applied sciences

MDPI

Article

Hybrid Centralized Training and Decentralized Execution Reinforcement Learning in Multi-Agent Path-Finding Simulations

Hua-Ching Chen [1], Shih-An Li [2], Tsung-Han Chang [2], Hsuan-Ming Feng [3,*] and Yun-Chien Chen [2]

[1] School of Information Engineering, Xiamen Ocean Vocational College, Xiamen 361100, China; galaxy.km@gmail.com
[2] Department of Electrical and Computer Engineering, Tamkang University, New Taipei City 10650, Taiwan; lish-yhan@gms.tku.edu.tw (S.-A.L.); zxc455233@gmail.com (T.-H.C.); ycpss91255@gmail.com (Y.-C.C.)
[3] Department of Computer Science and Information Engineering, National Quemoy University, Kinmen 89250, Taiwan
* Correspondence: hmfenghmfeng@gmail.com

Abstract: In this paper, we propose a hybrid centralized training and decentralized execution neural network architecture with deep reinforcement learning (DRL) to complete the multi-agent path-finding simulation. In the training of physical robots, collisions and other unintended accidents are very likely to occur in multi-agent cases, so it is required to train the networks within a deep deterministic policy gradient for the virtual environment of the simulator. The simple particle multi-agent simulator designed by OpenAI (Sacramento, CA, USA) for training platforms can easily obtain the state information of the environment. The overall system of the training cycle is designed with a self-designed reward function and is completed through a progressive learning approach from a simple to a complex environment. Finally, we carried out and presented the experiments of multi-agent path-finding simulations. The proposed methodology is better than the multi-agent model-based policy optimization (MAMBPO) and model-free multi-agent soft actor–critic models.

Keywords: deep reinforcement learning; multi-agent path-finding; robotics

Citation: Chen, H.-C.; Li, S.-A.; Chang, T.-H.; Feng, H.-M.; Chen, Y.-C. Hybrid Centralized Training and Decentralized Execution Reinforcement Learning in Multi-Agent Path-Finding Simulations. *Appl. Sci.* **2024**, *14*, 3960. https://doi.org/10.3390/app14103960

Academic Editor: Seokwon Yeom

Received: 26 March 2024
Revised: 28 April 2024
Accepted: 2 May 2024
Published: 7 May 2024

1. Introduction

Reinforcement learning (RL) is a process whereby the agent decides what action to take based on what it sees as the state of the environment. Then, it generates the next moment's state of the environment and the corresponding reward from the environment that provided the basis for the previous action. Thus, the state of the environment obtained by the agent is a very important issue of the reinforcement learning process. DRL mainly uses neural networks to replace the traditional Q-table. When applied to complex problems, Q-tables can be too large to represent or exhaust. After the introduction of the Deep Q-Learning Network (DQN) in 2013, an improved version of the algorithm was proposed in 2015 [1]. One of the improvements focuses on the original version's course of the autonomous navigation of robots [2]. A DQN lets robots operate more efficiently and safely in related outdoor environments; however, one problem of the DQN is that the distribution of states varies considerably, making it difficult to converge the network. Therefore, the target Q-network is added to the learning process. During the learning process, the current Q-network is periodically copied to the target Q-network for sequestration. The development of the Double Deep Q-Learning Network (DDQN) [3] in 2016 adjusted the original Q-value estimation for deep Q-networks. Originally, the value was estimated directly using the Q-network once; however, in the DDQN, the Q-value estimation is conducted using two separate Q-networks preventing the overestimation of Q.

Currently, DRL can be divided into value-based and policy-based approaches. The value-based DRL approach is best known as the DQN. It evaluates the value of each action through a value function. Meanwhile, the policy-based DRL method is the earliest

reinforcement learning algorithm, and it functions by selecting the best action for the current state. Intensive learning algorithms, such as the actor–critic network (ACN), use the advantages of both approaches [4]. The ACN consists of two networks, the actor network and the critic network. The actor network is a policy-based reinforcement learning network that decides actions based on the current state. Meanwhile, the critic network evaluates the value of the actions taken by the actor network. Park et al. [5] used soft actor–critic to complete the high dimensionality and continuous action in robotic problems. Automated guided vehicles (AGVs) are directed by the soft actor–critic (SAC) RL algorithm, which uses a type of a sum-tree prioritized experience replay strategy for autonomous navigation applications [6].

A multi-agent DRL machine involves multiple agents that interact with the environment at the same time. However, the actions of each agent affect the state observed by other agents. Many researchers have used multi-agent DRL to accomplish tasks by a sequence-to-sequence multi-agent deep deterministic policy gradient (SMADDPG) algorithm [7]. One study found that the sum-of-costs optimal solution for the classical multi-agent path-finding (MAPF) problem is NP-hard [8]. Many optimal classical MAPF solvers aim to find collision-free paths with minimal cost by searching the joint configuration space or collision resolution space. For example, Sartoretti et al. [9] published PRIMAL, a multi-agent path-finding method based on reinforcement learning and imitation learning (IL). At its core, it uses Asynchronous Advantage Actor–Critic (A3C) [10] as a learning framework for reinforcement learning. Recently, many studies on multi-agent path-finding based on DRL have referred to the success of PRIMAL, such as the GLAS proposed by Rivière et al. [11]. The paper used information from area observations as input to the neural network. However, GLAS goes one step further by incorporating a dynamics model into the system architecture. It enables more sophisticated multi-agent path planning for small quadcopters. In the G2RL proposed by Binyu Wang [12], the same concept of regional observation as PRIMAL is used as well as the concept of short-term and long-term memory. There are also sequential inputs that use the same concept as GLAS. G2RL has implemented and analyzed algorithms for complex environments. The results proved that G2RL can maintain good results in different environments.

The A* algorithm [13] has excellent path planning ability in one robot and can find the shortest path in a very short time. However, the A* algorithm is still limited in terms of path planning for multiple robots in the same field. It contains overlapping paths that may cause collisions, and the amount of computation may be too large. Several studies modified A* for multi-robot applications, including the windowed hierarchical cooperative A* (WHCA*) proposed by Silver [14]. Various methods have also been developed to compute the optimal solution for MAPF problems including subdimensional expansion [15], compilation-based solver [16], and integer programming-based methods [17] and Conflict-Based Search (CBS) [18,19]. However, they are associated with a long computation time [20], making it one of the difficulties in a multi-agent learning engine. In this study, A* is proposed to explore the best local path planning as an expert's guide in the design reward function.

There are three main architectures for multi-agent learning, namely, fully decentralized, fully centralized, and centralized training and decentralized execution [21]. Due to the powerful agent-centered, multi-agent, and support of actor–critic methods, multi-agent learning could generate fair credit assignment and better generalizability. It also allows information to be distributed evenly across multiple agents [22]. Fully decentralized, multi-agent reinforcement learning is an environment where each agent is individualized, and observed states and actions are not shared between agents. The core architecture of reinforcement learning is that the agent obtains the state of the environment after executing an action. Even when only one agent is interacting with the environment, no problem arises, because the environment is only affected by the actions of the agent. However, with multiple agents operating in the same environment, the actions performed by each

agent affect the state of the other agent. A fully decentralized structure would therefore be difficult to converge.

In a fully centralized architecture, a central neural network determines the actions to be performed by all agents and waits for all agents to finish running before collecting the status and rewards from all agents. It ensures that the neural network is aware of the state of the environment, avoiding the problems of a completely decentralized framework. However, since it needs to wait for all the agents to finish their actions before data collection, a slowdown problem could occur. Therefore, a centralized learning with decentralized execution (CLDE) framework is selected to form multi-agent model-based policy optimization (MAMBPO) for better sample efficiency than the model-free multi-agent soft actor–critic [23,24]. Centralized training coupled with decentralized execution combines the advantages of both structures [25]. In this paper, we used an actor–critic framework to avoid stochastic state changes due to agent interactions. Each agent has a policy network to execute actions and observe the state of the environment. The central review network collects all the actions performed by the agents, the observed status, and the rewards to judge the system. This ensures that there is a centralized network of reviewers who can update the overall training of the system at the time of training. When the whole system learns to converge, the policy network of all agents finds the best parameters to use. The final execution phase only requires that each agent acts through its own policy network.

The important policy gradient methods of Proximal Policy optimization (PPO) are taken to improve the performance of the DQN [26]. In this paper, the concept of a deep deterministic policy gradient (DDPG) [27] in the design of multi-agent path-finding is an improved algorithm based on the policy gradient (PG) [28] and deterministic policy gradient (DPG) [29]. The DDPG is a type of off-policy algorithm and can be regarded as a DQN that realizes a great action in explore space. To increase the robustness of DRL in different environments, this paper relied on the observation information required for reinforcement learning in robot simulation environments. It improved multi-agent research results in the path planning of multiple robot applications. In response to the other novel study, an experience replay training method is verified to achieve more smooth learning, reduce correlations, and facilitate offline training [30]. It is an indispensable tool for tackling a wide range of DRL challenges. In this study, a simple and efficient one-step experiment reply will be established. This experience replay is also a key focus of our future research in the development of mobile robot path planning application.

