merging task is used to examine the effects of the proposed method.

The simulation results show that the proposed approach can successfully enhance the arrival rate and decrease the collision rate even in crowded traffic scenarios. Furthermore, evaluation experiments are conducted to examine the performance of the safety check system with different time thresholds.

Chapter 5

Visualizing Deep Q-learning to Understand Behaviors of Swarm Robotic System

5.1 Introduction

Swarm robotics (SR) [2][3] is the study of how systems composed of multiple robots can be employed to perform collective tasks that are beyond the abilities of a single robot. There is no centralized control, and robots can only have partial information about the environment. In particular, a robotic swarm exhibits the following advantages; Fault-tolerance, Flexibility, and Scalability. Fault tolerance means the system can still achieve the task while some individuals cannot work. Flexibility means the robotic swarm can cope with similar environments and tasks. Scalability is the ability to complete the assigned work with various group sizes.

The design methods in SRS can be mainly divided into two approaches, i.e., behavior-based design method and automatic design method [6]. A trial-and-error process is used in behavior-based design to develop, test, and refine each robot's individual behavior until the robots exhibit the desired collective behavior [12] Behavior-based design methods have some advantages over other methods, such as simplicity, modularity, reusability, and adaptability. However, this method requires expert knowledge, and the system's performance is entirely dependent on the human designer. Another design method is automatic design [13], which means the controller is automatically created by turning the design problem into an optimization problem. Reinforcement Learning (RL) is a representive automatic design method. Moreover, by developing end-to-end control strategies, Deep Q-Network (DQN), one of the most popular RL algorithms is shown as an effective method for SRS problems. However, because DQN is a black box, it is not easy to explain why it

gives us successful results. Therefore, a novel method of visualizing and interpreting the decision-making process of DQN is proposed in this study.

In this chapter, we present a method of analyzing the latent representations learned by DQN on SRS. The proposed method records the neural activations of the last hidden layer, and then applies Gradient-weighted Class Activation Mapping and Deconvolutional Network for visualization. Gradient-weighted Class Activation Mapping (Grad-CAM) [62] is an Explainable Artificial Intelligence technique that creates a heat map highlighting important regions in the input image, so that we can describe which parts of an input image led the model to its final decision. Deconvolutional Network (Deconvnet) is adopted to generate the reconstructed image and display what information is being preserved in the deep features. A round trip task is employed to illustrate the effects, where the robotic swarm needs to visit two locations alternatively as many times as possible.

Simulation results show that the proposed method is able to interpret the policies learned by DQN in a round trip task. This is very valuable for increasing our understanding of Deep Reinforcement Learning (DRL). However, a problem in explainable reinforcement learning is the lack of an absolute standard. Unlike the classification task, it is difficult to say whether an action is correct or wrong in DRL. This might be an important direction for explainable reinforcement learning in the future.

5.2 Experimental Settings

In this study, 3D physics engine *Unity* is used to create simulation environments. And open source framework for neural networks – *Chainer* is used to create the CNN. *Unity*'s script is written in *C#* while *Chainer*'s script is written in *Python*. Table 5.1 summarize the software environment of this computer simulation.

Content	**Software**	**Computer Languages**
Operating system	Ubuntu 18.04.2 LTS	
Physics engine	Unity3D 2017.3.0b1	C#
Open source library	Chainer 5.3.0	*Python*3.7.3
Computing platform	CUDA 9.0	C++

Table 5.1: Software environment of the computer simulation

The entire experiment was done on a desktop workstation. Table 5.2 summarize the hardware specifications of this desktop workstation.

Content	Specifications
Memory	31.4 GiB
Processor	Intel core i7-6700k @ 4.00GHz × 8
Graphics Processor	GeForce GTX 1080

Table 5.2: Hardware specifications of the desktop workstation

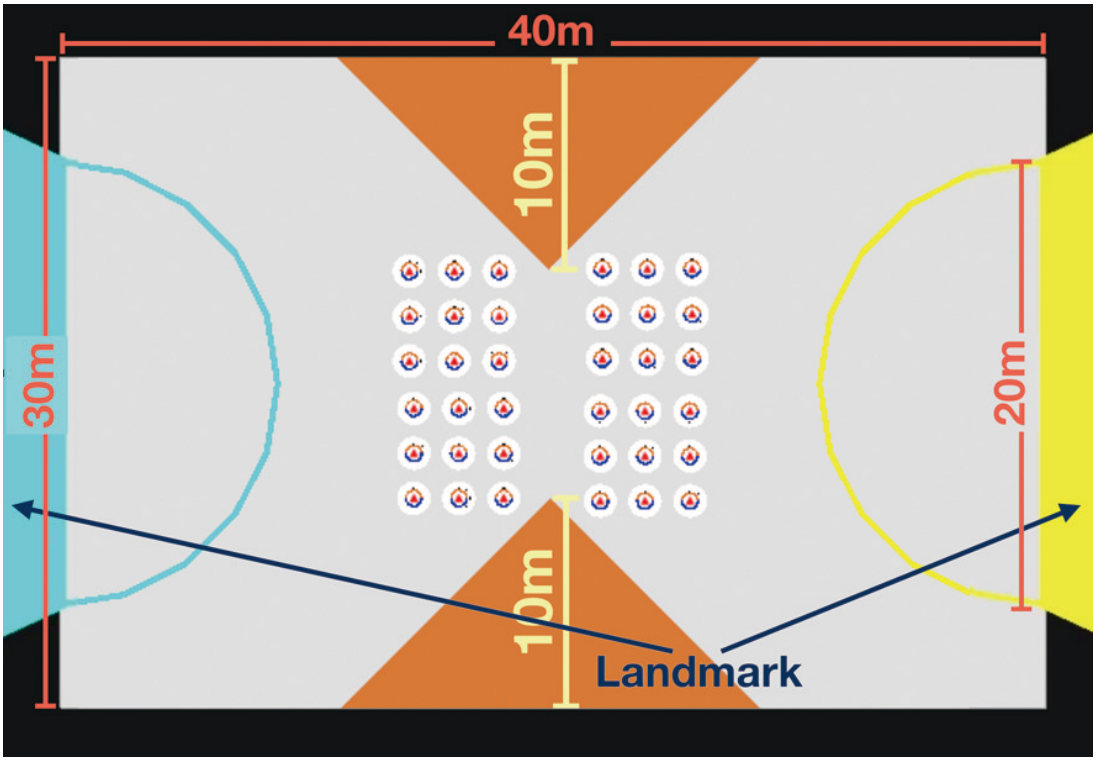


Figure 5.1: Top view of the environment

5.2.1 Environment

The experimental environment is shown in Fig. 5.1. The field is a rectangular area of 40m × 30m surrounded by a 6m high wall. Two landmarks (10m in radius, 6m in height) with different colors are placed on the left and right sides of the field, which show the direction of the destination of the robot. Two orange obstacles are located in the outer area, blocking two-thirds of the way. Obstacles limit the robot group to observe and choosing its path, which makes the task more difficult. The inside view of the environment is shown in Fig. 5.2.

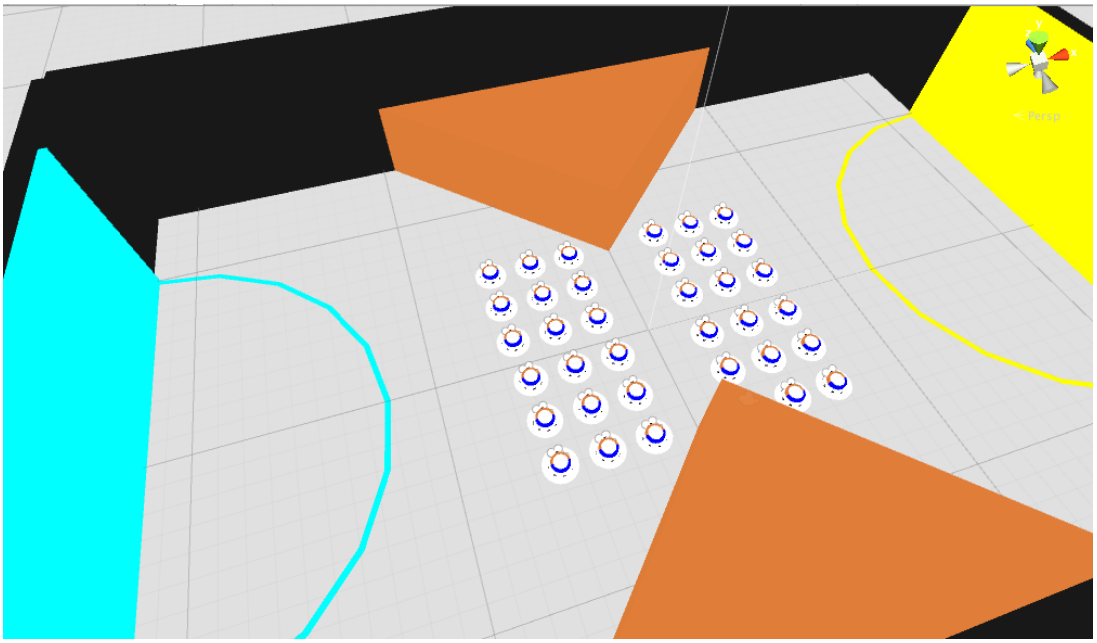


Figure 5.2: Inside view of the environment

5.2.2 Agent

The robot in this experiment is illustrated in Fig. 5.3. The diameter of the robot is 1.0 m. The triangle at the center represents the robot's face direction. Each robot is equipped with eight IR sensors, one LED, one camera, and two motors. Eight IR sensors are positioned in various directions to detect the distance between the robot and other objects within 1.0m and return the distance. The LED is mounted on the top of the body, and the coloring pattern can be changed. When the target is yellow, the robot's LED will emit blue light. Similarly, when the target is blue, LED will turn to yellow light. The agent is also equipped with an electric compass.

The camera produces an RGB image of 128 × 128 pixels whose angle of view is horizontal 90 and vertical 90. Fig. 5.4 displays an example of an image obtained by the camera. The movement is performed by a differential driving system with two sets of motors and wheels. The maximum movement speed is set to 1.0. The output is selected from the six target velocities shown in the following equation.

$$\begin{aligned}(v_{left}, v_{right}) &= ((0,0),(1,1),(0,1),(1,0) \\ &= (0,0.5),(0.5,0))\end{aligned} \tag{5.1}$$

where controls motors to achieve the target velocity, v_{right} is the target velocity of the right wheels, and v_{left} is the target velocity of the right wheels. The agent is also equipped with an electric compass.

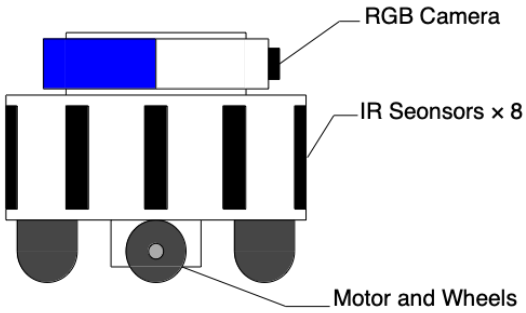


Figure 5.3: Top view of the agent.

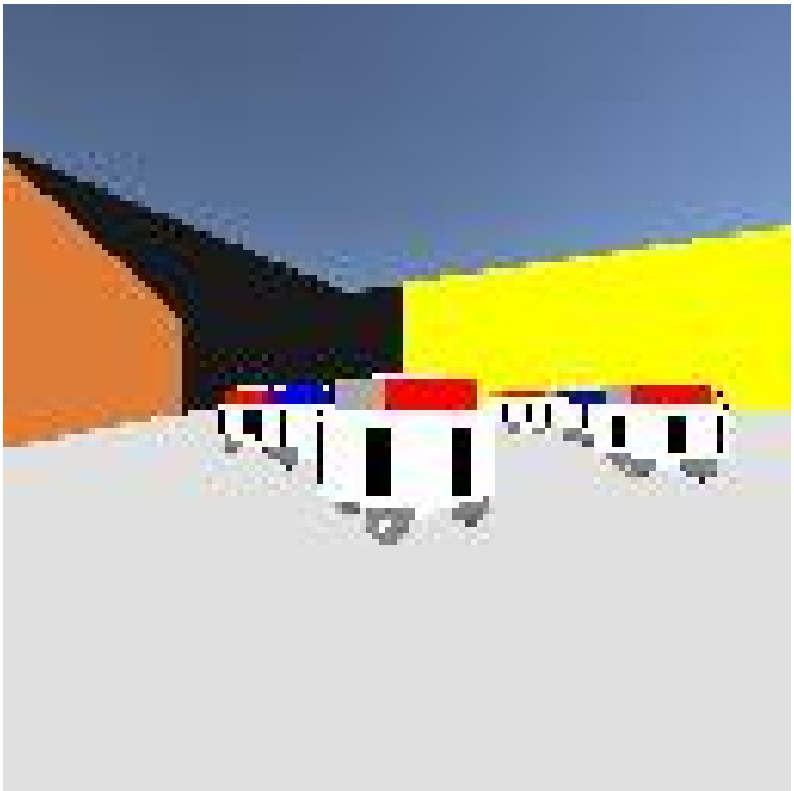


Figure 5.4: An example of the images obtained by the camera.