Currently, a lot of research has been conducted in robotics based on the Gazebo simulator [31,32]. Therefore, the Gazebo simulator can be regarded as a stable and reliable experimental platform. Due to the higher system requirements of the Gazebo simulator, reinforcement learning takes about 0.4 microseconds for each step of the Gazebo training. This results in a higher training time for more learning experiences to occur; therefore, this training simulation uses OpenAI's multi-agent particle environment to modify and design the required training model [33,34]. The sampling time for the training in this study was about 0.01 s to complete the simulation of the discrete action of the subsequent multi-agent. In this study, the simple particle multi-agent simulator uses its own robot position to capture a fixed-size field of view (FOV) as input to the neural network instead of using an occupation grid map of the entire domain and imports the local environment as observed from your local field of view. In addition to significantly reducing the computational burden of neural networks, it also helps the networks to be smoothly implemented in real robots.

Section 2 will explain the reinforcement learning and training environment development, the basic design of multi-agent path-finding will be explained in Section 3, a path-finding design with multi-agent deep deterministic policy gradients in Section 4, and experimental results and analyses in Section 5. Finally, conclusions and future works are discussed in the last section. The main contributions of this paper are as follows:

1. This study is a multi-agent reinforcement learning architecture combining centralized training and decentralized execution. In centralized training, a self-developed reward

function can enhance the multi-agent to complete the information aggregation of the training environment and effectively achieve the purpose of loss convergence.

2. The current design of training methods for multi-agent path-finding. Five different training environments combine the simple-to-complex schedule to enhance the learning efficiency. Based on the experimental results, the designed training schedule can effectively learn different environmental information and improve the capability of multi-agents.

2. Reinforcement Learning and Training Environment Development

The DRL machine in this paper is divided into a training phase and an application phase. In the training phase, a simple particle multi-agent simulator was made as the training environment to allow the agents to interact with the environment as much as possible. After completing the training cycle, the trained parameters of the network were then committed to the Gazebo simulator for validation. The system architecture of the reinforcement learning and the simple particle multi-agent simulator is shown in Figure 1. Different map data sets were used to reduce the complexity of the training cost; it was adjusted with the increment of training steps from a simple case to a complex one. More training strategies for reinforcement learning are explained in the following section. Observation information was provided by the environment as the state input for reinforcement learning, and the state information was received by the agent which decides the action through the policy network. The possible collision and the next moment's position were calculated by the physics engine. The computation of the reward function includes whether the physics engine has computed a collision or not. The details of the reward function are described in the next section. The reinforced system architecture of the Gazebo simulator for the application phase is shown in Figure 2.

When the Gazebo simulator is running, its node in the robot operating system is named gazebo_ros. Gazebo_ros was mainly used to capture information from the API, which was provided by the Gazebo simulator and sent to the robot operating system. The Gazebo simulator mainly emulates a virtual environment with mutual physical characteristics, robots, and sensors. Finally, all the information was transmitted to the robot operating system through gazebo_ros. This study focused on obtaining the localization information of all the robots and information on the collision and LIDAR sensors.

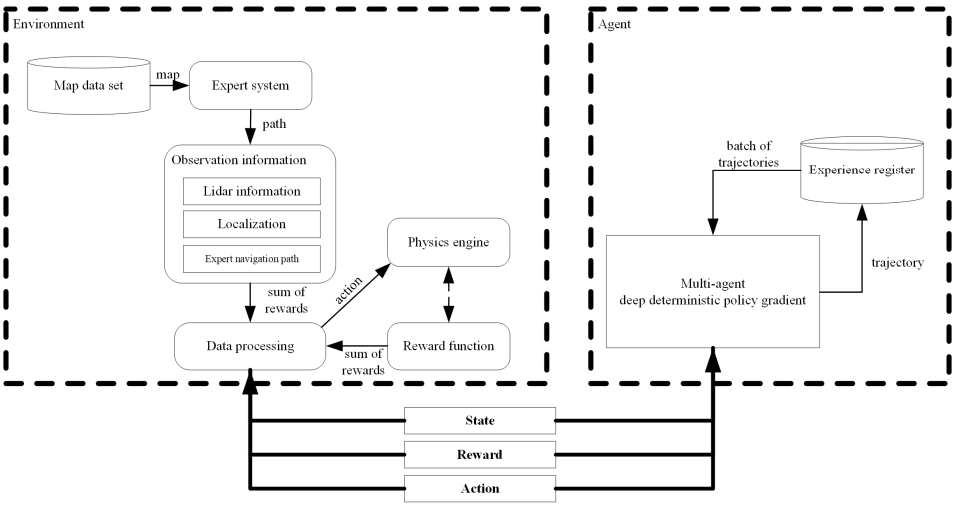


Figure 1. Reinforcement learning framework for simple particle multi-agent simulators.

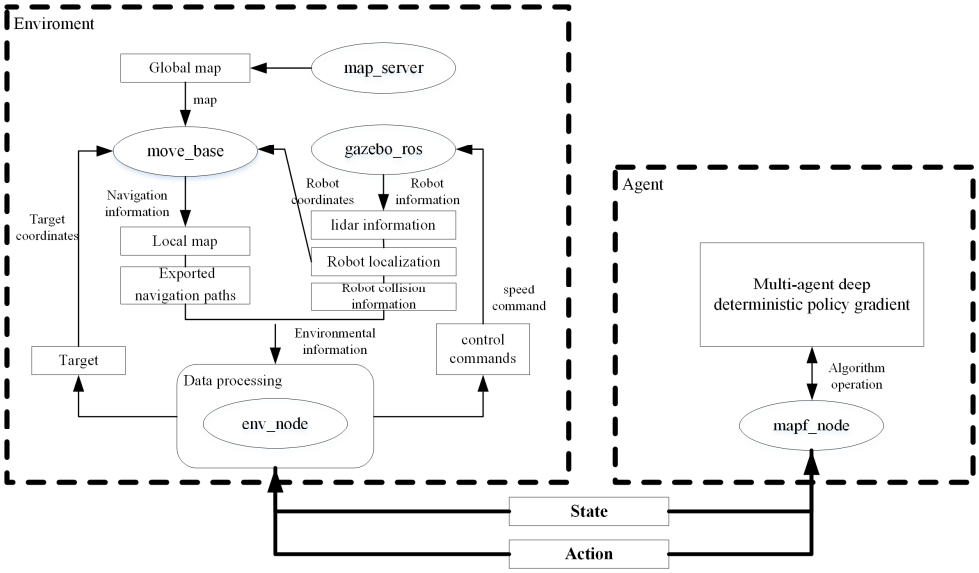


Figure 2. Architecture diagram of Gazebo simulator for reinforced learning applications.

More useful information was obtained through other software packages in the robot operating system (ROS) [35]. For example, map_server converts map information into an occupancy lattice map and publishes it in the ROS. move_base is a mobile robot navigation package that provides area map information based on the robot's location from a global map. The expert navigation paths are described in the following section. In addition to the simulated objects, the Gazebo simulator also provides the control signal to the simulated objects by the ROS.

In this paper, the implementation of a multi-agent reinforced learning architecture is more complex than that with only one single learning architecture. The state information related to the environment of all agents is large compared to the single-agent reinforced learning architecture. Therefore, in the framework of multi-agent reinforced learning, instead of using the whole environment as the state input of the agents, the local state information observed by the agents' own FOV was used as the input for reinforced learning. The local state information is called observation information, which is denoted as o_i, where i is the number of the agent; this is shown in Figure 3.

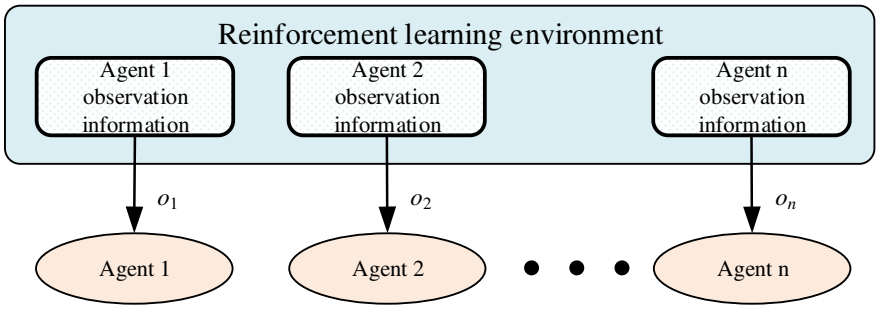


Figure 3. Observation information schematic diagram.

3. Basic Design of Multi-Agent Path-Finding

In the design of multi-agent path-finding, it originates from Q-learning, and it is known that if there is an optimal action value function $Q^*(s, a)$, the optimal action a*(s) can be obtained given a state s, as shown in Equation (1):

$$a^*(s) = \underset{a}{argmax} Q^*(s, a) \quad (1)$$

The DDPG algorithm allows the agent to continuously interact with the environment and updates the parameters according to the reward values, which are obtained from the reward function. However, if a continuous action space is used, it is very difficult to obtain the maximum Q value because it is not possible to exhaust all the actions. Approximating the maximum Q through a useful policy is the most efficient and feasible approach, as shown in Equation (2):

$$max_a Q(s,a) \approx Q(s,\mu(s)) \tag{2}$$

The optimal action value function $Q^*(s,a)$ can be written as Equation (3) using Berman's equation. *Q(s, a)* denotes the optimal expectation value obtained by performing an action in the current state s, where *Q(s, a)* is s′ the state sampled from the environment, and the distribution of the state transfer probability is denoted as $P(\cdot|s,a)$. Because future expectations are less relevant to the present moment, a discount rate γ represents the relative importance of future rewards compared to immediate rewards. A higher discount rate prioritizes long-term rewards, while a lower discount rate focuses on immediate gains. γ = 0.95 is added to reduce the dependence on future expectations.

$$Q^*(s,a) = \underset{s' \sim P(\cdot|s,a)}{E}\left[r(s,a) + \gamma \underset{a'}{max} Q^*(s',a')\right] \tag{3}$$

The mean squared Bellman Error (MSBE) is used in the design of the action value network to estimate the value of the error of the network, which is with respect to the optimal action value function and is expressed as Equation (4), where y_ϕ is the target network of $Q_\phi(s,a)$. r is the reward value, and d is termination status.

$$L(\phi, D) = E_{(s,a,r,s',d)\sim D}\left[\left(Q_\phi(s,a) - y_\phi\right)^2\right] \tag{4}$$

The purpose of the action value network is to calculate the reward value, which is expected to be obtained by performing action A in a state S. It is denoted in Equation (5):

$$y_\phi = r + \gamma(1-d)\underset{a'}{max} Q_\phi(s',a') \tag{5}$$

in the training process, the target network y_ϕ is regarded as the best action value function, and the error value (loss) $L(\phi, D)$ is referred to as the loss between the current action value network and the target network. The system calculates the mean square deviation between the current action value network $Q_\phi(s,a)$ and the target network for training verification. In this case, the value of d is either 0 or 1 and is used to represent the termination state. The agent will not calculate the reward value after reaching the termination state. Further, D is a register for storing the trajectory of past experiences. The DDPG learns by experience replay, which is similar to the DQN. This is an efficient way to utilize the experience of different rounds of the training cycle through offline learning methodology. A larger replay buffer provides more data for training, potentially improving policy accuracy, but may increase memory usage.

The depth deterministic policy gradient also refers to the DDQN, which takes advantage of the target network to avoid the problem of overestimation. The network parameters are those of the target network ϕ_{target}. The final loss function can be expressed as Equation (6):

$$L(\phi, D) = E_{(s,a,r,s',d)\sim D}\left[\left(Q_\phi(s,a) - y_{\phi_{\text{target}}}\right)^2\right] \tag{6}$$

where

$$y_{\phi_{\text{target}}} = r + \gamma(1-d)\underset{a'}{max} Q_{\phi_{\text{target}}}(s',a') \tag{7}$$

In Equation (7), action a' is also the estimated response through the target policy $\mu_{\phi_{\text{target}}}(s')$. Therefore, it can be expressed as Equation (8) below:

$$y_{\phi_{\text{target}}} = r + \gamma(1-d)\underset{a'}{max} Q_{\phi_{\text{target}}}(s', \mu_{\phi_{\text{target}}}(s')) \tag{8}$$

Register D holds a large number of past experiences and consumes a lot of resources since it needs to frequently count all past experiences and update them continuously. Therefore, the action value network is updated by sampling a batch size of experience trace B from the temporary register. This is shown in Equation (9):

$$B = \{t \in D \,||t| = \text{batch-size}\} \tag{9}$$

t denotes the current trajectory, which is randomly sampled from the register of past experiences. The whole batch experience trajectory B is obtained from one batch size. The batch size used in this paper is 1024.

Neural network parameters are updated with the gradient descent method; its work is represented in Equation (10):

$$\nabla_\phi J = \nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d)\in B} \left(Q_\phi(s,a) - y_{\phi_{\text{target}}}\right)^2 \tag{10}$$

The gradient value is approached by batch experience trajectory B, where the sum of the mean square deviations is obtained by dividing the batch size.

In the deep deterministic policy gradient, the goal of the policy network $\mu_\theta(s)$ is to maximize the action value network, where θ is mathematically expressed and is calculated as Equation (11):

$$\nabla_\theta J \approx max_\theta E_{s\sim D}\left[Q_\phi(s, \mu_\theta(s))\right] \tag{11}$$

The gradient representation is shown in Equation (12):

$$\nabla_\theta J = \nabla_\phi \frac{1}{|B|} \sum_{s\in B} Q_\phi(s, \mu_\theta(s)) \tag{12}$$

Each action value network $Q_\phi(s,a)$ and policy network $\mu_\theta(s)$ has a target network $Q_{\phi_{\text{target}}}(s,a)$ and $\mu_{\theta_{\text{target}}}(s)$, respectively. Once the action value network and strategy network have been updated, the target network needs to be updated at the same time. The target network is updated through Equation (13):

$$\begin{aligned} \phi_{\text{target}} &\leftarrow \rho\phi_{\text{target}} + (1-\rho)\phi \\ \theta_{\text{target}} &\leftarrow \rho\theta_{\text{target}} + (1-\rho)\theta \end{aligned} \tag{13}$$

In Equation (13), $\rho = 0.01$ is the soft updated rate. A higher ρ value slows down target network updates, providing smoother policy improvements, while a lower ρ value speeds up updates but may introduce instability. Even if the target network parameters ϕ_{target} and θ_{target} of the action value network and the policy network are using the current network parameters ϕ and θ, the update of their network parameters is gradually completed incrementally.

The architecture of the depth deterministic policy gradient is shown in Figure 4. At the beginning of the flowchart, the parameters ϕ and θ of the action value network and the policy network are initialized, and the register D is cleared. The parameters of the target network from the action value network and the policy network are also initialized by $\phi_{\text{target}} \leftarrow \phi$ and $\theta_{\text{target}} \leftarrow \theta$ from the current network state.

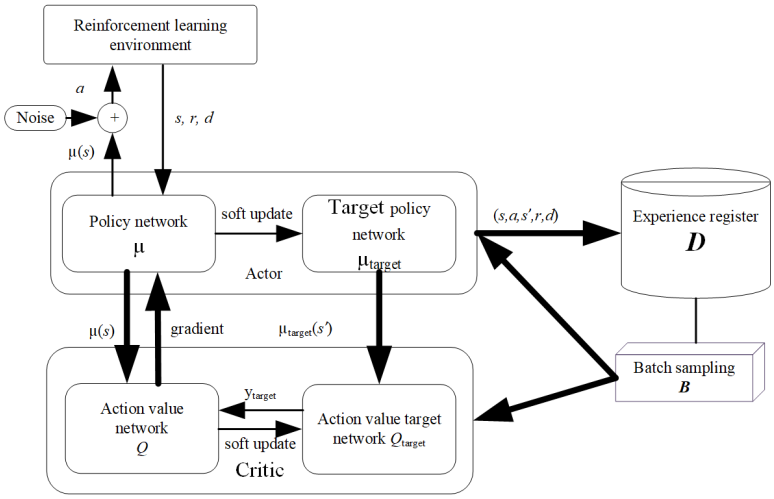


Figure 4. The architecture of the deep deterministic policy gradient.

In each training round, the training environment is first reset; the agent obtains the states of the environment. The action a is selected via the strategy network, and Gaussian noise ε [36] is added to improve the ability and effectiveness to optimally explore the space in response to changes in its environment. It should be noted that noise was only used during the training cycle. No noise was taken to improve the exploration after completing the training round. On the other hand, the action value a was set in such a way that it will not exceed the range in which the agent can normally operate.

In this study, action a is performed in the training environment to get to the next state s'; then, the reward value r is given by the environment and the termination state d. Then, the track (s, a, r, s', d) of this step is saved into register D. If action a is terminated, the environment is reset again, and a new turn begins.

While the algorithm executes several rounds to reach the condition of updating the network parameters, it samples a batch size of data B from register D. It calculates the gradient of the action value network and the strategy network with the trajectory of data B and updates it as the explained range from the action value network and strategy network. Finally, the soft updated method is proposed to complete the parameters of the target network.

4. Path-Finding Design with Multi-Agent Deep Deterministic Policy Gradients

A multi-agent deep deterministic policy gradient was applied in this study. The architecture diagram is shown in Figure 5. We combined the centralized training and decentralized execution architecture for the multi-agent reinforcement learning target for path-finding. In this study, there were N agents under a multi-agent reinforcement learning cycle, where I denotes the agent number. Each agent had its own local observation information. The observation information obtained by the agent is denoted as o_i, and the observation information of all agents is denoted as $\vec{o} = o_1, \ldots, o_N$.

The action performed by each agent is represented as a_i, and the actions performed by all agents are denoted as $\vec{a} = a_1, \ldots, a_N$. The policies of all agents are represented as $\vec{\mu} = \mu_1, \ldots, \mu_N$ with their parameter set at $\vec{\theta} = \theta_1, \ldots, \theta_N$.

The transfer function between the observation information and the action set was expressed as $\mu_{\theta_i} : o_i \mapsto a_i$. N agents interact together in a training environment, and all agents interacting in the environment are viewed as an overall set of states S.

Although, Markov games capture the intricate interactions between multiple agents for the more realistic modeling of MARL scenarios. Because each agent can only observe its own local observation information, it can also be regarded as a partially observable Markov

Decision process. But this computational complexity is large. This study treated the multi-agent reinforcement learning as a multi-agent version of Markov Decision processes [37].