Table 5.3: Reward settings

r_d	r_l	r_f	r_p
5.0	1.0	5.0	- 1.0

5.2.3 Round-Trip Task

In the beginning, 36 robots are located in the center of the field, and the initial destination of each robot is set randomly. In order to train the agents to have the ability to recognize landmarks and obstacles, they are set as yellow, blue, and orange respectively. Once a robot arrives at its current destination, i.e., gets into the corresponding semicircle area, the destination of the robot is automatically set to another one. The goal of the robotic swarm is to travel between two destinations as much as possible. The quality of the round-trip task will be evaluated by the number of times each robot reaches the landmarks and the number of collisions.

5.2.4 Reward Settings

The DQN approximates the value function by combing the RL with a DNN. The value function plays a crucial role in RL, as it determines the expected future rewards for an agent in a given state. However, if the optimal value function is very complex, it may be challenging to learn an accurate low-dimensional representation of it using traditional RL algorithms. This is because the complexity of the value function can make it difficult to accurately estimate and represent using a small number of parameters or a simple function approximation method. In this case, one important task in RL is how to simplify a seemingly complicated task into a more simple task. Therefore, the reward system, this value function is much easier to learn. To make appropriate behavior emerge we customized the system like this.

The reward given to the robots at each time step is composed of four parts: (1) destination (r_d); a robot receives the reward r_d when arrives at its current destination. (2) landmark (r_l); a robot gets the reward r_l if its camera observes more pixels of its current landmark than the last time step. (3) fellow robot (r_f); a robot gets the reward r_f if its camera observes more pixels of the LED of fellow robots (with the same current destination) than the last time step. (4) penalty for collision (r_p); a robot gets a penalty <0.5 m) to other objects. Specific settings are as shown in Table.

First rewards system encourages robots to focus on observing the corresponding landmark directly.

Second rewards system encourage robots to looking for fellow robots (with the same

Table 5.4: Reward settings

r_d	r_l	r_f	r_p
1.0	5.0	5.0	- 1.0

Table 5.5: Reward settings

r_d	r_l	r_f	r_p
5.0	1.0	5.0	- 1.0

current destination).

Final rewards system relate to IR sensors values. Encouraging robots to reduced the collisions the most.

5.2.5 Hyperparameters Settings

In this section, we will show the hyperparameters settings.

- **Episode = 100**
- **Time step $\leq$ 3000**
- **Discounted future reward (γ) = 0.95**
- **Experience replay buffer size = 3000**
- **Initial exploration time = 3000**
- **Batch size = 32**
- **ϵ-greedy degradation**

  The ϵ-greedy strategy (i.e., at each time step, with probability ε select an action randomly, otherwise, select the action with the highest Q-value) is applied to produce the final control policy, in which $\epsilon = 1.0$ for episode 0, $\epsilon = 1.0 - t\Delta 310^{-5}$ for episodes 1 to 8, and $\epsilon = 0.1$ after episode 9, where t is the number of cumulative timesteps.

5.3 Experiment 1: Visualizing Training Process by Deconvolutional Network

In this section, a novel approach of visualizing the DQN's decision-making process is presented employing a deconvolutional network (Deconvnet). Computer simulations of the round-trip task are employed to demonstrate the proposed approach. The simulation experiments are performed 5 times.

5.3.1 Network Architecture

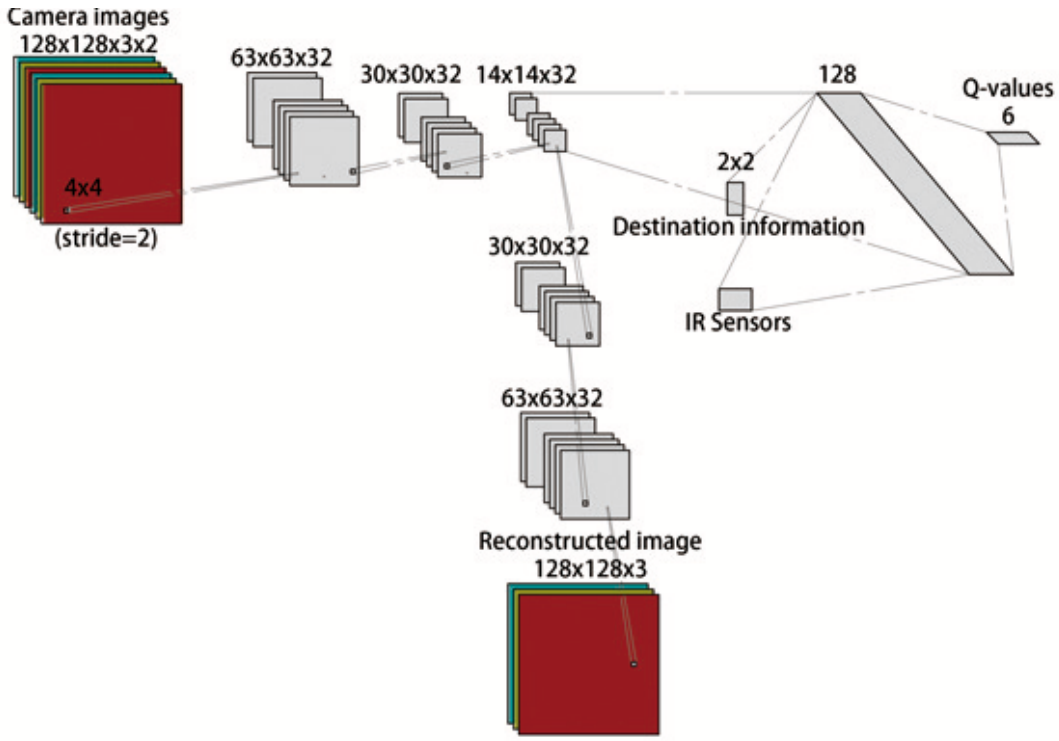


Figure 5.5: Architecture of the deep neural network.

Fig. 5.5 shows the network architecture of the proposed method. Similar to the traditional DQN architecture, the proposed network contains a series of convolutional layers and fully-connected layers. Each convolutional layer has 32 filters of which receptive field size is 4×4 and the stride is set to 2. The input to the convolutional layer is raw RGB pixels from the camera of the past two time steps (128×128×3×2). There are three hidden layers, with 63, 30, and 14 units respectively. After the last convolutional layer, we add a series of deconvolutional layers to synthesize a reconstructed image with the same size as the input. The output of the last convolutional layer is taken along with the distance information from IR sensors and destination information as inputs by the first fully-connected layer

Table 5.6: Reward settings

r_d	r_l	r_f	r_p
5.0	1.0	5.0	- 1.0

(128 neurons). The second full-connected layer (6 neurons) holds the estimated Q-values corresponding to the six motor patterns. We choose the first reward system mentioned in the last section.

5.3.2 Loss Function

We define different loss components that can be used to train the network.
Bellman Error ($L_{bellman}$): The usual value function estimation error.

$$L_{bellman}(\theta) = (r + \gamma max_{a'} Q(s', a'; \theta) - Q(s, a; \theta)^2 \tag{5.2}$$

Reconstruct Error ($L_{reconstructed}$) : Reconstruct Error is the mean squared error between reconstructed image s_t^* and original input s_t.

$$L_{reconstructed}(\theta) = \frac{1}{2}||s_t^* - s_t||_2^2 \tag{5.3}$$

Finally, the model is trained to minimize the sum of two weighted linear losses. Hyperparameter λ is introduced for weighting different loss components.

$$L_{final}(\theta) = \lambda L_{bellman}(\theta) + (1 - \lambda) L_{reconstruct}(\theta) \tag{5.4}$$

5.3.3 Results

In order to examine the interpretability under different loss component weights, three settings are performed in the loss function we mentioned in the last section. Five trials are performed for each setting. To evaluate the performance of different λ settings, which λ = 1.0, 0.5, 0.1 respectively, we recorded how many times the robots have arrived at their destinations. Fig. 5.6 shows the trajectories of arrivals during the learning process in experiments.

In general, we found $\lambda = 1.0$ (which has no interpretability) can work well on round-trip tasks, where the robots arrived at their destinations 74 times at maximum (averaged by 5 trials). Decreasing λ to 0.5 and 0.1 can get a better reconstruction image visually. The best performance was obtained in the experiment $\lambda = 0.5$, in which robots reached

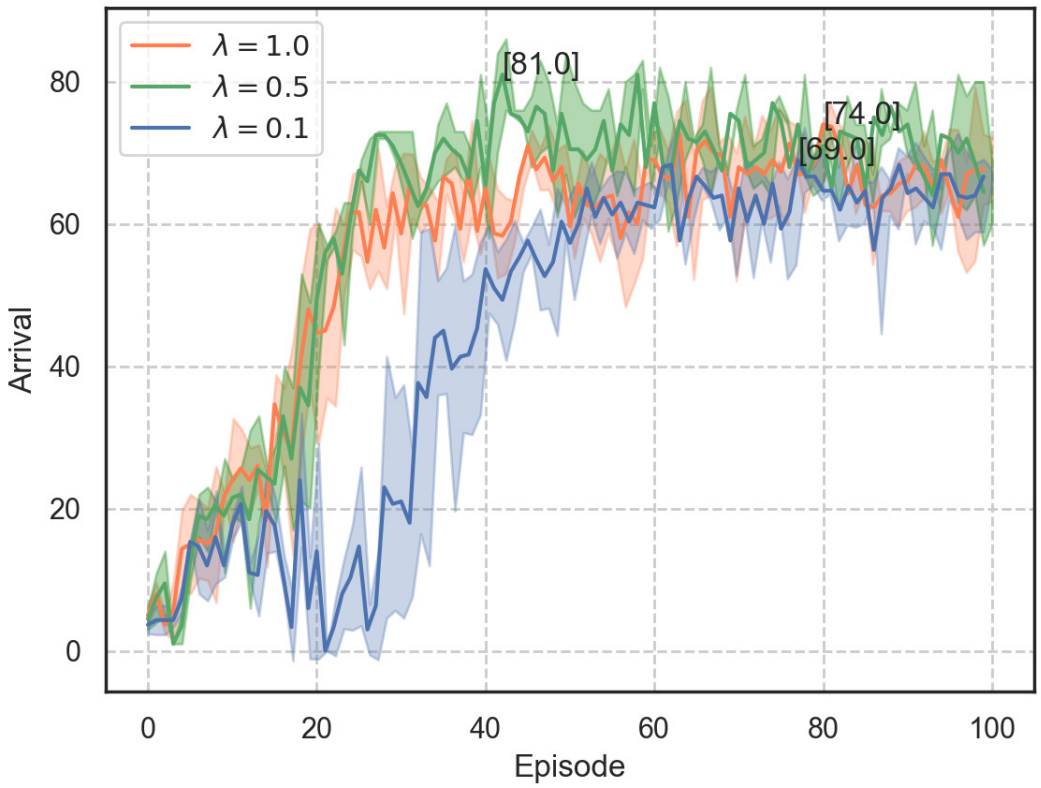
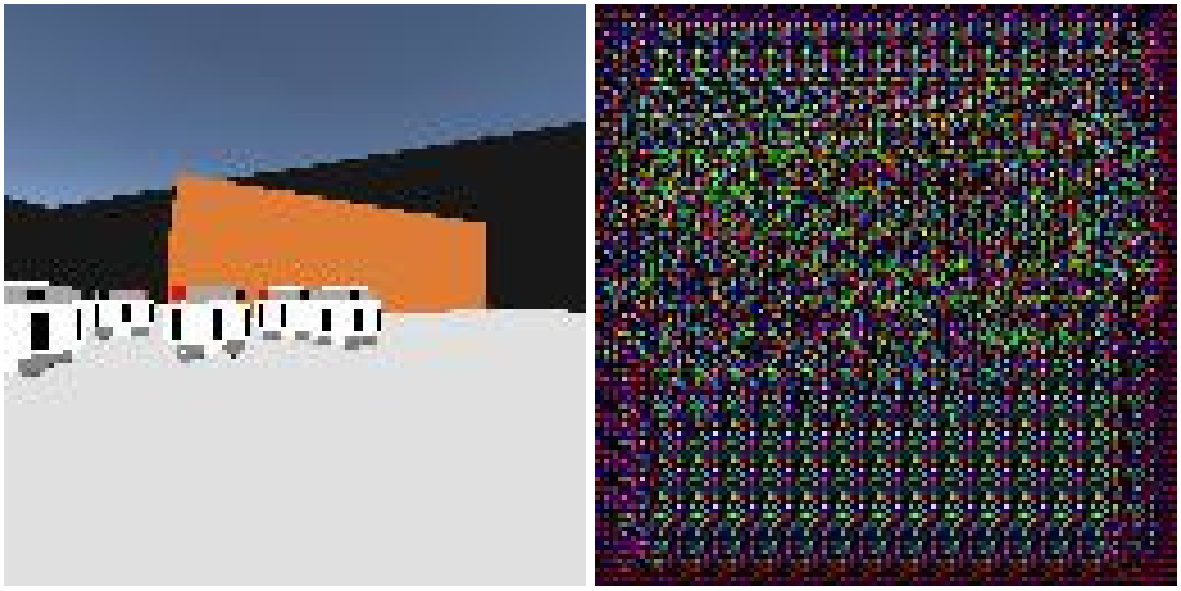


Figure 5.6: Trajectories of arrivals, where each data point is the average of 5 trials (with standard deviation).

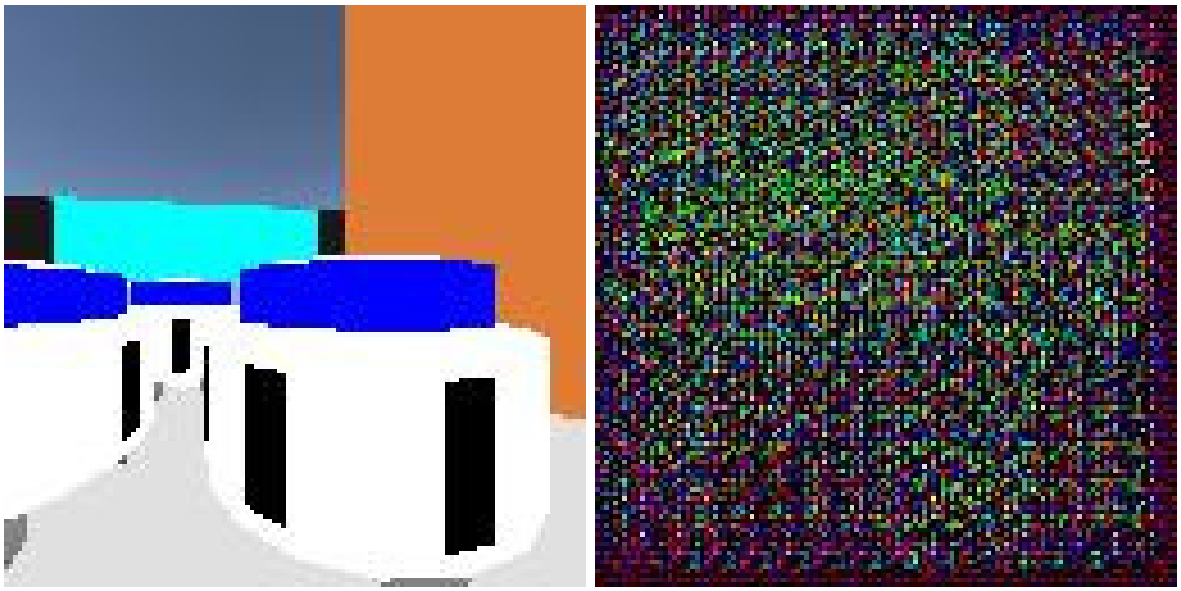
the maximum of 81 arrivals. However, decreasing λ to 0.1 results in slower convergence and a slight loss in performance, where the maximum arrivals were 69 (drop by 15%).

By considering the converge rate, swarm performance, and reconstructed image quality, the setting $\lambda = 0.5$ seems can keep the balance between the bellman loss and reconstruct loss. Examples for $\lambda = 1.0$ are shown in Fig. 5.7 and Fig. 5.8. Examples for $\lambda = 0.5$ are shown in Fig. 5.9 and Fig. 5.10. For $\lambda = 1.0$, the reconstructed image contains only some textures that are hard to interpret. For $\lambda = 0.5$, with the training process, the interpretability of reconstructed images has improved. As the reconstructed images change, we can observe the learning process of CNN. Fig. 4.10(a) shows the input image in episode 14. Fig. 4.10(b) shows the reconstructed image in episode 14. Different regions in the input image are segmented simply, reconstructed image contains color information, which is an improvement over the result of $\lambda = 1.0$. It can be observed from Fig. 4.11(b), the edges of the landmarks and fellow robot LEDs are preserved by the trained model in deep features.



(a) The input image. (b) The reconstructed image.

Figure 5.7: Experiments using $\lambda = 1.0$ in episode 9. The experiment consists of 100 episodes.



(a) The input image. (b) The reconstructed image.

Figure 5.8: Experiments using $\lambda = 1.0$ in episode 88. The experiment consists of 100 episodes.

(a) The input image. (b) The reconstructed image.

Figure 5.9: Experiments using $\lambda = 0.5$ in episode 14. The experiment consists of 100 episodes.

(a) The input image. (b) The reconstructed image.

Figure 5.10: Experiments using $\lambda = 0.5$ in episode 66. The experiment consists of 100 episodes.

5.4 Experiment 2: Visual Policy Rationalizations Using Grad-CAM for Different Reward Settings

The value function or strategy of an agent is estimated by DRL approaches using neural networks. Therefore, it is necessary to reason why specific measures are taken. However, there is no simple connection between the weights in a neural network and the function it is trying to approximate. It is not easy to inspect the weights and draw intuitive conclusions about the network.

Knowing why a decision is made might help improve the understanding of the agent. One such explainability method that calculates class-based discriminative features is a Gradient-weighted Class Activation Map (Grad-CAM). It generates visual explanations for CNN-based classifiers and can also be used for DRL methods. In contrast to the Prediction Difference Analysis method which can also be applied in a wide variety of CNNs, the Grad-CAM method is computationally more tractable and feasible for a significant amount of inputs. This section will explain how this visual rationalization model can be adapted to our study. This section will also provide which reward setting works best and analyze the behavior pattern of a robotic swarm.

5.4.1 Experimental Settings

In the first experiment, the proposed method (i.e, combining deconvolutional network with traditional DQN) was demonstrated to greatly improve the Interpretability without losing swarm performance. Then, in this experiment, we still choose the same network structure. Hyperparameter λ is set as 0.5.

In order to present the performance of swarm under different reward settings, three experiments are performed. The numerical settings are shown in Table 5.7. The robots are rewarded more for observing the landmark of their current destination and observing other fellow robots (with the same destination) in experiment A (using reward setting A) and experiment B (using reward setting B), respectively. In experiment C (using reward setting C), the robots receive more penalty when they are too close to other objects.

Table 5.7: Reward settings

Reward setting	r_d	r_l	r_f	r_p
A	5.0	1.0	5.0	- 1.0
B	1.0	5.0	5.0	- 1.0
C	1.0	1.0	5.0	- 5.0

5.4.2 Grad-CAM Procedure for DRL

Grad-CAM is adopted as our visualization technique in this study, since it is a generic and applicable method for any convolutional neural network (CNN)-based architecture without requiring retraining. Grad-CAM combines gradient information with Class Activation Maps (CAM) to visualize the importance of each input feature map. The following several steps are needed. Firstly, a well-trained DRL model on the round-trip task is prepared, and the neural activations of the last hidden layer are recorded. The Grad-CAM technique computes the gradient of the output Y_c with respect to the feature map A^k of the last hidden layer. Then we calculated a global average pooling to obtain weights α_c^k.

$$\alpha_c^k = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k} \tag{5.5}$$

where Z is the size of the feature map and k is the number of the feature map. α_c^k represents the importance of the activation layer k for predicted action c. The output heatmap $L_{Grad-CAM}^c$ is the weighted combination of feature maps with ReLU.

$$L_{Grad-CAM}^c = ReLU(\sum_k \alpha_c^k A^k) \tag{5.6}$$

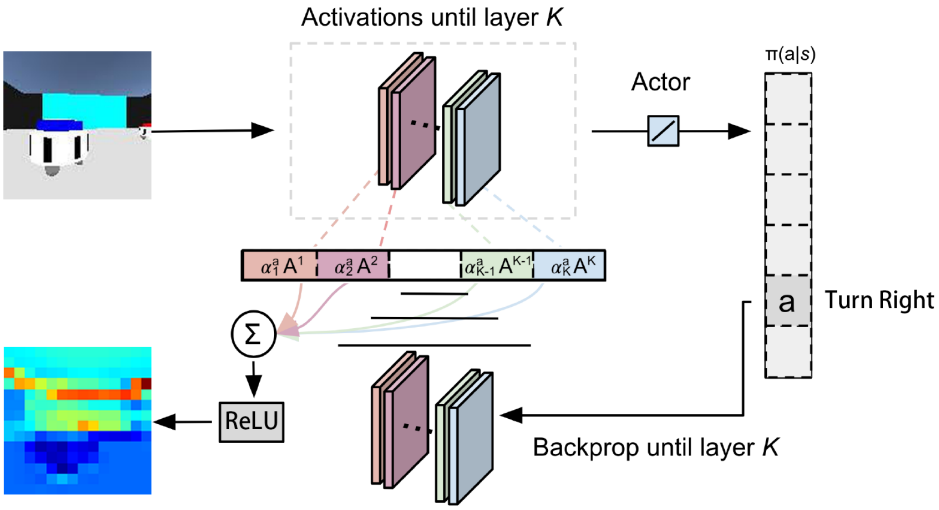


Figure 5.11: Architecture for applying Grad-CAM to DQN.

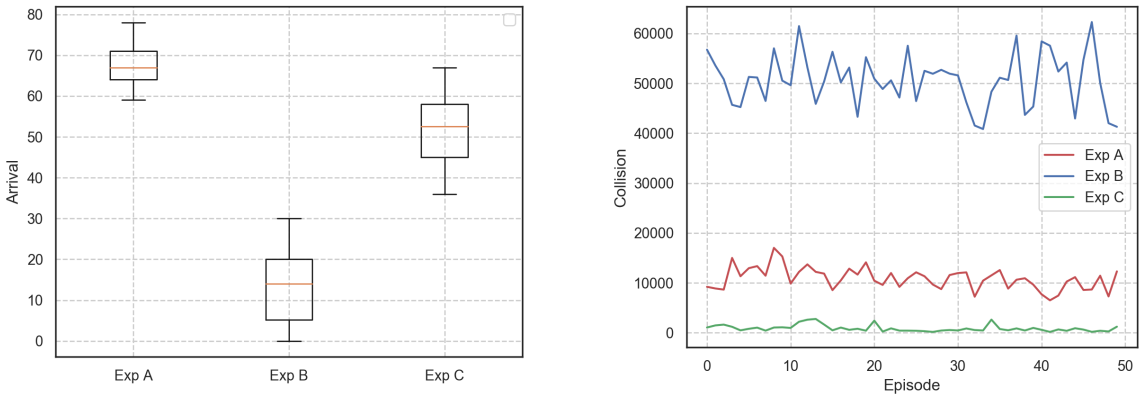
The generated activation map can be extrapolated to the size of the input state. The outcome is a high-quality heat map that shows the areas that are essential to the agent to conduct a particular action, layered on top of the input state. The heat map highlights the areas of the input that have the highest impact on the agent's decision-making process,

allowing for a better understanding of the agent's behavior. By visualizing the regions of the input that the agent focuses on, we can gain insights into the agent's decision-making process and improve its performance. A visual representation of this process is depicted in Fig. 5.11.

5.4.3 Results

We prepared well-trained models with three different reward settings and tested them for 50 episodes each. To evaluate the performance of the developed control policies, we examine how many times have the robots arrived at their destinations and the number of collisions.

Fig. 5.12 shows the arrival and collision of all three experiments during the test. As can be observed in Fig. 5.12(a), Experiment A encourages the robots to focus on observing the landmark directly led to the best performance. Experiment C applying too much collision punishment resulted in lower performance. Experiment B encourages robots more for looking for fellow robots leading to the lowest performance. Fig. 5.12(b) shows the number of collisions. As can be seen, Experiment B led to a significant increase in the number of collisions during the learning process. The collisions in Experiment A stabilized at low levels. The strong punishment in experiment C indeed reduced the collisions the most.



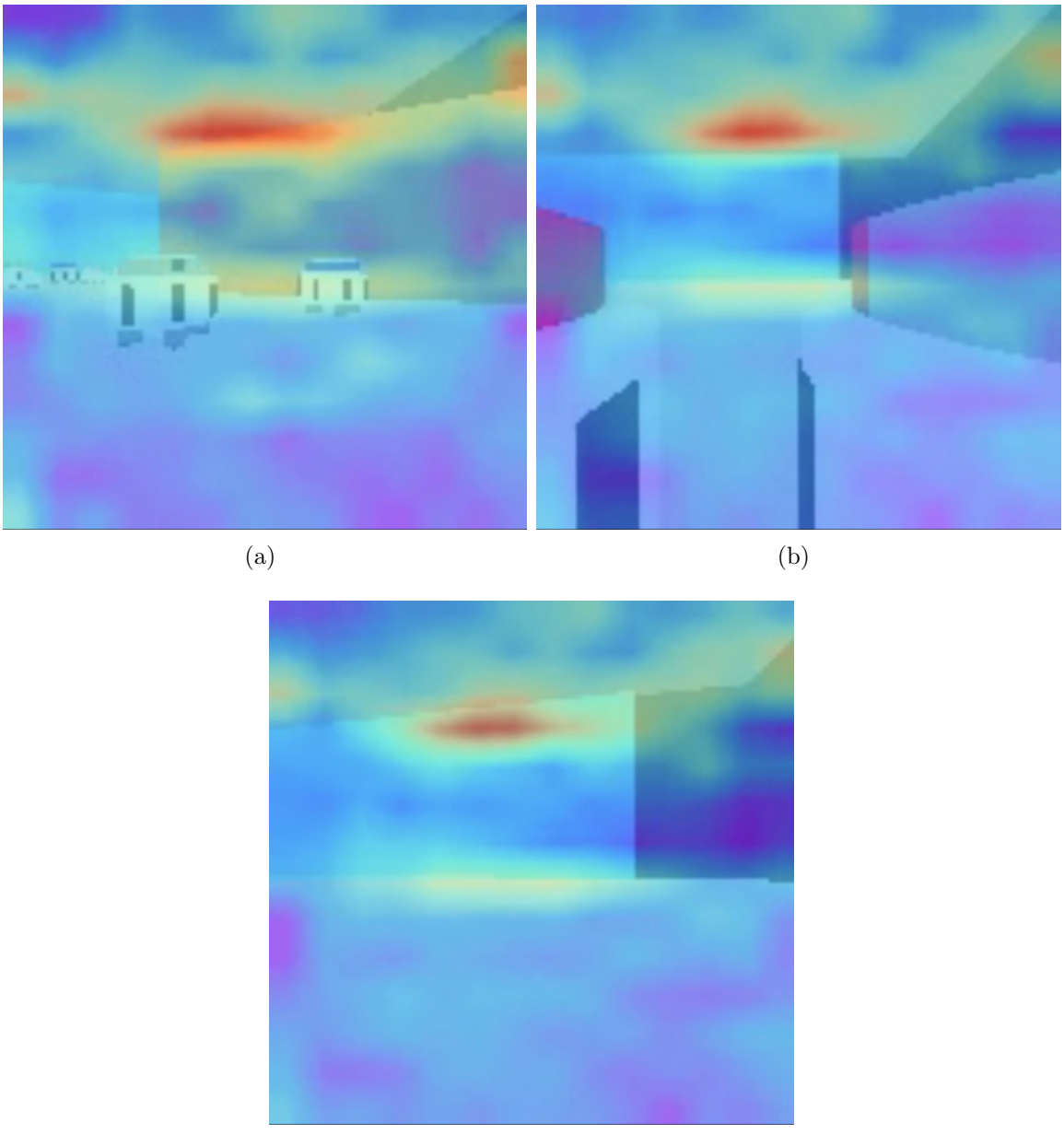
(a) Number of arrivals.