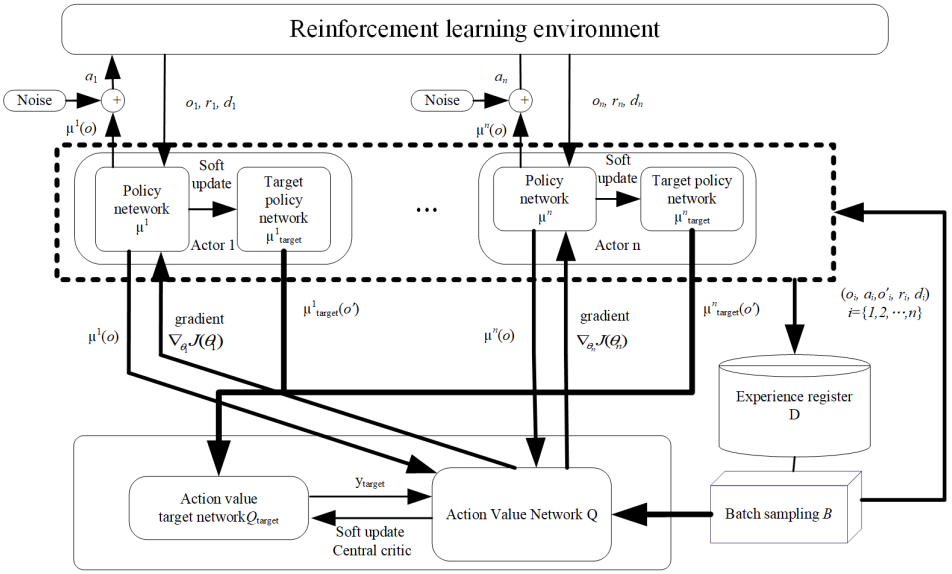


Figure 5. Multi-agent deep deterministic policy gradient flowchart diagram.

The learning difficulty of multi-agent reinforcement learning lies in the fact that the overall system state transfer function can be expressed as Equation (14).

$$T : S \times a_1 \times \ldots \times a_N \mapsto S' \tag{14}$$

In the centralized training part, the central critic of the action value network mainly collected the traces of all the agents and evaluated the actions performed by each actor. Actors can obtain gradients from the central critic and update their network parameters. The gradient representation of the actor update is shown in Equation (15).

$$\nabla_{\theta_i} J(\theta_i) = E_{\vec{o},a\sim D}\left[\nabla_{a_i} Q_i^{\vec{\mu}}(\vec{o}, a_1, \ldots, a_N)\nabla_{\theta_i}\mu_{\theta_i}(o_i)|_{a_i=\mu_{\theta_i}(o_i)}\right] \tag{15}$$

The central critic's loss function is calculated using Equation (16), where y is the target function as shown in Equation (17). In Equation (17), r_i denotes the reward value obtained from the environment by agent i when it is in state $\vec{o}'$, and all the agents take individual action a_i'.

$$L(\theta_i) = E_{\vec{o},a_1,\ldots,a_N,r_1,\ldots,r_N,\vec{o}'}\left[\left(Q_i^{\vec{\mu}}(\vec{o}, a_1, \ldots, a_N) - y\right)^2\right] \tag{16}$$

$$y = r_i + \gamma Q_i^{\vec{\mu}}(\vec{o}', a_1', \ldots, a_N')|_{a_j'=\mu_{\theta_j}'} \tag{17}$$

This paper used multi-agent deep deterministic policy gradients as the trainer for the path-finding reinforcement learning results. In this case, each agent had a strategy network and action value network of the central critic, and each policy had a target policy network. The action value network also had a target network. The central critic's network input was used for all agent observation information and actions.

The central critic network is for the action value network. Its purpose is to evaluate the state of the agent and the value of the actions performed by the agent. Therefore, in addition to the observation information, the network input also included the actions performed by the agent as input to the network. This paper involves a multi-agent reinforcement learning architecture using decentralized execution with a centralized training cycle. The central

critic's input is a stack of observation information and execution actions from all agents used as input to the network. The final output was the value that the central critic places on the actions taken by all actors. The system architecture is shown in Figure 6.

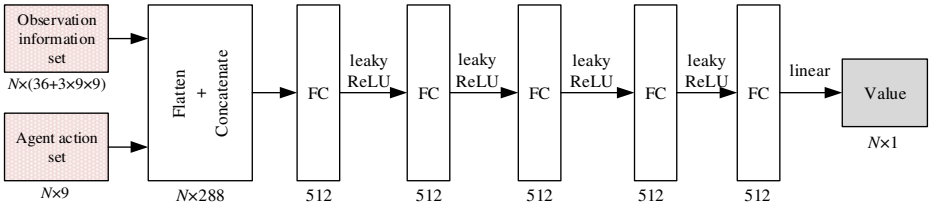


Figure 6. The architecture of the central critic network.

In Figure 6, N is denoted as the number of all agents. The observation information, i.e., the design of the area observation information presented in this paper, includes the following: the dimension size of the agent information was 36, and the collaborative information included three 9 × 9 lattice maps. Each agent had a discrete set of actions of size nine. Finally, the observation information of all agents and the set of actions within all agents were amalgamated and combined. Therefore, the dimension size was N × 288, which was then inputted to the hidden layer with a dimension size of 512. The overall structure consisted of five layers.

An actor network is a policy network that allows the agent to decide what to perform based on the observed state of the environment. The architecture of the agent network is shown in Figure 7. The final output included nine values, which were the size of each discrete action calculated by the neurons of the strategy network based on the input information and the weight values. The related action mapping value of the operator's outputs is shown in Table 1. In practice, the maximum value of these nine discrete actions is taken and executed as the calculation of Equation (18), where u^i is the final action performed by agent i.

$$u^i = argmax\left(\mu_\theta^i\left(o^i\right)\right) \tag{18}$$

Table 1. Policy network output and action mapping table.

Action Number	Actual Action
0	Stop
1	$\overrightarrow{v} = (1, 0)$ Rightwards and run in the simulator
2	$\overrightarrow{v} = (1, 1)$ Right upper and run in the simulator
3	$\overrightarrow{v} = (0, 1)$ Upwards and run in the simulator
4	$\overrightarrow{v} = (-1, 1)$ Left upper and run in the simulator
5	$\overrightarrow{v} = (-1, 0)$ Leftwards and run in the simulator
6	$\overrightarrow{v} = (-1, -1)$ Left down and run in the simulator
7	$\overrightarrow{v} = (0, -1)$ Downwards and run in the simulator
8	$\overrightarrow{v} = (1, -1)$ Right down and run in the simulator

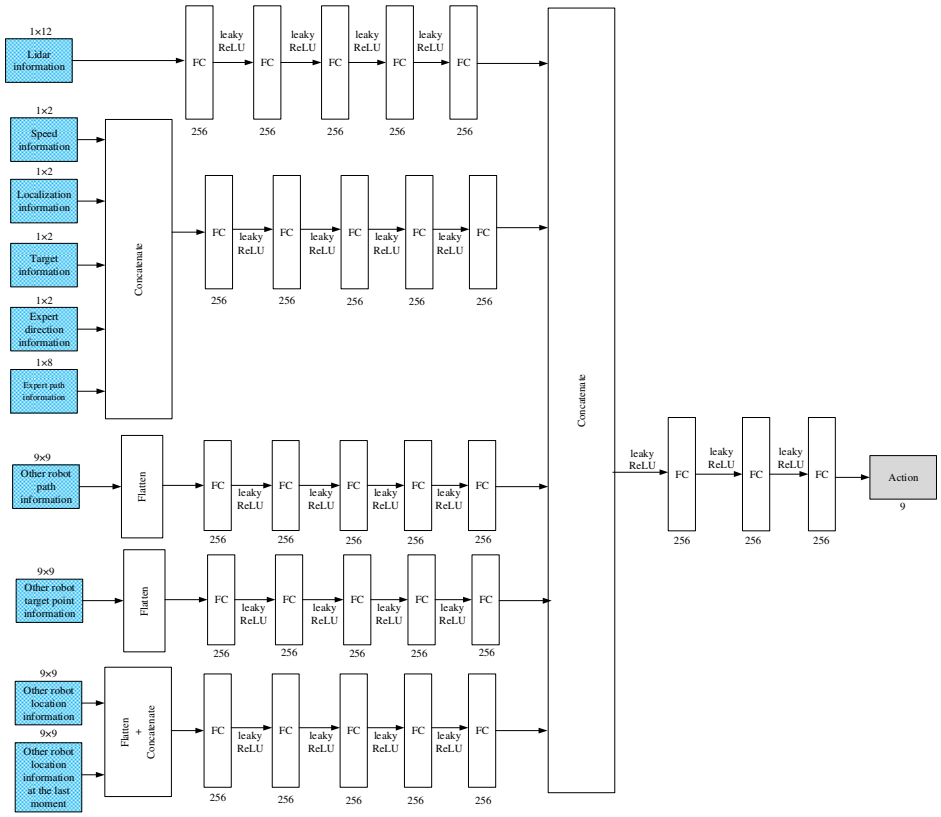


Figure 7. The system architecture of the action network.

The design of the reward function in this paper allows the mobile robot to move into the target point while maintaining an effective operation of the overall system. It also fulfills the desired objectives shown in Table 2.

Table 2. Reward function design.