(b) Trajectories of collisions

Figure 5.12: Tests using different reward settings. The test consists of 50 episodes.

Fig. 5.13 shows the Grad-CAM results by reward setting A. The results display information that was processed by the final hidden layer while taking into account input states and chosen action. Since the results highlight the important region, we can see that the part of the convolutional layers concerned is the boundary between objects, especially

between the landmarks and the sky. Therefore, we are able to assume these parts led the model to make its final decision.



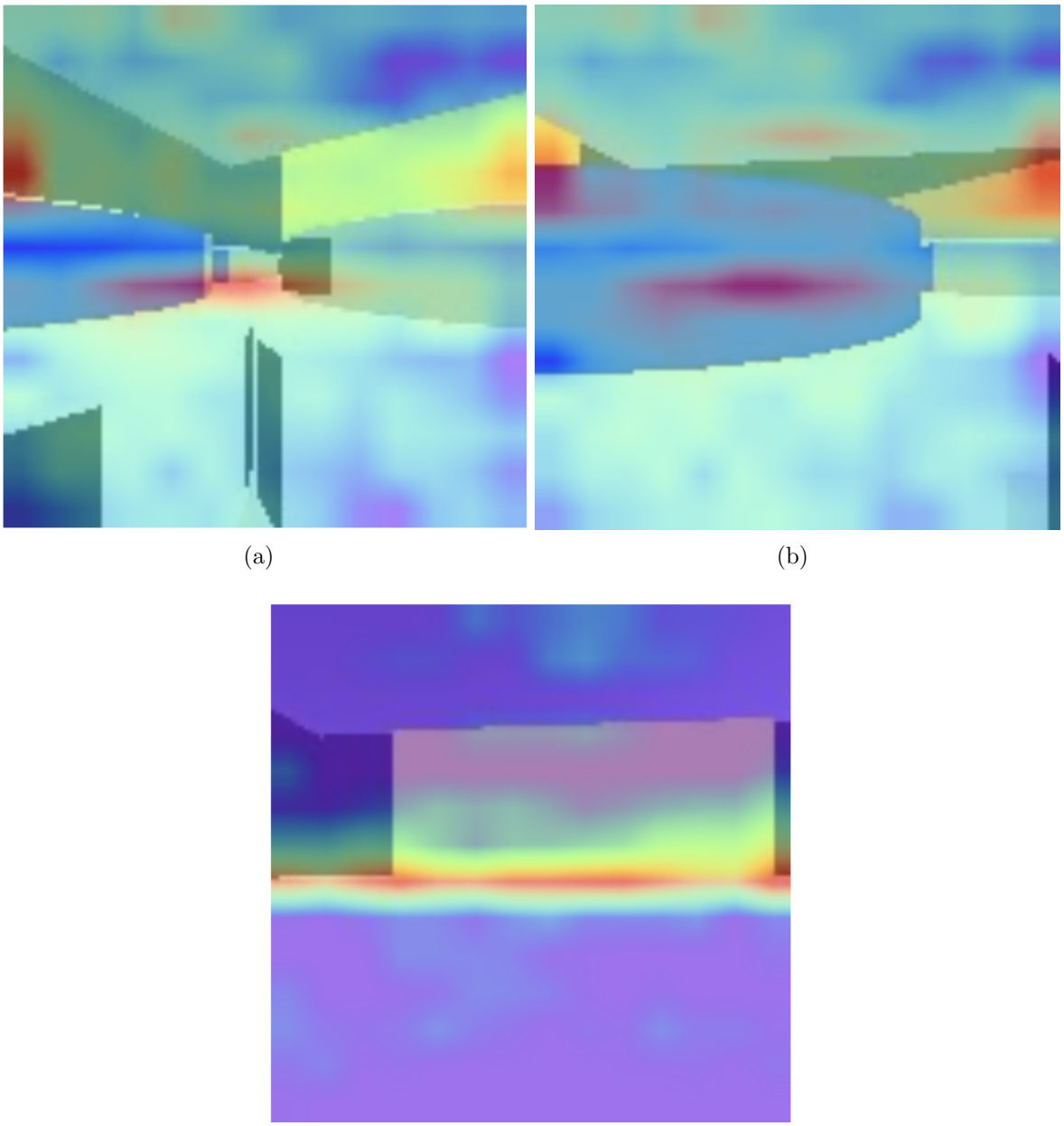
(a) (b)

(c)

Figure 5.13: Grad-CAM results for reward setting A. The Grad-CAM model outputs a class activation map with colors ranging form red to blue. Red/orange indicate a high activation, yellow/green indicate a medium activation and blue indicates a low activation.

Fig. 5.14 shows the Grad-CAM results by reward setting B. It can be clearly observed from the heat map, the fellow robot LEDs are highlighted. If no fellow robot is observed, Grad-CAM will highlight the horizon. In this case, we can assume that the model will

pay more attention to fellow robots' LEDs.



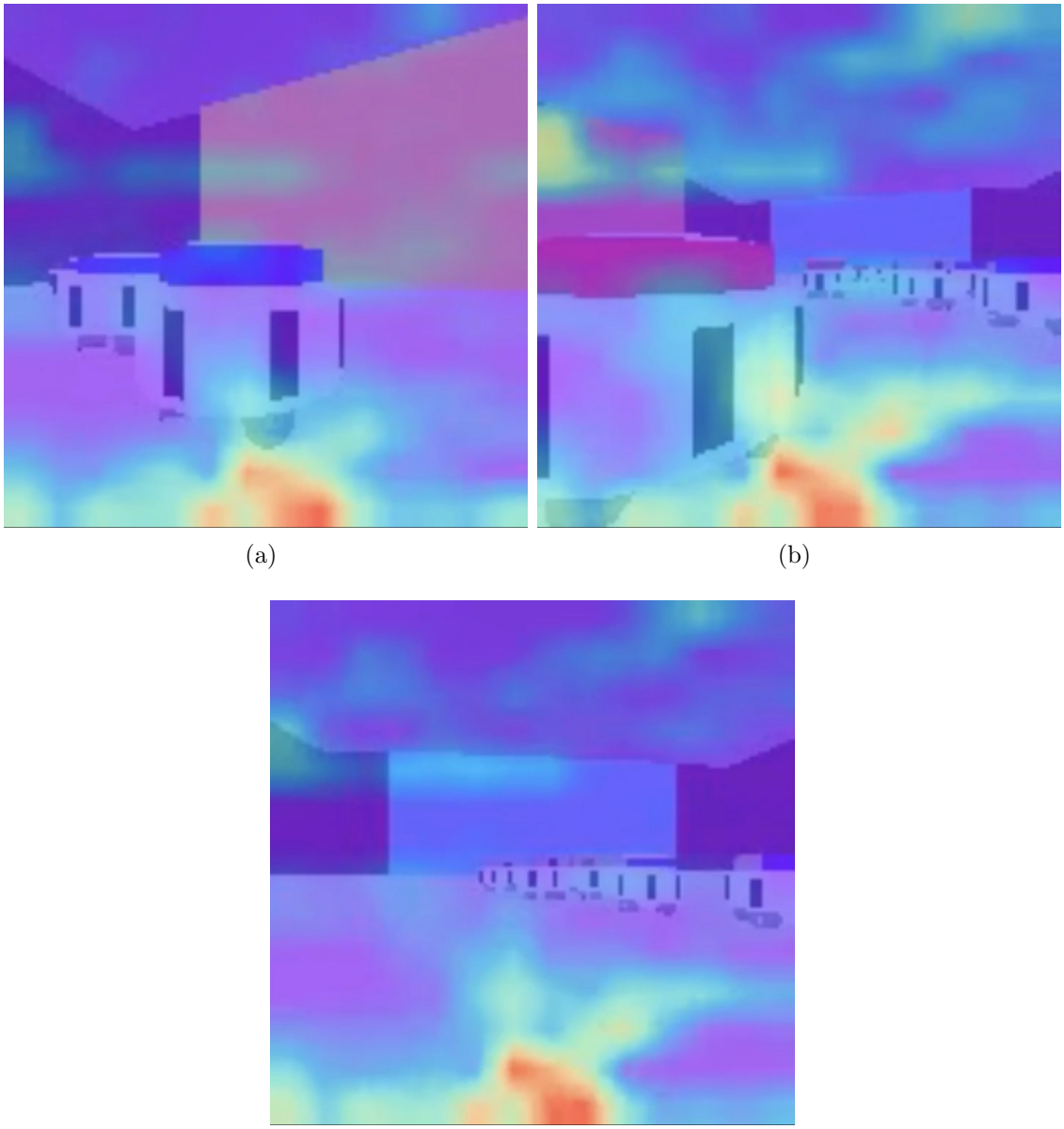
(a) (b)

(c)

Figure 5.14: Grad-CAM results for reward setting B. The Grad-CAM model outputs a class activation map with colors ranging form red to blue. Red/orange indicate a high activation, yellow/green indicate a medium activation and blue indicates a low activation.

Fig. 5.15 shows the Grad-CAM results by reward setting C. We can see that the lower part of the input image is highlighted. So that we can assume that if an object appears in the lower part of the input image, it means that the robot is too close to the object. In order to avoid collisions as much as possible, the model will pay more attention to

this area. It is also worth mentioning that we used to think that obstacle avoidance is performed by the IR sensors. Based on these Grad-CAM results, we found that obstacle avoidance can also be accomplished by the image input at the same time.



(a) (b)

(c)

Figure 5.15: Grad-CAM results for reward setting C. The Grad-CAM model outputs a class activation map with colors ranging form red to blue. Red/orange indicate a high activation, yellow/green indicate a medium activation and blue indicates a low activation.

Considering the input to the convolutional layer is raw RGB pixels from the camera of the past two time steps, we can assume that the trained model is observing the movement of the found object.

Simulation snapshots of the best controller developed by different reward settings are shown in Fig. 5.16, Fig. 5.17 and Fig. 5.18. As it can be clearly observed, the developed control policies for the robotic swarm exhibit different behaviors in the three experiments. In experiment A, the swarm forms a chain-like pattern, in which most robots follow fellow robots in front of them and move in a clockwise circle toward their destinations. The robots in experiment B aggregate and travel between two destinations as a whole. In experiment C, the swarm has the most sparse spatial distribution and each robot acts much more independently to avoid colliding with each other.

Comparing heat maps and simulation snapshots, although they have different aspects, (i.e. simulation snapshots focus on the behavior pattern of the swarm, Grad-CAM focuses on which regions motivate robots to take the selected action from the robot's perspective) they all reflect the same important information. In summary, Grad-CAM can be used to rationalize the decisions made by the DQN.

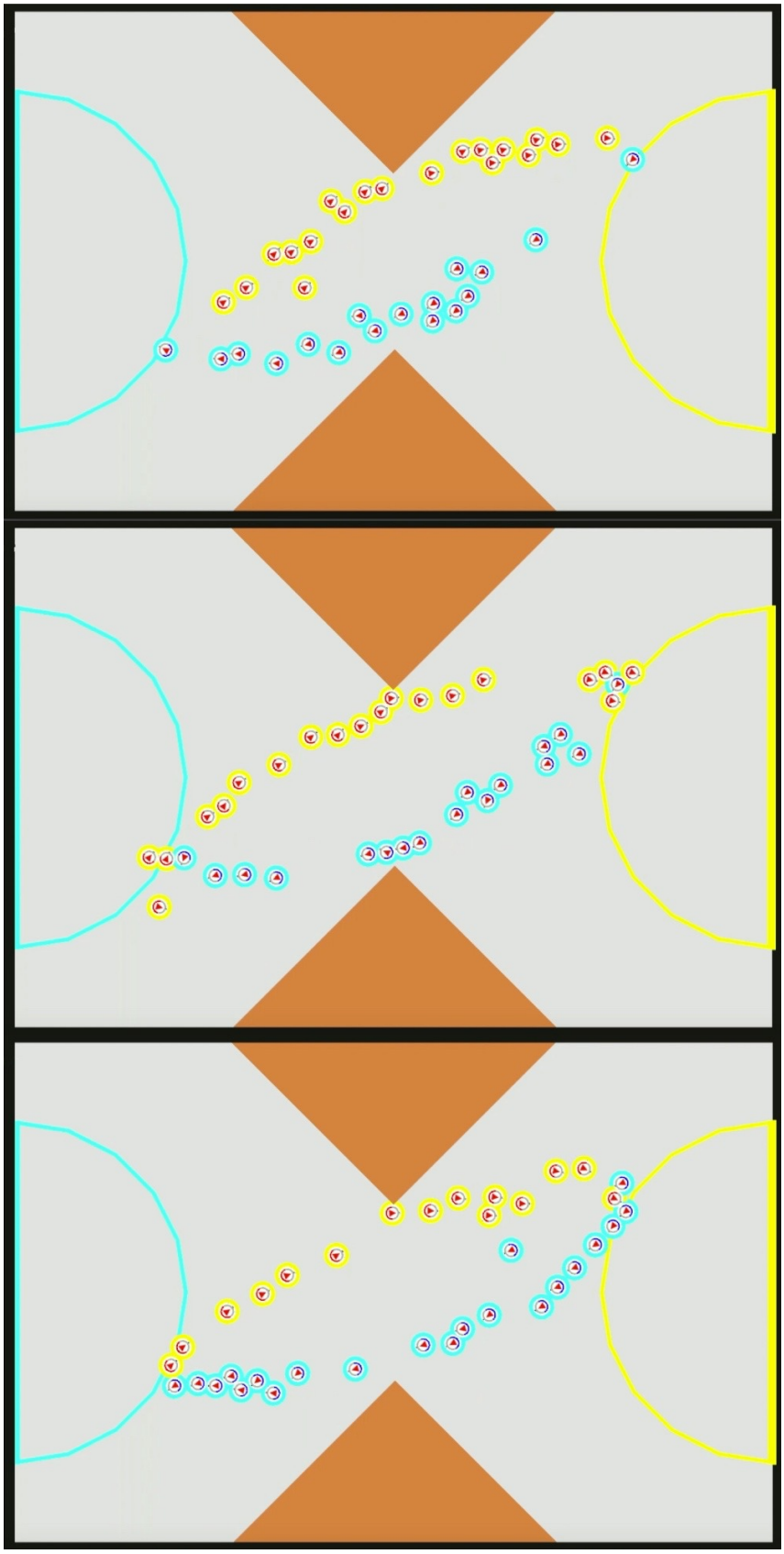


Figure 5.16: Simulation snapshots of the best controller developed by reward setting A.

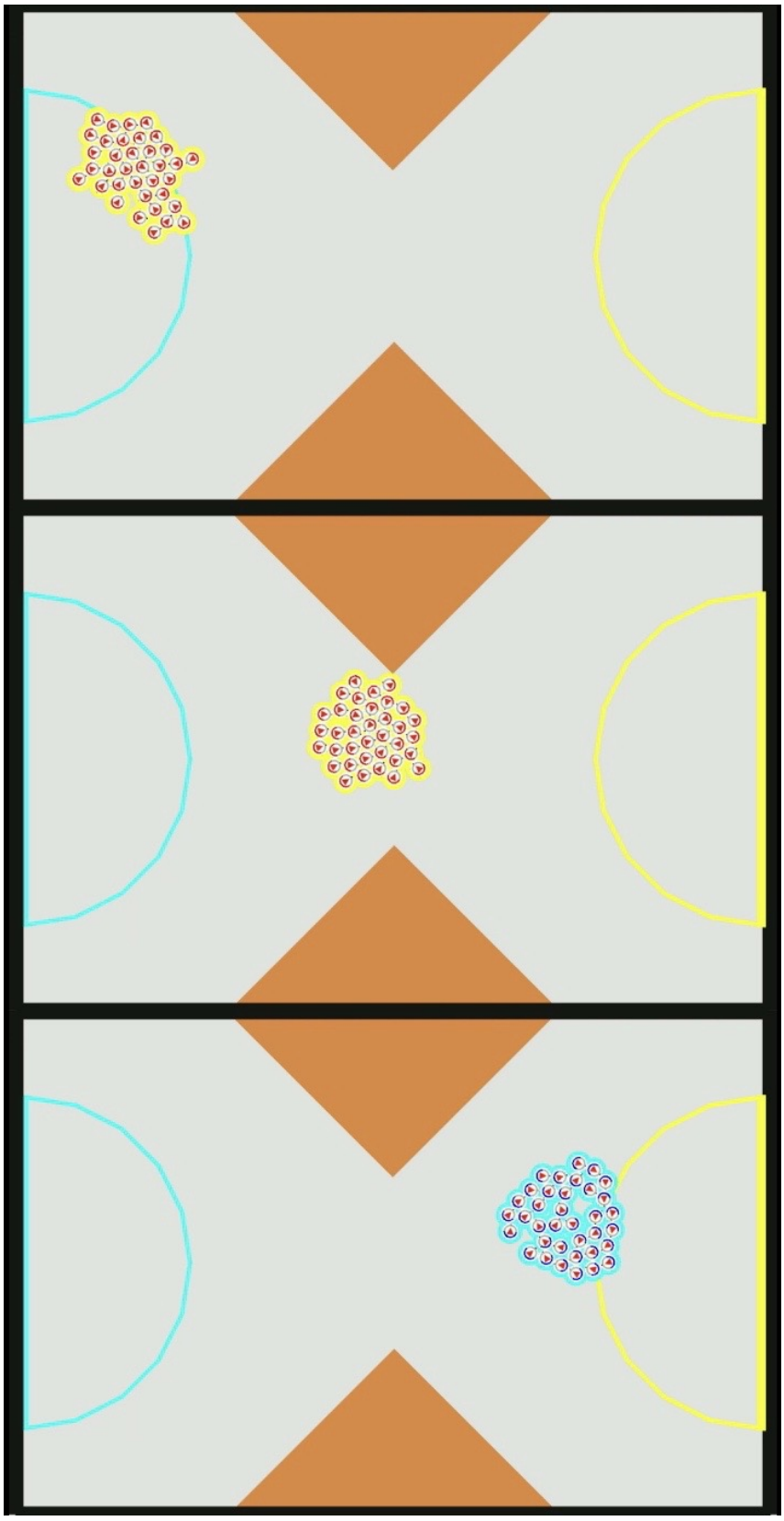


Figure 5.17: Simulation snapshots of the best controller developed by reward setting B.

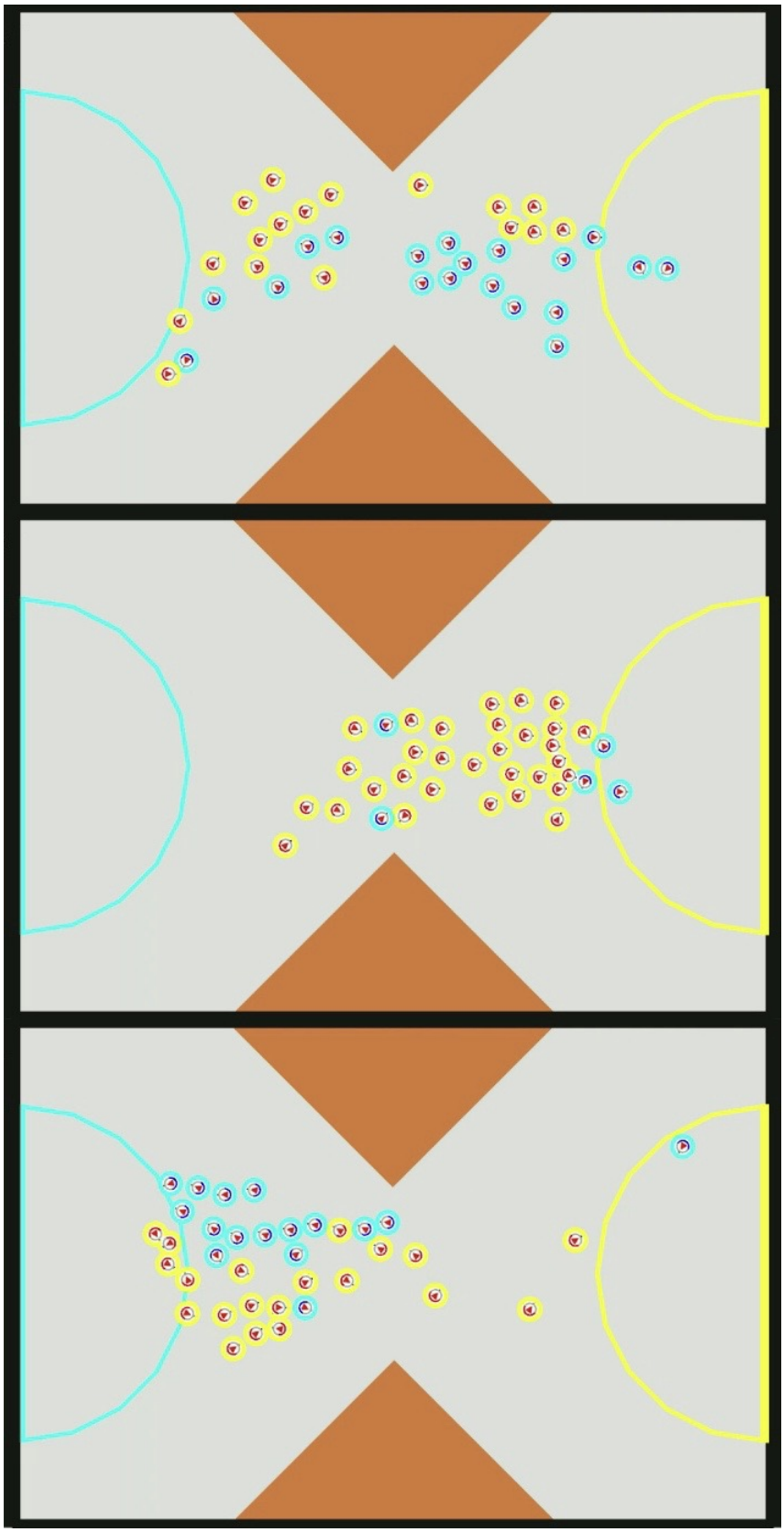


Figure 5.18: Simulation snapshots of the best controller developed by reward setting C.

5.5 Experiment 3: Perturbation-based Methods

A typical question in the field of computer vision is whether the model accurately locates the object in the image or only makes decisions based on the surrounding context. The perturbation-based method attempts to answer this question by quantifying how modifications to the input affect the model's output. In summary, perturbation-based methods are to measure how a model' s output changes when a certain region of the input alters.

Inspired by this question, in this experiment, we slide an occluding gray rectangle across the input image and evaluate the change in the swarm performance, which results in the importance of each occluded region. By analyzing the changes, we can gain insights into which regions of the input are crucial for the model's decision-making process. It can also be used to identify which features the model is focusing on when making predictions and to understand the underlying mechanisms of the model's decision-making process.

5.5.1 Experimental Settings

Same as the previous experiments, we adopted the network structure combining traditional DQN with Deconvnet.

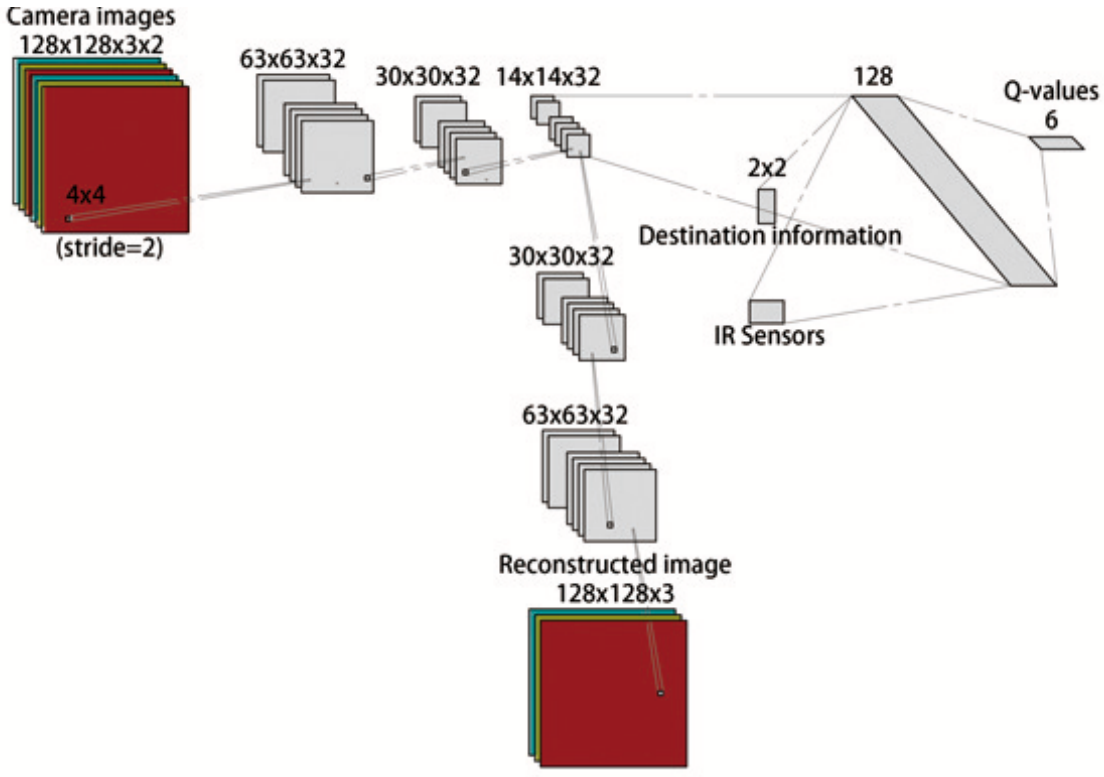


Figure 5.19: Architecture of the deep neural network.