State	Rewards	Objective
Achievement of Targets r_{goal}	$exp(-k1 \times \text{dist})$	Obj 1
Occurrence of collisions $r_{collision}$	collision_times	Obj 2
Repulsion Award $r_{collision\_force}$	$-max(exp(-k2 \times d(P_{\text{agent}}, I)))$	Obj 3
Expert Direction Award r_{dir}	$\begin{cases} -k3 & \frac{u}{\lvert u \rvert} = \frac{v}{\lvert v \rvert} \\ -k4 \text{ otherwise} \end{cases}$	Obj 4
Stop Rewards r_{stop}	$-k5$	Obj 5

Obj 1: The achievement of the target to obtain the reward and set the termination condition for the agent to be reached. The reward function is $exp(-k1 \times \text{dist})$, and the distance between the agent and its target is represented by dist. $k1 = 10$ is used to adjust the degree of convergence between dist size and rewards. Obj 2: The reward value of collision_times for a collision, where collision_times is the number of collisions. Obj 3: The reward value of $-max(exp(-k2 \times d(P_{\text{agent}}, I)))$ is used to calculate the repulsive force; the closer the distance, the greater the repulsive force, in which I is the set of vectors from the agent to other objects. $k2 = 0.2$ is the experience value, which can be adjusted for different distance sizes and repulsive force variations. Obj 4: V denotes the expert action vector, and u denotes the agent action vector. When the agent performs an action that is different from the one planned by the expert, it is rewarded poorly. The size of $k3 = 0.2$ is chosen mainly in the same value as Obj 2. $k4 = 0.4$ is worse, because the direction is different from the expert. Obj 5: When the agent stops moving, it gives a negative reward. $k5 = 0.8$ is chosen based on the fact that the severity of stop rewards is 4 times the size of the collision.

The current paper is designed to address the five states encountered during the training process of path-finding reinforcement learning. The first is the state of reaching the target point, which is a relatively rare state in the environment during training. If the reward function was designed to be too sparse in this state, it may result in the state being of little benefit in the overall learning. Therefore, this paper used an exponential function to design the reward for reaching the target point, where the agent is rewarded for being within a certain distance. As the distance between the agent and the target is shortened, the agent can obtain a higher reward value.

The second state is a collision, which is inevitable during training. This paper did not consider collision as a termination condition because collision gives negative rewards. All of these collisions were used in the experience replay. Setting it as a termination condition would instead make the collision condition sparse. In order to avoid continuous collisions between the agents, it was set in a way that as the number of collisions increased, the heavier the penalty it received.

The third state is the repulsive state. To avoid the state of collision being too sparse, the distance between the agent and other objects was regarded as the repulsion force. The repulsive force increases as one gets closer, and the repulsive force is summed up in the reward function. This award not only enhanced the agent's ability to avoid collisions but also densified the state of the collision.

The fourth state is the expert direction, and the A* algorithm was used as a guide for the path. The reason for this is that if we used the straight-line distance of the target point as a reference, we may encounter the problem of having a wall in between despite having a very close distance. If this happens, the agent will be required to perform a detour. Therefore, this paper treated the expert's proposed action as a preferred option. However, if the agent performed other actions, it would only be a little bit worse than the expert action. The difference between the two different action rewards would not cause the agent to follow exactly the expert's path. Instead, it will just be a reference to what the experts were doing.

The last state is a stop state. In order to motivate the agent to explore, a negative reward was given when the agent chose to stop moving.

The action taken by the agent obtains the reward value of the step according to the reward function designed in this paper. It is calculated using Equation (19).

$$r^i = S1 \times r^i_{goal} + S2 \times r^i_{collision} + r^i_{collision\_force} + r^i_{dir} + r^i_{stop} \tag{19}$$

where i is regarded as the agent number. $S1$ is selected as 5 to regulate the overall formula. $S2 = -0.2$ is mainly an empirical value, and it can be used as a reduction in the reward value for collisions.

5. Experimental Results and Analyses

The main objective of this experiment was to validate the effectiveness of the proposed multi-agent path-reinforcement learning based on deep deterministic policy gradients. In the validation process, different training environments and numbers of agents were analyzed and explored.

In the experimental environment, the software learning framework used was PyTorch 3.7. Linux Ubuntu 20.04, deep learning framework paddle 2.3.0, hardware acceleration framework CUDA 11.1, and hardware acceleration library cuDNN 8 were also used.

This study uses an RFL framework with centralized training and decentralized structure. Therefore, the complexity calculation mainly includes the number of computations required for training the data during the training cycle and the number of computations required for decentralized execution and agent interaction during application. This is because the complexity of centralized training is much higher than the amount of computation involved in decentralized execution. For multi-agent interactions, the effects of inter-agent interactions are approximated to the constant calculation time (CT) due to the locality of the individual agent using partially observed information. So, the complexity of this study

will focus on the computation of each iteration during the centralized training. Based on the general principles of the previous literature [38,39], the following presentations are deduced to be the key points to be considered for the calculation of the complexity in the centralized training and decentralized execution framework of the DDPG for this paper.

1. Number of neural network parameters (θ): DDPG algorithms usually contain two main neural networks: the actor network and the policy network. Each network has its own parameters, mainly weights and biases, and the number of parameters in a neural network is an important factor in the complexity of the calculation.
2. Dimensions of state space and action space. During the centralized training phase, the states and actions of all agents may be combined into a global state and action vector. Therefore, the state and action dimensions will be extended to the sum of all agent state and action dimensions.
3. DDPG algorithms usually require a large amount of training data (states, actions, rewards and next states sampled from the environment) to train effectively. In addition, the number of iterations in the training process also directly affects the computational complexity. The approximate complexity formula (CF) used in this study is as follows:

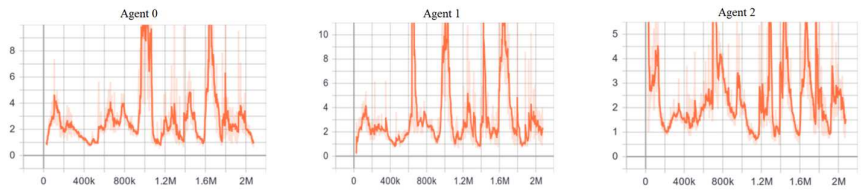
$$\mathrm{CF}(\theta_A, \theta_C, d_s, d_a, N, \eta, B) = \eta \times \mathrm{CT} \times (B \times ((\theta_A \times N \times d_s) + (\theta_C \times N \times (ds + d_a)))) \quad (20)$$

where the state dimension of each agent is d_s, and the action dimension of each agent is d_a. The total number of agents is N. In addition, θ_A is the total number of the parameters of the actor network, and θ_C is the total number of the parameters of the critic network. η is the total update number in the training process, and B is the size of the batch processed in each iteration of the training process.

- **Case 1: Agent Reinforcement Learning—Decentralized Validation**

A multi-agent reinforcement learning architecture using centralized training and decentralized execution is illustrated in Figure 5. To enhance multi-agent collaboration in observing the information center for agents, this paper used the concordance information as input to the neural network in the form of occupied lattice maps. The advantage of this approach is that different numbers of agents can be used for training and execution. This was utilized since during execution, it only needed to copy the trained policy network parameters to other agents.

Firstly, this paper used three agents (agent 0, agent 1, and agent 2) for training during the centralized training process. The training results are shown in Figure 8. The x-axis is the number of training iterations; the y-axis is the loss value. The loss function is the amount of error calculated by the central critic on the three agents during the centralized training, where the smoothing parameter for the data graph was 0.8. The dotted lines are the original values and the solid lines are the experimental results of loss value with smoothing method.



(**a**) Agent 0 (**b**) Agent 1 (**c**) Agent 2

Figure 8. Central critics' response to loss functions of agents.

Experimentally, it can be seen that the loss function of the central critic varied for different agents. This is because each agent observed a different state. Different strategy networks also performed different actions with corresponding reward values. However, the performances of the three proxies were similar, partly because they used the same

reward function. The random generation of locations can help the agent to learn different experiences during training.

Figure 9 shows the average value of rewards received by all agents in each round, where the x-axis is the number of training iterations, and the y-axis is the reward value. It can be observed that as the number of rounds increased, the number of times the agent interacted with the environment also increased. The agent also learned from the reward values obtained by the environment how to achieve better results from the designed training environment.

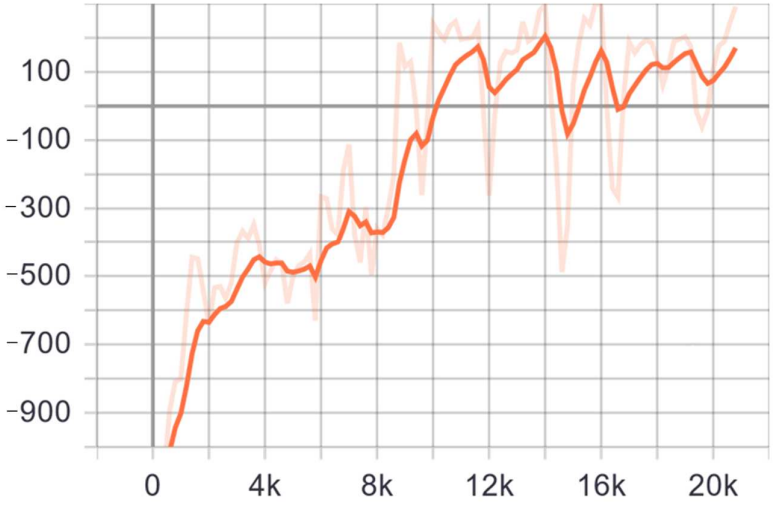
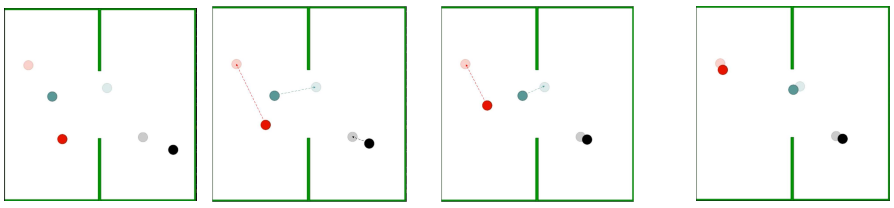


Figure 9. Average round reward value of agents. The dotted and solid line lines present the original and smoothing values, respectively.