Table 5.8: Reward settings

r_d	r_l	r_f	r_p
5.0	1.0	5.0	- 1.0

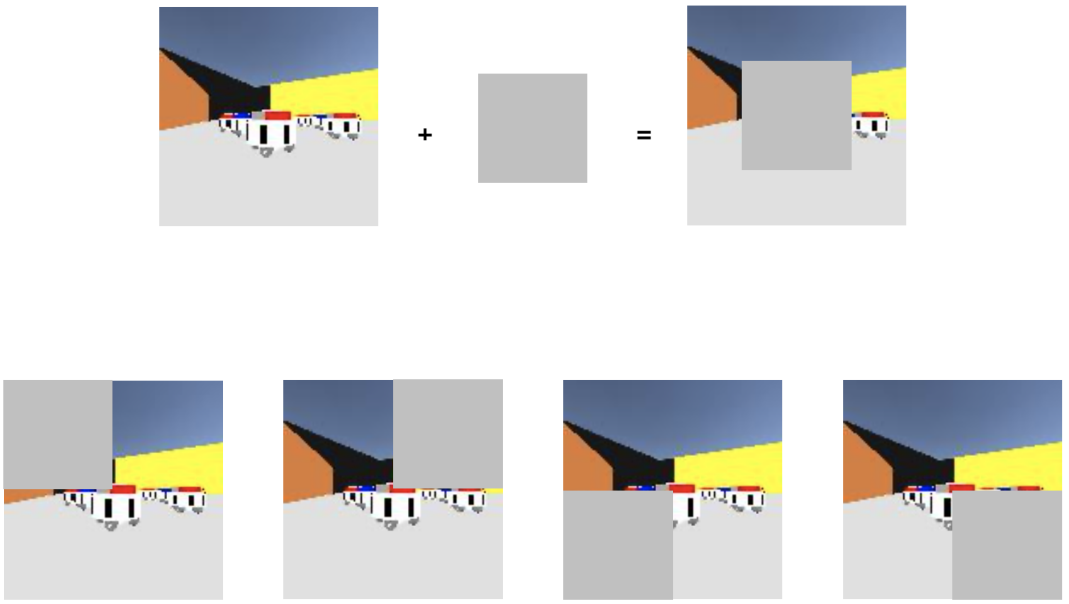


Figure 5.20: Images used for perturbation experiments.

The first reward system mentioned in last section is selected, which encourages robots to focus on observing the corresponding landmark directly. The numerical settings are shown in Table 5.8.

We slide a grey square on the input image to occlude the middle part, upper left part, upper right part, lower left part, and lower right part of the image, respectively. Gray square is 64 $\times$ 64 pixels and can block 25% of the input image (input image is 128 $\times$ 128 pixels). The specific locations are shown in Fig. 5.20.

5.5.2 Results

In order to present the impact of perturbation in different locations, five experiments are performed. We first prepared a well-trained model in the round-trip task. Thirty episodes are tested for each setting. To evaluate the performance of different occlude positions,

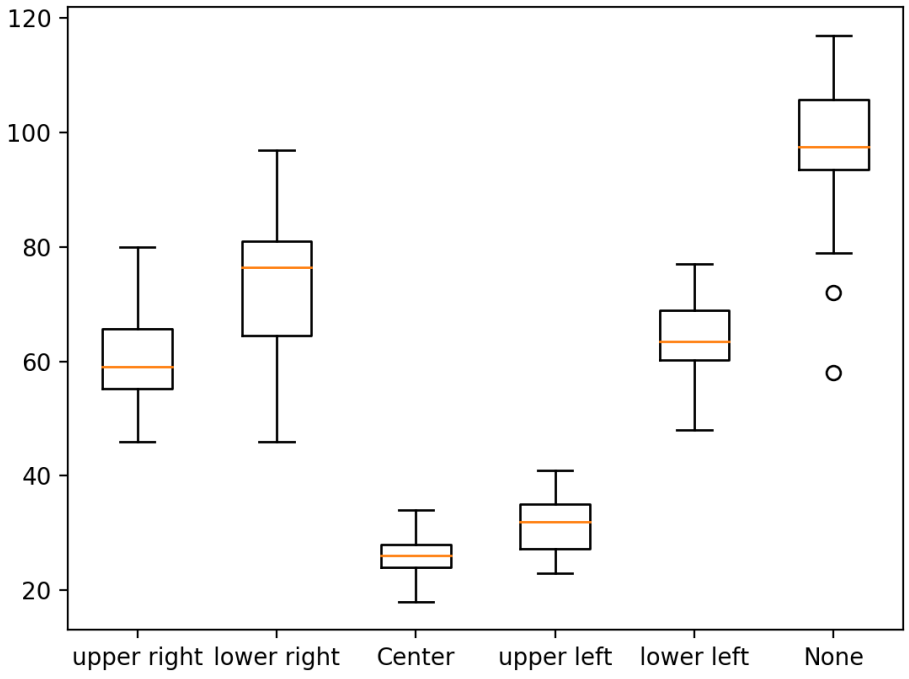


Figure 5.21: Measure of importance for different occlude positions. The lower arrivals indicates that the rigion occluded by the grey square is more important.

we recorded how many times robots arrived at their destinations. Fig. 5.21 shows the number of arrivals. In general, we find that the best performance was obtained without occlusion, where the robot can reach its destination up to 117 times with a median of 97.5 (excluding 2 abnormal values). When the upper right part, lower right part, and lower left part are blocked, the impact is similar. The robot can reach the destination up to 80 times, 97 times, and 77 times respectively, the median are 59, 76.5, and 63.5. However, when the upper left part and center part are blocked, swarm performance will be greatly decreased. The robot only can reach the destination up to 41 and 34 times, with a median of 32 and 26(around 70% decrease).

Table 5.9 shows the comparison of the swarm performance when gray squares occlude at different locations.

The lower arrivals indicate that the area occluded by the grey square is more important. However, in our experiments, the environment is dynamic. Why does occluding different regions of the input image lead to different results? We observed that during the process of training, in order to avoid collisions and higher efficiency, robots gradually move in a clockwise circle or counterclockwise circle.

Table 5.9: Comparison of the swarm performance

Occlusion location	None	Upper left	Upper right	Lower left	Lower right	Center
Max	117.0	41.0	80.0	77.0	97.0	34.0
Min	79.0	23.0	46.0	48.0	46.0	18.0
Median	97.5	32.0	59.0	63.5	76.5	26
Standard deviation	12.56	5.09	7.30	9.35	13.13	3.65

Fig.5.22 shows the snapshot of the swarm behavior of the selected model. Fig. 5.23 shows the images acquired by the robot camera during the test. We observed that in the selected model, most robots follow fellow robots in front of them and move in a counterclockwise circle toward their destinations. Therefore, the destination will always appear on the robot's front left side. It can be clearly seen from Fig. 5.23, landmarks appear in the upper left corner of the input image. Accordingly, we infer that for the selected model, the upper left part and the center part are important regions.

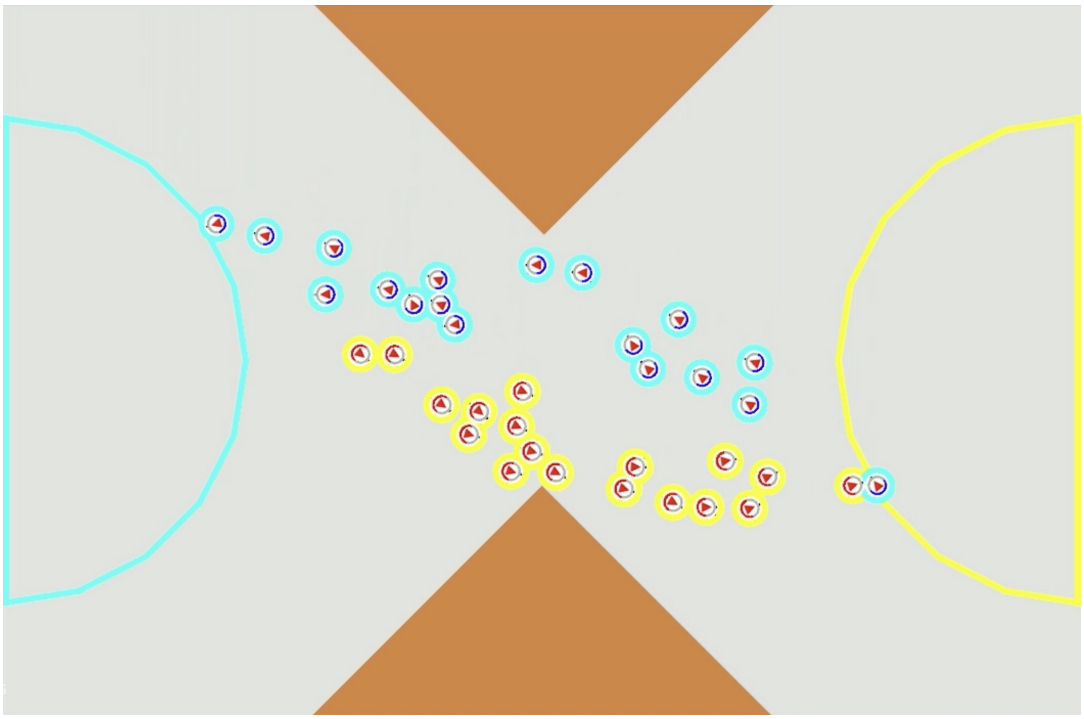
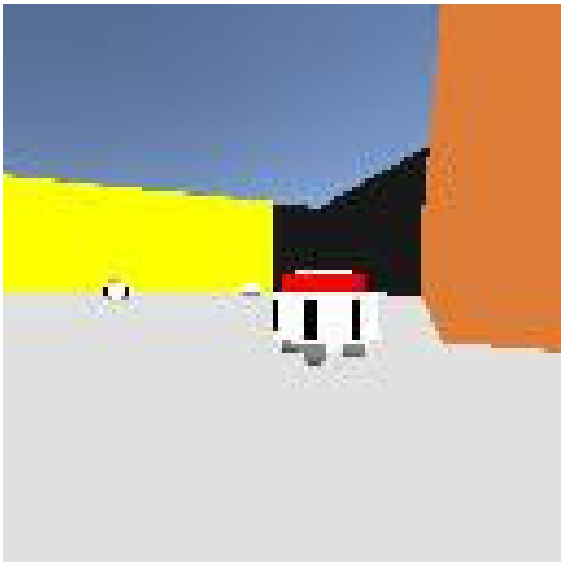
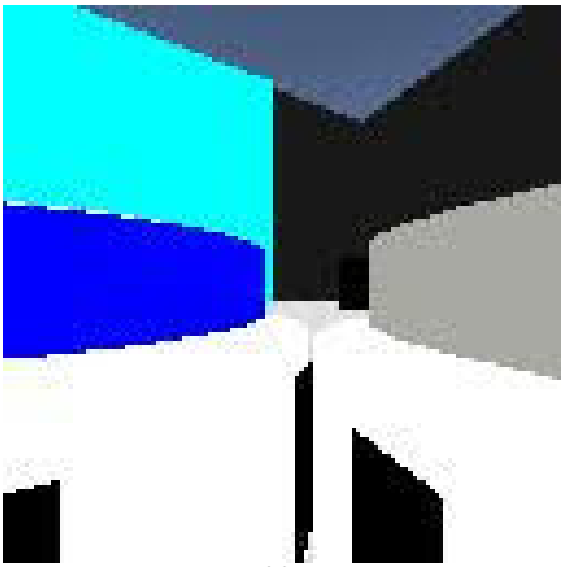


Figure 5.22: Snapshot of the swarm behavior of the selected model.



(a)

(b)



(c)

Figure 5.23: Images acquired by the robot camera during the test.

5.6 Conclusion

In this chapter, an end-to-end control strategy for SRS using a DQN algorithm in a round-trip task was successfully developed. The effectiveness of the proposed method was examined by testing control policies under various experimental settings. The results show that the proposed method is capable of developing control policies using high-dimensional camera raw inputs, and that these policies can be successful under proper reward design.