The simulation results of the actual interaction with the environment are shown in Figure 10, where the red, black, and green agent locations and their target locations are randomly generated in the environment. The target locations are represented as semi-transparent dots of the corresponding colors. Figure 10a shows the initial locations of the randomly generated agents and targets. Figure 10b shows that each agent starts to move towards the target point. Figure 10c shows the black agent moving to the target point and waiting for other agents to reach the target point. Figure 10d shows the target point for all agents.

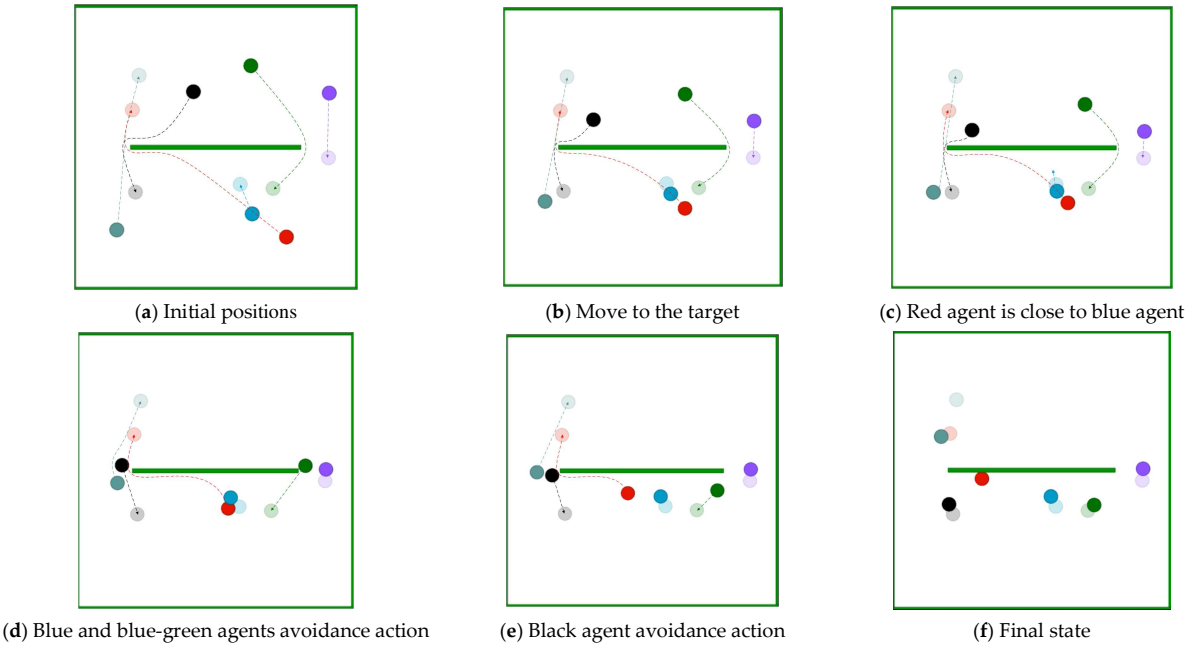


(**a**)Initial positions (**b**)Move to target (**c**)Black circle to target (**d**)Achievement of target

Figure 10. Mirror image of 3 agents in training environment. The different color dotted-line indicate different trajectories of agents, solid circles represent initial positions of different agents and different semi-transparent circles represent targets.

The method of expanding the proxy experiment is to add three more proxies from the previous experiment of three proxies, so there are six proxies in total. The experimental results are shown in the split-mirror diagram in Figure 11. Figure 11a shows the movement of the random generator agent towards the target position. Figure 11b shows the waiting for the end of the round for the agent that has reached the target point and confirming if there is an avoidance action to be performed. Figure 11c shows that the red agent path passes through the blue agent, so Figure 11d shows that the blue agent avoids the red agent

path. In the left half of Figure 11e, it can also be observed that the black and blue-green agents are avoiding each other when they meet because they want to reach their respective targets. Figure 11f shows the final state of the round. These results show that most of the agents can reach their respective targets, and the red and blue-green agents will terminate the round because they have reached the maximum number of steps in the round.



(**a**) Initial positions (**b**) Move to the target (**c**) Red agent is close to blue agent

(**d**) Blue and blue-green agents avoidance action (**e**) Black agent avoidance action (**f**) Final state

Figure 11. Expanded 6-agent mirror experiment. The different color dotted-line indicate different trajectories of agents, solid circles represent initial positions of different agents and different semi-transparent circles represent targets.

- **Case 2: Experimental analysis of different environments**

The aim of this example in analyzing different environments was to determine the effectiveness of the overall system development and operation. The collision rate and average operating time were used as the criteria for evaluating the overall system operation. Figure 12 shows the proposed experiment setup consisting of five training environments of varying environmental complexity. The focus of this paper on training neural networks for DRL is developed in a progressive learning approach. In the training procedure, its sequence is designed to start from the low-complexity Environments 1 and 2 of Figure 12, move into the medium-complexity Environments 3 and 4, and then to the high-complexity Environment 5. The switching strategy of the training environment is based on the number of iterations of the interaction with the environment to actually switch to the next new training environment by the sequence of environmental complexity. The training rounds for Environment 1 fall in the range of 0–1000, Environment 2 in the range of 1001–2000, Environment 3 in the range of 2001–3000, Environment 4 in the range of 3001–4000, and Environment 5 in the range of 4001–5000. In addition, it is designed to ensure that the learning process is accelerated, while the training radius is selected within the specific distribution of the target, whose training range is gradually expanded from small to large. In our experiment, the duration of each training round is controlled to less than 5000 times for a quick convergence test in this case.

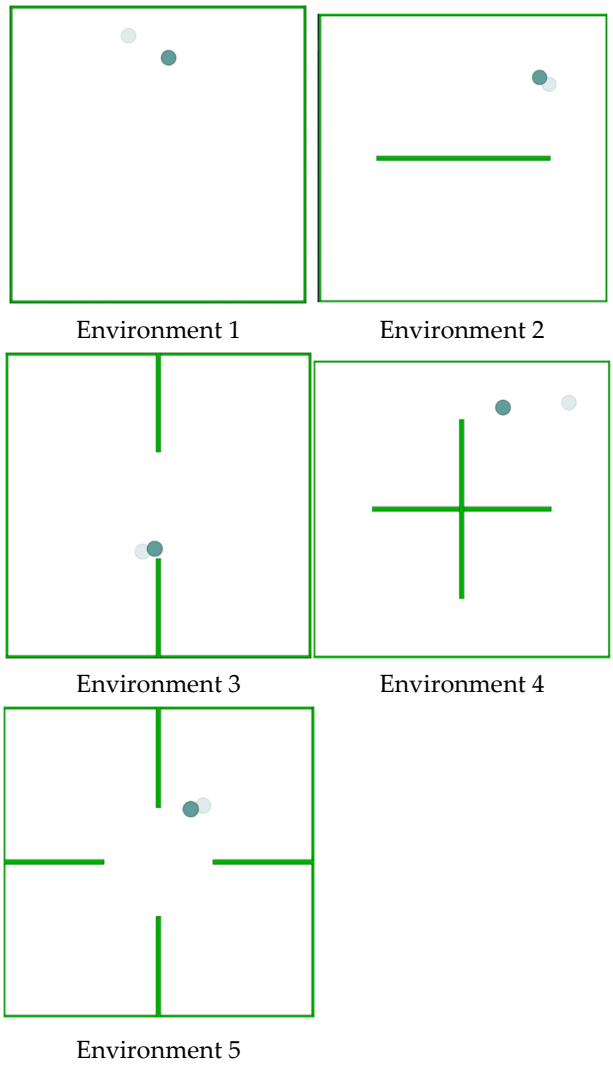


Figure 12. Training environment with 5 different complexities. The solid circles represent initial positions of different agents and different semi-transparent circles represent targets.

A total of 100 rounds were run for each environment, and the time and number of collisions to reach the target point for each round were calculated. The data results are presented in terms of the average time spent and average number of collisions for 100 rounds.

The experimental numerical analysis is shown in Tables 3 and 4. It can be seen in Table 3 that the time taken by all the agents to reach their respective destinations varied depending on the environment, and they were all able to reach the target. The time in Table 3 is denoted as the simulated time. Based on the analysis, it was found that the agent operated very efficiently from the start point to the target in this simulated environment. The results show that the more complex the environment, the longer the time that is required. As shown in Table 4, all had a great deal of interaction and training accuracy. In Environments 1 and 2, no collision occurred due to the relative simplicity of the environment. In Environment 3, Environment 4, and Environment 5, because the agent needed to pass through an intermediate region, the chance for collision increased. The worst total collision ratio was 38% percent, i.e., its worst success ratio was 62%. This indicates that the experiment result had a better success rate within than in the model-free multi-agent soft actor–critic (25.9%) and multi-agent model-based policy optimization (MAMBPO) (37.1%) [25]. According to the analysis of the collision probability after learning, more different environments can be redesigned for specific requirements in the future.

Table 3. Average time spent by 4 agents in different environments (simulation time).

Agentnt / Environment	Agent 1	Agent 2	Agent 3	Agent 4
Environment 1	0.203	0.233	0.217	0.205
Environment 2	0.305	0.316	0.351	0.323
Environment 3	0.410	0.414	0.428	0.435
Environment 4	0.512	0.586	0.579	0.538
Environment 5	0.615	0.622	0.628	0.639

Table 4. The average collision rate of 4 agents in different environments.

Agentnt / Environment	Agent 1	Agent 2	Agent 3	Agent 4
Environment 1	0.00	0.00	0.00	0.00
Environment 2	0.00	0.00	0.00	0.00
Environment 3	7%	6%	5%	5%
Environment 4	9%	12%	11%	13%
Environment 5	18%	18%	19%	20%
Total	34%	35%	35%	38%

- **Case 3: Motion Approximation Experiment**

The final experimental result of this paper is presented in a video where the operation of each agent's execution state and must-be-allowed policies learned between agents can be observed. The experimental video is divided into two parts; the first one is the experimental video of the four agents (https://youtu.be/30EO3bLNnNM; accessed on 3 July 2023), and the second part is the experimental video of six agents (https://youtu.be/OOachwbKgCI; accessed on 3 July 2023).

As shown in the videos, one of the training environments was used randomly in each round, and the agents and targets in the environment were randomly generated. The agent is presented by the solid circle color code, and the target points are indicated by the corresponding colors of the translucent circles.

The interaction of the four agents in Environment 4 is shown in Figure 13. Figure 13a shows the randomly generated agent and target locations. Figure 13b,c show the red and purple agents arriving at the target, waiting for the end of the turn, and confirming whether there is an avoidance action to be taken. In Figure 13d–f, the black agents can be seen. In the process of traveling to the target point, the black proxy had to pass through two proxies that had already reached the target point, namely the purple and the blue-green agents. Each of these two agents took an avoidance action to avoid blocking the black agent from reaching its target point. Figure 13h shows the final status at the end of the round.

The interaction of the four agents in Environment 5 is shown in Figure 14. This can be seen at the 01:22~01:36 time stamp in the experimental video. Figure 14a shows the randomly generated agent and target locations. Figure 14b,c show each agent going to the destination. Figure 14d–g show the continuous graph of the purple agent reaching the target point. Since the black agent had to pass through the location of the purple agent, the purple agent's avoidance behavior allowed all agents to reach their respective destinations smoothly. Figure 14h shows the final status at the end of the round.

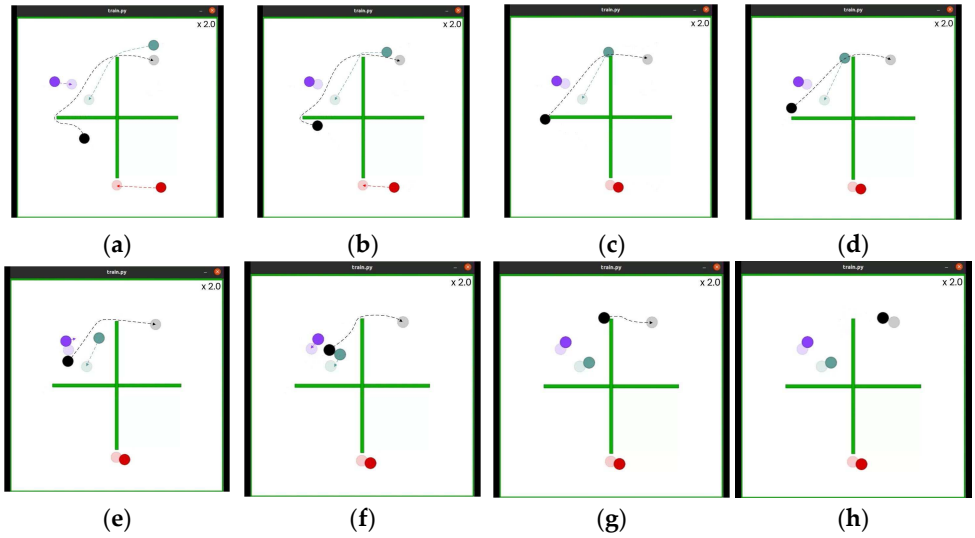


Figure 13. The interaction navigation of 4 agents in the environment 4. The different color dotted-line indicate different trajectories of agents, solid circles represent initial positions of different agents and different semi-transparent circles represent targets. (**a**) Initial positions. (**b**) Red agent approaches target. (**c**) Purple and red agents reach target. (**d**) Black Agent ready for target. (**e**) Black agent goes to target. (**f**) Black agent for obstacle crossing action. (**g**) Black agent completes the obstacle course. (**h**) Achievement of targets.

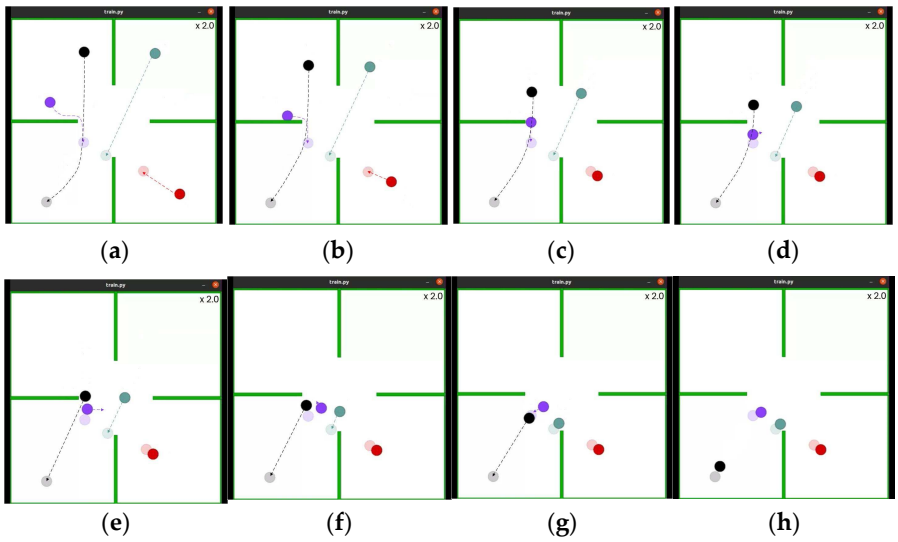


Figure 14. The interaction navigation of 4 agents in Environment 5. The different color dotted-line indicate different trajectories of agents, solid circles represent initial positions of different agents and different semi-transparent circles represent targets. (**a**) Initial positions. (**b**) 4 agents moving towards the target. (**c**) Red agent reaches target. (**d**) Black agent reaches target. (**e**) Purple agent backs off. (**f**) Black agent goes through obstacle. (**g**) Green agent reaches target. (**h**) All agents reach targets.

The interaction of four agents in an unknown environment is shown in Figure 15. This can be seen at the 01:44~02:00 time stamp in the experimental film. Figure 15a shows the randomly generated agent and target locations. Figure 15b,c show the agents going to the target and avoiding each other. Figure 15e–g show the successive cases where the black and purple proxies met below the environment and then fell into the region's best solution.

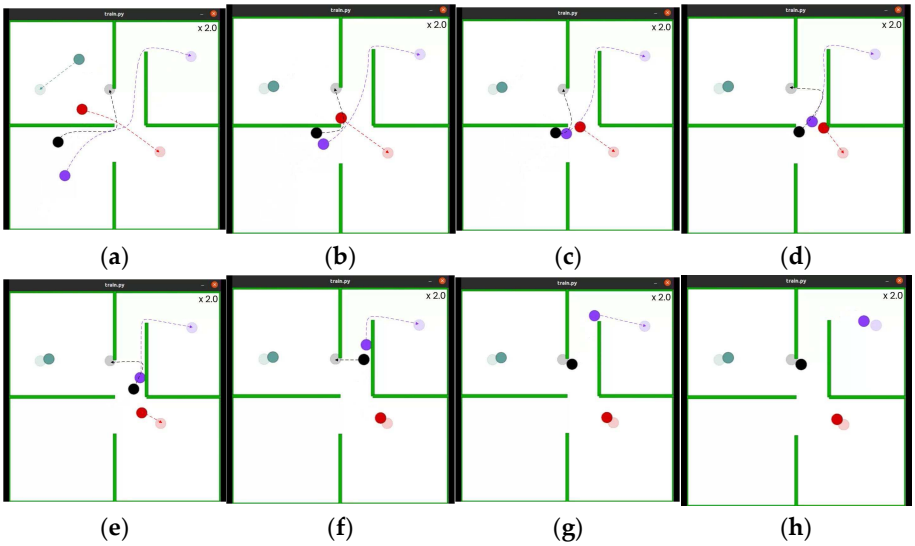


Figure 15. The interaction navigation of 4 agents in an unknown environment. The different color dotted-line indicate different trajectories of agents, solid circles represent initial positions of different agents and different semi-transparent circles represent targets. (**a**) Initial positions. (**b**) Blue-green agent reaches target. (**c**) Purple agent goes through obstacle. (**d**) Black agent goes through obstacles. (**e**) Purple and black agents move towards targets. (**f**) Red agent reaches target. (**g**) Black agent reaches target. (**h**) All agents reach tar-gets.

6. Conclusions and Future Works

Hybrid centralized training and decentralized execution reinforcement learning in multi-agent path-finding is proposed in this paper to show excellent results with a higher success rate and quick convergence. However, there are still a few items that can be investigated more deeply for the research topic of this paper, which are described as follows:

1. The design of the reward function is currently premised on a lot of prior knowledge. However, in addition to the effort involved in the design process, it is possible that we may come across situations where it is difficult to design a reward function for a target. Therefore, in the future, we can consider adding imitation learning or inverse reinforcement learning to replace the problem of manually designing reward functions.
2. In the part of the neural network design, we can further explore the learning method of optimizing the number of network layers and related parameters in the future. We will try to improve the problem of designing neural network architectures that are often adjusted by reference to the literature or by trial and error.
3. In the observation information, this paper divides the agent and other agents into personal information and collaborative information. In the future, we can further explore whether there is unnecessary information in the observation information or whether other useful information can be obtained from the environment. In addition, this paper adds only the last-moment agent location information to the current state observation for experience information replay. In the future, the use of recurrent neural networks or long short-term memory (LSTM) can be investigated to assist the agent in learning the related sequence data.