Moreover, an explainable reinforcement learning approach is also proposed in this chapter. Three experiments are conducted to visualize the policies learned by deep Q-network. The first experiment proposes a network structure with several deconvolutional layers to view the neural network's feature map during various training stages. The second experiment employs a saliency map method Gradient-weighted Class Activation Mapping to determine which state variables the robot attends to during strategy execution. Lastly, the third experiment utilizes a perturbation-based visualization method to evaluate the fault tolerance of the controller. Simulation results show that the proposed method is able to interpret the policies learned by DQN. It is very valuable for increasing understanding of DRL.

Chapter 6

Conclusion

This thesis focuses on optimizing controllers of SRS with DRL. In recent years, DRL has demonstrated great potential in designing controllers to various static environment tasks, such as playing video games. However, it is difficult for traditional DRL to learn effective policies in dynamic environments due to the lack of complete observability and the non-stationarity of the environment. SRS is a field that involves the coordination of multiple robots working in a decentralized manner. In SRSs, each robot only observes a partial view of the environment and other robots are treated as part of the environment. This characteristic results in a highly dynamic environment that poses significant challenges for DRL algorithms. To address this challenge, one of the efficient ways is to integrate with other algorithms, such as curriculum learning, to improve the performance of control algorithms. On the other perspective, enhancing the understanding of DRL through explainable algorithms is also critical. By understanding decision-making processes and policy characteristics, potential issues will be discovered and improved. In this thesis, three contributions are presented to the field of SRS and DRL.

Firstly, this thesis proposed a novel automatic curriculum learning method called Self-Teaching Automatic Curriculum Learning (STACL). When faced with complex tasks, DRL is insufficient for directly training end-to-end controllers in a dynamic environment. Chapter 3 showed how the proposed method is applied to solving this issue. In order to illustrate the effectiveness of STACL, this study uses a collective wall-jumping task, in which the robots have to jump over the high wall collectively and reach the goal as quickly as possible. The proposed algorithm integrates robot training with curriculum scheduling in one neural network. The reward function can calculate the learning rate for different curricula, then select the next subtask to be trained automatically for the next episode. The proposed method can ensure that the neural network remains in an optimal state for learning. The proposed approach is compared with the manual CL, random CL, and a conventional RL algorithm. Simulation resluts demonstrate that the proposed method has the quickest convergence speed since it can automatically schedule lessons

and is unaffected by manual settings. Additionally, we also performed experiments to examine the flexibility of the proposed approach.

Secondly, this thesis presented how DRL is utilized to address a decision-making problem in a multi-autonomous vehicle task. In this task, multi-autonomous vehicles are formulized as a SRS controlled by DRL algorithm. Environmental vehicles, are seen as part of the environment. The positions and actions of environmental vehicles are unpredictable, and their movements may affect the decisions of autonomous vehicles, which makes the environment more dynamic and dangerous. Therefore, it is necessary to equip autonomous vehicles with a security assurance mechanism. In Chapter 4, we utilize time-to-collision (TTC) as the feature representation and propose a TTC-based safety check system. In this chapter, a ramp merging task is used to illustrate the effect. The action output by the DRL controller would be replaced with a safer action chosen by the safety check system when an agent detects a potential collision. Simulation results show that the proposed method can effectively improve the arrival rate and reduce the collision rate, even in the case of dense traffic situations. Furthermore, we also examine the performance of the safety check system with different time thresholds.

Thirdly, this thesis proposed an explainable reinforcement learning approach. The lack of interpretability problem limits the understanding and optimization of the model's decision-making in dynamic environments. Chapter 5 utilizes several visualization approaches to improve the understanding of the control strategy. In this chapter, we applied a deep Q-learning algorithm to develop controllers for a SRS that take raw camera images as input. This chapter is structured around three experiments. The first experiment proposes a network structure with several deconvolutional layers to view the neural network's feature map during various training stages. At each stage of the learning process, this approach promotes a more comprehensive understanding of the underlying control strategy. The second experiment employs a saliency map method Grad-CAM to determine which state variables robots attend to view. Lastly, the third experiment utilizes a perturbation-based visualization method to evaluate the fault tolerance of the controller. Simulation results show that the proposed approach can interpret the control policies.

Overall, the proposed methods can optimizing controllers of SRS with DRL, provide promising solutions for addressing complex tasks in dynamic environments that traditional DRL approaches struggle with.

6.1 Future Work

On the basis of this thesis, one future research direction is investigate the application of SRS in real-world scenarios. Although this thesis explores optimizing controllers of SRS with DRL in dynamic environments, it is still far from real-world applications. SRS has

great potential to solve a wide range of real-world problems, such as search, rescue and environmental monitoring. However, the adoption of SRS for these problems has been limited due to several challenges, including the reality gap. Therefore, in the future, it is crucial to improve the performance of SRS in more complex and realistic environments.

One way to achieve this is through the use of data augmentation [96]. Data augmentation is a technique used in machine learning to increase the diversity and quantity of available training data by applying various transformations and manipulations to the existing data. It can be used in SRS to process existing data, increasing the diversity and quantity of the data samples, and improving the model's generalization ability. By applying operations such as rotation, translation, scaling, and noise addition to robot sensor data, different environments can be simulated, thereby providing robots with stronger adaptability and better performance.

Another effective approach is through the use of joint training [97]. Joint training, as a multi-task learning technique, can be employed in SRS field. Joint training refers to the joint training of multiple models so that they can jointly handle more complex tasks or environments. By integrating data from different sources, such as sensors and images, joint training can enhance the perception abilities of the robots, making them more adaptable to diverse surroundings. Additionally, the curriculum learning method STACL proposed in this thesis may be combined with joint training. For instance, STACL can be used to automatically introduce various collective tasks. Joint training can enable robots to perform tasks in multiple different environments simultaneously, which can imporve their generalization performance.

In conclusion, these approaches have the potential to bridge the reality gap and make it possible to apply SRS to real-world problems.

References

[1] Eric Bonabeau, Directeur de Recherches Du Fnrs Marco, Marco Dorigo, Guy Théraulaz, Guy Theraulaz, et al. *Swarm intelligence: from natural to artificial systems*. Number 1. Oxford university press, 1999.

[2] Erol Şahin. Swarm robotics: from sources of inspiration to domains of application. In *Swarm Robotics*, volume 3342 of *Lecture Notes in Computer Science*, pages 10–20. Springer, 2005.

[3] James Kennedy. Swarm intelligence. In *Handbook of nature-inspired and innovative computing*, pages 187–219. Springer, 2006.

[4] Vito Trianni, Stefano Nolfi, and Marco Dorigo. Evolution, self-organization and swarm robotics. In *Swarm Intelligence*, pages 163–191. Springer, 2008.

[5] Marco Dorigo, Mauro Birattari, and Manuele Brambilla. Swarm robotics. *Scholarpedia*, 9(1):1463, 2014.

[6] Manuele Brambilla, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo. Swarm robotics: A review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41, 2013.

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[8] Heiko Hamann, Yara Khaluf, Jean Botev, Mohammad Divband Soorati, Eliseo Ferrante, Oliver Kosak, Jean-Marc Montanier, Sanaz Mostaghim, Richard Redpath, Jon Timmis, Frank Veenstra, Mostafa Wahby, and Aleš Zamuda. Hybrid societies: Challenges and perspectives in the design of collective behavior in self-organizing systems. *Frontiers in Robotics and AI*, 5:14, 2018.

[9] KN McGuire, Christophe De Wagter, Karl Tuyls, HJ Kappen, and Guido CHE de Croon. Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment. *Science Robotics*, 4(35):eaaw9710, 2019.

[10] Melanie Schranz, Martina Umlauft, Micha Sende, and Wilfried Elmenreich. Swarm robotic behaviors and current applications. *Frontiers in Robotics and AI*, 7:36, 2020.

[11] Luneque Del Rio Silva Junior and Nadia Nedjah. Efficient strategy for collective navigation control in swarm robotics. In *Swarm Intelligence Based Optimization*, volume 639 of *Studies in Computational Intelligence*, pages 817–826. Springer, 2016.

[12] Wenguo Liu and Alan FT Winfield. Modeling and optimization of adaptive foraging in swarm robotic systems. *The International Journal of Robotics Research*, 29(14):1743–1760, 2010.

[13] Gianpiero Francesca and Mauro Birattari. Automatic design of robot swarms: achievements and challenges. *Frontiers in Robotics and AI*, 3:29, 2016.

[14] Stefano Nolfi, Dario Floreano, and Director Dario Floreano. *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. MIT press, 2000.

[15] Maja J Matarić. Reinforcement learning in the multi-robot domain. In *Robot colonies*, pages 73–83. Springer, 1997.

[16] Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

[17] Dario Floreano, Peter Dürr, and Claudio Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.

[18] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[19] Martin Friedmann. *Simulation of autonomous robot teams with adaptable levels of abstraction*. PhD thesis, Technische Universität, 2010.

[20] Adam T Hayes, Alcherio Martinoli, and Rodney M Goodman. Swarm robotic odor localization: Off-line optimization and validation with real robots. *Robotica*, 21(4):427–441, 2003.

[21] Priya Bannur, Purvi Gujarathi, Karthik Jain, and Anand J Kulkarni. Application of swarm robotic system in a dynamic environment using cohort intelligence. *Soft Computing Letters*, 2:100006, 2020.

[22] Giovanni Beltrame, Ettore Merlo, Jacopo Panerati, and Carlo Pinciroli. Engineering safety in swarm robotics. In *Proceedings of the 1st International Workshop on Robotics Software Engineering*, pages 36–39, 2018.

[23] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[24] Shae T Hart, Jake Kamenetsky, and Christopher A Kitts. Dynamic elliptical shaping control for swarm robots. *IEEE Access*, 11:17454–17470, 2023.

[25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[26] Maximilian Hüttenrauch, Sosic Adrian, Gerhard Neumann, et al. Deep reinforcement learning for swarm systems. *Journal of Machine Learning Research*, 20(54):1–31, 2019.

[27] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[28] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6. Ieee, 2017.

[29] Christos H Papadimitriou and John N Tsitsiklis. The complexity of markov decision processes. *Mathematics of operations research*, 12(3):441–450, 1987.

[30] Peter Marbach and John N Tsitsiklis. Simulation-based optimization of markov reward processes. *IEEE Transactions on Automatic Control*, 46(2):191–209, 2001.

[31] Richard S Sutton. Reinforcement learning: Past, present and future. In *Asia-Pacific Conference on Simulated Evolution and Learning*, pages 195–197. Springer, 1998.

[32] Kenji Doya, Kazuyuki Samejima, Ken-ichi Katagiri, and Mitsuo Kawato. Multiple model-based reinforcement learning. *Neural computation*, 14(6):1347–1369, 2002.

[33] Alexander L Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L Littman. Pac model-free reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 881–888, 2006.

[34] WQ Yang and TA York. New ac-based capacitance tomography system. *IEE Proceedings-Science, Measurement and Technology*, 146(1):47–53, 1999.

[35] Maximilian Hüttenrauch, Adrian Sosic, and Gerhard Neumann. Guided deep reinforcement learning for swarm systems. *CoRR*, abs/1709.06011, 2017.

[36] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[37] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[38] Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015.

[39] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[40] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR, 2014.

[41] Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron Courville, and Yoshua Bengio. An actor-critic algorithm for sequence prediction. *arXiv preprint arXiv:1607.07086*, 2016.

[42] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

[43] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[44] Petru Soviany, Radu Tudor Ionescu, Paolo Rota, and Nicu Sebe. Curriculum learning: A survey. *CoRR*, abs/2101.10382, 2021.

[45] Alessandro Lazaric and Marcello Restelli. Transfer from multiple mdps. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

[46] Anestis Fachantidis, Ioannis Partalas, Grigorios Tsoumakas, and Ioannis Vlahavas. Transferring task models in reinforcement learning agents. *Neurocomputing*, 107:23–32, 2013. Timely Neural Networks Applications in Engineering.

[47] Matthew E. Taylor, Peter Stone, and Yaxin Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8(1):2125–2167, 2007.

[48] Haitham Bou-Ammar, Eric Eaton, José-Marcio Luna, and Paul Ruvolo. Autonomous cross-domain knowledge transfer in lifelong policy gradient reinforcement learning. In *International Joint Conference on Artificial Intelligence*, 2015.

[49] Massimo Caccia, Jonas Mueller, Taesup Kim, Laurent Charlin, and Rasool Fakoor. Task-agnostic continual reinforcement learning: In praise of a simple baseline. *arXiv preprint arXiv:2205.14495*, 2022.

[50] Sanmit Narvekar, Jivko Sinapov, Matteo Leonetti, and Peter Stone. Source task creation for curriculum learning. In *Proceedings of the 2016 international conference on autonomous agents & multiagent systems*, pages 566–574, 2016.

[51] Yong Liu, Yujing Hu, Yang Gao, Yingfeng Chen, and Changjie Fan. Value function transfer for deep multi-agent reinforcement learning based on n-step returns. In *IJCAI*, pages 457–463. Macao, 2019.

[52] Rémy Portelas, Cédric Colas, Lilian Weng, Katja Hofmann, and Pierre-Yves Oudeyer. Automatic curriculum learning for deep RL: A short survey. *CoRR*, abs/2003.04664, 2020.

[53] Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher-student curriculum learning. *CoRR*, abs/1707.00183, 2017.

[54] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving rubik's cube with a robot hand. *CoRR*, abs/1910.07113, 2019.

[55] Bowen Baker, Ingmar Kanitscheider, Todor M. Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. *CoRR*, abs/1909.07528, 2019.

[56] Rui Zhao and Volker Tresp. Curiosity-driven experience prioritization via density estimation. *CoRR*, abs/1902.08039, 2019.

[57] Felix Grün, Christian Rupprecht, Nassir Navab, and Federico Tombari. A taxonomy and library for visualizing learned features in convolutional neural networks. *ArXiv*, abs/1606.07757, 2016.

[58] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[59] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2921–2929, 2016.

[60] Aravindh Mahendran and Andrea Vedaldi. Visualizing deep convolutional neural networks using natural pre-images. *International Journal of Computer Vision*, 120(3):233–255, 2016.

[61] Jianming Zhang, Sarah Adel Bargal, Zhe Lin, Jonathan Brandt, Xiaohui Shen, and Stan Sclaroff. Top-down neural attention by excitation backprop. *International Journal of Computer Vision*, 126(10):1084–1102, 2018.

[62] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[63] Aditya Chattopadhay, Anirban Sarkar, Prantik Howlader, and Vineeth N Balasubramanian. Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 839–847. IEEE, 2018.

[64] Luisa M Zintgraf, Taco S Cohen, Tameem Adel, and Max Welling. Visualizing deep neural network decisions: Prediction difference analysis. *arXiv preprint arXiv:1702.04595*, 2017.

[65] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3319–3328. JMLR. org, 2017.

[66] Julius Adebayo, Justin Gilmer, Ian Goodfellow, and Been Kim. Local explanation methods for deep neural networks lack sensitivity to parameter values. *arXiv preprint arXiv:1810.03307*, 2018.

[67] Pieter-Jan Kindermans, Sara Hooker, Julius Adebayo, Maximilian Alber, Kristof T. Schütt, Sven Dähne, Dumitru Erhan, and Been Kim. The (un)reliability of saliency methods, 2017.

[68] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[69] Piotr Dabkowski and Yarin Gal. Real time image saliency for black box classifiers. In *Advances in Neural Information Processing Systems*, pages 6967–6976, 2017.

[70] Ruth C Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3429–3437, 2017.

[71] Jonathan L Long, Ning Zhang, and Trevor Darrell. Do convnets learn correspondence? In *Advances in neural information processing systems*, pages 1601–1609, 2014.

[72] Alexey Dosovitskiy and Thomas Brox. Inverting convolutional networks with convolutional networks. *arXiv preprint arXiv:1506.02753*, 4, 2015.

[73] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5188–5196, 2015.

[74] Tom Zahavy, Nir Ben-Zrihem, and Shie Mannor. Graying the black box: Understanding dqns. In *International conference on machine learning*, pages 1899–1908. PMLR, 2016.

[75] Sam Greydanus, Anurag Koul, Jonathan Dodge, and Alan Fern. Visualizing and understanding atari agents. *arXiv preprint arXiv:1711.00138*, 2017.

[76] Diansheng Chen, Kewei Chen, Ziqiang Zhang, and Benguang Zhang. Mechanism of locust air posture adjustment. *Journal of Bionic Engineering*, 12(3):418–431, 2015.

[77] Minkyun Noh, Seung-Won Kim, Sungmin An, Je-Sung Koh, and Kyu-Jin Cho. Flea-inspired catapult mechanism for miniature jumping robots. *IEEE transactions on robotics*, 28(5):1007–1018, 2012.

[78] John W Romanishin, Kyle Gilpin, and Daniela Rus. M-blocks: Momentum-driven, magnetic modular robots. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4288–4295. IEEE, 2013.

[79] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

[80] Riccardo Scarinci and Benjamin Heydecker. Control concepts for facilitating motorway on-ramp merging using intelligent vehicles. *Transport reviews*, 34(6):775–797, 2014.

[81] Xinpeng Wang, Ding Zhao, Huei Peng, and David J LeBlanc. Analysis of unprotected intersection left-turn conflicts based on naturalistic driving data. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 218–223. IEEE, 2017.

[82] Xuedong Yan and Essam Radwan. Effect of restricted sight distances on driver behaviors during unprotected left-turn phase at signalized intersections. *Transportation research part F: traffic psychology and behaviour*, 10(4):330–344, 2007.

[83] Saad Yousif, Zaid Nassrullah, and Sarah H Norgate. Narrow lanes and their effect on drivers' behaviour at motorway roadworks. *Transportation research part F: traffic psychology and behaviour*, 47:86–100, 2017.

[84] Alexandra Kondyli, David K Hale, Mohamadamin Asgharzadeh, Bastian Schroeder, Anxi Jia, and Joe Bared. Evaluating the operational effect of narrow lanes and shoulders for the highway capacity manual. *Transportation research record*, 2673(10):558–570, 2019.

[85] Brian Paden, Michal Cáp, Sze Zheng Yong, Dmitry S. Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *CoRR*, abs/1604.07446, 2016.

[86] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical Review E*, 62(2):1805–1824, aug 2000.

[87] Arne Kesting, Martin Treiber, and Dirk Helbing. General lane-changing model mobil for car-following models. *Transportation Research Record*, 1999(1):86–94, 2007.

[88] Wenshuo Wang, Letian Wang, Chengyuan Zhang, Changliu Liu, and Lijun Sun. Social interactions for autonomous driving: A review and perspective, 2022.

[89] Jonas Jansson. *Collision Avoidance Theory: With application to automotive collision mitigation*. PhD thesis, Linköping University Electronic Press, 2005.

[90] Samyeul Noh and Woo-Yong Han. Collision avoidance in on-road environment for autonomous driving. In *2014 14th International Conference on Control, Automation and Systems (ICCAS 2014)*, pages 884–889. IEEE, 2014.

[91] Ali Baheri, Subramanya Nageshrao, H. Eric Tseng, Ilya Kolmanovsky, Anouck Girard, and Dimitar Filev. Deep reinforcement learning with enhanced safety for autonomous highway driving. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1550–1555, 2020.

[92] Subramanya Nageshrao, H. Eric Tseng, and Dimitar Filev. Autonomous highway driving using deep reinforcement learning. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 2326–2331, 2019.

[93] Richard Van Der Horst and Jeroen Hogema. Time-to-collision and collision avoidance systems. 1993.

[94] John C Hayward. Near miss determination through use of a scale of danger. 1972.

[95] Edouard Leurent. An environment for autonomous driving decision-making, 2018.

[96] Guozheng Ma, Zhen Wang, Zhecheng Yuan, Xueqian Wang, Bo Yuan, and Dacheng Tao. A comprehensive survey of data augmentation in visual reinforcement learning. *arXiv preprint arXiv:2210.04561*, 2022.

[97] Jonathan J Tompson, Arjun Jain, Yann LeCun, and Christoph Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. *Advances in neural information processing systems*, 27, 2014.

Appendix A

Publications Presented in the Thesis

This appendix provides a list of publications that are presented in the thesis. This appendix only shows a list of work published in academic journals and international conferences. The full list of publications is in Appendix B.

Chapter 3

- Xiaotong Nie, Yupeng Liang, Ziyao Han and Kazuhiro Ohkura, "Generating collective wall-jumping behavior for a robotic swarm with self-teaching automatic curriculum learning", Artificial Life and Robotics, Vol. 28, No. 1, pp. 67–75 (2023)

Chapter 4

- Xiaotong Nie, Yupeng Liang, and Kazuhiro Ohkura, "Autonomous highway driving using reinforcement learning with safety check system based on time-to-collision", Artificial Life and Robotics, Vol. 28, No. 1, pp. 158—165 (2023)

- Xiaotong Nie, Yupeng Liang, and Kazuhiro Ohkura, "Autonomous Highway Driving Using Reinforcement Learning with Safety Check System based on Time-to-Collision", Proceedings of the Joint Symposium of the 28th International Symposium on Artificial Life and Robotics, the 8th International Symposium on BioComplexity, and the 6th International Symposium on Swarm Behavior and Bio-Inspired Robotics, pp. 619–625 (2023)

Chapter 5

- Yufei Wei, Xiaotong Nie, Motoaki Hiraga, Kazuhiro Ohkura, and Zlatan Car, "Developing End-to-End Control Policies for Robotic Swarms Using Deep Q-Learning", Journal of Advanced Computational Intelligence and Intelligent Informatics, Vol. 23, No. 5, pp. 920–927 (2019)
- Xiaotong Nie, Motoaki Hiraga, and Kazuhiro Ohkura, "Visualizing Deep Q-Learning to Understanding Behavior of Swarm Robotic System", Proceedings of the 23rd Asia Pacific Symposium on Intelligent and Evolutionary Systems, pp. 118–129 (2019)
- Xiaotong Nie, Kepeng Zhang, and Kazuhiro Ohkura, "Visual Policy Rationalizations Using Grad-CAM in a Robotic Swarm Environment", 第64回システム制御情報学会研究発表講演会講演論文集, GS21-5, pp.920–924 (2020)

Appendix B

List of Publications

This appendix provides a list of publications that are presented in the thesis. This appendix only shows a list of work published in academic journals and international conferences. The full list of publications is in Appendix B.

Journal Publications

- Xiaotong Nie, Yupeng Liang, and Kazuhiro Ohkura, "Autonomous highway driving using reinforcement learning with safety check system based on time-to-collision", Artificial Life and Robotics, Vol. 28, No. 1, pp. 158—165 (2023)
- Xiaotong Nie, Yupeng Liang, Ziyao Han and Kazuhiro Ohkura, "Generating collective wall-jumping behavior for a robotic swarm with self-teaching automatic curriculum learning", Artificial Life and Robotics, Vol. 28, No. 1, pp. 67–75 (2023)
- Yupeng Liang, Ziyao Han, Xiaotong Nie and Kazuhiro Ohkura , "Improving generative adversarial network with multiple generators by evolutionary algorithms", Artificial Life and Robotics, Vol. 27, No. 4, pp. 761–769 (2022)
- Yufei Wei, Xiaotong Nie, Motoaki Hiraga, Kazuhiro Ohkura, and Zlatan Car, "Developing End-to-End Control Policies for Robotic Swarms Using Deep Q-Learning", Journal of Advanced Computational Intelligence and Intelligent Informatics, Vol. 23, No. 5, pp. 920–927 (2019)

International Conferences

- Xiaotong Nie, Yupeng Liang, and Kazuhiro Ohkura, "Autonomous Highway Driving Using Reinforcement Learning with Safety Check System based on Time-to-Collision", Proceedings of the Joint Symposium of the 28th International Sympo-

sium on Artificial Life and Robotics, the 8th International Symposium on BioComplexity, and the 6th International Symposium on Swarm Behavior and Bio-Inspired Robotics, pp. 619–625 (2023)

- Xiaotong Nie, Motoaki Hiraga, and Kazuhiro Ohkura, "Visualizing Deep Q-Learning to Understanding Behavior of Swarm Robotic System", Proceedings of the 23rd Asia Pacific Symposium on Intelligent and Evolutionary Systems, pp. 118–129 (2019)

Domestic Conferences

- Xiaotong Nie, Kepeng Zhang, and Kazuhiro Ohkura, "Visual Policy Rationalizations Using Grad-CAM in a Robotic Swarm Environment", 第64回システム制御情報学会研究発表講演会講演論文集, GS21-5, pp.920–924 (2020)

- Kepeng Zhang, Yupeng Liang, Xiaotong Nie, and Kazuhiro Ohkura, "Source Localization by a Swarm Robotics Systems in an Unknown Environment", 第 64 回システム制御情報学会研究発表講演会講演論文集, GS17-6, pp.793–795 (2020)

- Xiaotong Nie, Yufei Wei, Kazuhiro Ohkura, "Evaluating Optimizers of Deep Reinforcement Learning on Swarm Robotic Systems", 第 27 回計測自動制御学会中国支部学術講演会論文集, 1E-5, pp. 63–64 (2018)

- Shunichi Kataoka, Xiaotong Nie, Yufei Wei, 大倉和博, "深層強化学習を適用したスワームロホ゛ティクスシステムによる協調搬送行動の生成", 第62回システム制御情報学会研究発表講演会講演論文集, 326-1, 7pages (2018)

- Xiaotong Nie, Shunichi Kataoka, and Kazuhiro Ohkura, "Generating Chain Formation For A Robotic Swarm using Deep Reinforcement Learning", 第27回インテリシ゛ェント･システム･シンホ゜シ゛ウム講演原稿集, pp.149–153 (2017)

Acknowledgements

During the preparation of the thesis, I received a lot of invaluable help from many people. Their comments and advice contribute to the accomplishment of this work.

First and foremost, I would like to thank my supervisor, Prof. Kazuhiro Ohkura for his patient supervison and constant support during my academic life at Hiroshima University. Also, I would like to thank the Ph.D. thesis committee members, Prof. Nobutaka Wada, Prof. Ryo Kikuuwe, and Prof. Yoshiyuki Matsumura, for revising the thesis and providing insightful comments.

I would also like to thank my colleagues in Machine Intelligence and Systems A Laboratory (formerly Manufacturing Systems A Laboratory) for much help and support.

This work was partially supported by the Initiative for Realizing Diversity in the Research Environment (Specific Correspondence Type).

Finally, great thanks to my parents, my family, and my friends for their unconditional love, understanding, and encouragement.

2023. 07
Xiaotong Nie