Author Contributions: Conceptualization, S.-A.L. and H.-M.F.; methodology, H.-C.C., S.-A.L., T.-H.C. and H.-M.F.; software, T.-H.C. and Y.-C.C.; validation, H.-C.C., T.-H.C. and Y.-C.C.; formal analysis, S.-A.L. and H.-M.F.; investigation, T.-H.C. and Y.-C.C.; resources, H.-C.C., T.-H.C. and Y.-C.C.; data curation, S.-A.L. and H.-M.F.; writing—original draft preparation, T.-H.C.; writing—review and editing, S.-A.L. and H.-M.F.; visualization, Y.-C.C.; project administration, H.-M.F. and H.-C.C.; funding acquisition, H.-C.C., S.-A.L. and H.-M.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partly funded by the high-level talent research project of Xiamen Ocean Vocational and Technical College, with project number 2022001. This research was also partly funded by both Ministry of Science and Technology (MOST) of the Republic of China, under grant numbers MOST 111-2221-E-032-030, and National Science and Technology Council (NSTC) of the Republic of China, under grant numbers NSTC 112-2221-E-032-035-MY2, NSTC 112-2221-E-507-007-MY2.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef] [PubMed]
2. Escobar-Naranjo, J.; Caiza, G.; Ayala, P.; Jordan, E.; Garcia, C.A.; Garcia, M.V. Autonomous Navigation of Robots: Optimization with DQN. *Appl. Sci.* **2023**, *13*, 7202. [CrossRef]
3. Van Hasselt, H.; Guez, A.; Silver, D. Deep reinforcement learning with double q-learning. In Proceedings of the AAAI Conference on Artificial Intelligence, Phoenix, AZ, USA, 12–17 February 2016; Volume 30, pp. 1–13. [CrossRef]
4. Konda, V.; Tsitsiklis, J. Actor-critic algorithms. *Adv. Neural Inf. Process. Syst.* **1999**, *12*, 1–7.
5. Park, K.W.; Kim, M.; Kim, J.S.; Park, J.H. Path Planning for Multi-Arm Manipulators Using Soft Actor-Critic Algorithm with Position Prediction of Moving Obstacles via LSTM. *Appl. Sci.* **2022**, *12*, 9837. [CrossRef]
6. Guo, H.; Ren, Z.; Lai, J.; Wu, Z.; Xie, S. Optimal navigation for AGVs: A soft actor–critic-based reinforcement learning approach with composite auxiliary rewards. *Eng. Appl. Artif. Intell.* **2023**, *124*, 106613. [CrossRef]
7. Liu, Z.; Qiu, C.; Zhang, Z. Sequence-to-Sequence Multi-Agent Reinforcement Learning for Multi-UAV Task Planning in 3D Dynamic Environment. *Appl. Sci.* **2022**, *12*, 12181. [CrossRef]
8. Yu, J.; LaValle, S. Structure and intractability of optimal multi-robot path planning on graphs. In Proceedings of the AAAI Conference on Artificial Intelligence, Bellevue, WA, USA, 14–18 July 2013; Volume 27, pp. 1443–1449. [CrossRef]
9. Sartoretti, G.; Kerr, J.; Shi, Y.; Wagner, G.; Kumar, T.S.; Koenig, S.; Choset, H. Primal: Pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robot. Autom. Lett.* **2019**, *4*, 2378–2385. [CrossRef]
10. Mnih, V.; Badia, A.P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In Proceedings of the International Conference on Machine Learning, New York City, NY, USA, 19–24 June 2016; pp. 1928–1937.
11. Riviere, B.; Hönig, W.; Yue, Y.; Chung, S.J. Glas: Global-to-local safe autonomy synthesis for multi-robot motion planning with end-to-end learning. *IEEE Robot. Autom. Lett.* **2020**, *5*, 4249–4256. [CrossRef]
12. Wang, B.; Liu, Z.; Li, Q.; Prorok, A. Mobile robot path planning in dynamic environments through globally guided reinforcement learning. *IEEE Robot. Autom. Lett.* **2020**, *5*, 6932–6939. [CrossRef]
13. Hart, P.E.; Nilsson, N.J.; Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.* **1968**, *4*, 100–107. [CrossRef]
14. Silver, D. Cooperative pathfinding. In Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Marina del Rey, CA, USA, 1–3 June 2005; Volume 1, pp. 117–122. [CrossRef]
15. Wagner, G.; Choset, H. Subdimensional expansion for multirobot path planning. *Artif. Intell.* **2015**, *219*, 1–24. [CrossRef]
16. Surynek, P.; Felner, A.; Stern, R.; Boyarski, E. Efficient sat approach to multi-agent path finding under the sum of costs objective. In Proceedings of the Twenty-Second European Conference on Artificial Intelligence, The Hague, The Netherlands, 29 August–2 September 2016; pp. 810–818. [CrossRef]
17. Lam, E.; Le Bodic, P.; Harabor, D.; Stuckey, P.J. Branch-and-cut-and-price for multi-agent path finding. *Comput. Oper. Res.* **2019**, *144*, 105809. [CrossRef]
18. Sharon, G.; Stern, R.; Felner, A.; Sturtevant, N.R. Conflict-based search for optimal multi-agent pathfinding. *Artif. Intell.* **2015**, *219*, 40–66. [CrossRef]
19. Ren, Z.; Rathinam, S.; Choset, H. A Conflict-Based Search Framework for Multiobjective Multiagent Path Finding. *IEEE Trans. Autom. Sci. Eng.* **2022**, *20*, 1262–1274. [CrossRef]
20. Gao, J.; Li, Y.; Li, X.; Yan, K.; Lin, K.; Wu, X. A review of graph-based multi-agent pathfinding solvers: From classical to beyond classical. *Knowl. Based Syst.* **2023**, *283*, 111121. [CrossRef]
21. Sharma, P.K.; Fernandez, R.; Zaroukian, E.; Dorothy, M.; Basak, A.; Asher, D.E. Survey of recent multi-agent reinforcement learning algorithms utilizing centralized training. In Proceedings of the Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications III, Online, 12–16 April 2021; Volume 11746, pp. 665–676. [CrossRef]
22. Lin, Q.; Ma, H. SACHA: Soft Actor-Critic with Heuristic-Based Attention for Partially Observable Multi-Agent Path Finding. *IEEE Robot. Autom. Lett.* **2023**, *8*, 5100–5107. [CrossRef]

23. Song, Z.; Zhang, R.; Cheng, X. HELSA: Hierarchical Reinforcement Learning with Spatiotemporal Abstraction for Large-Scale Multi-Agent Path Finding. In Proceedings of the 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Detroit, MI, USA, 1–5 October 2023; pp. 7318–7325. [CrossRef]
24. Ito, S.; Ohara, K.; Hoshi, Y.; Oya, H.; Nagai, S. A Robust Formation Control Strategy for Multi-Agent Systems with Uncertainties via Adaptive Gain Robust Controllers. *Int. J. Eng. Technol. Innov.* **2021**, *11*, 71–87. [CrossRef]
25. Willemsen, D.; Coppola, M.; de Croon, G.C. MAMBPO: Sample-efficient multi-robot reinforcement learning using learned world models. In Proceedings of the 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Prague, Czech Republic, 27 September–1 October 2021; pp. 5635–5640. [CrossRef]
26. Kozlica, R.; Wegenkittl, S.; Hiränder, S. Deep q-learning versus proximal policy optimization: Performance comparison in a material sorting task. In Proceedings of the 2023 IEEE 32nd International Symposium on Industrial Electronics (ISIE), Helsinki, Finland, 19–21 June 2023; pp. 1–6.
27. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. *arXiv* **2015**, arXiv:1509.02971.
28. Sutton, R.S.; McAllester, D.; Singh, S.; Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. *Adv. Neural Inf. Process. Syst.* **1999**, *12*, 1–7.
29. Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; Riedmiller, M. Deterministic policy gradient algorithms. In Proceedings of the International Conference on Machine Learning, Beijing, China, 21–26 June 2014; pp. 387–395.
30. Li, Y.; Aghvami, A.H.; Dong, D. Path planning for cellular-connected UAV: A DRL solution with quantum-inspired experience replay. *IEEE Trans. Wirel. Commun.* **2022**, *21*, 7897–7912. [CrossRef]
31. Gazebo Simulator. Available online: http://gazebosim.org/ (accessed on 20 June 2023).
32. Collins, J.; Chand, S.; Vanderkop, A.; Howard, D.A. Review of Physics Simulators for Robotic Applications. *IEEE Access* **2021**, *9*, 51416–51431. [CrossRef]
33. Lowe, R.; Wu, Y.I.; Tamar, A.; Harb, J.; Pieter Abbeel, O.; Mordatch, I. Multi-agent actor-critic for mixed cooperative-competitive environments. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 1–12.
34. Mordatch, I.; Abbeel, P. Emergence of grounded compositional language in multi-agent populations. In Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; Volume 32, p. 11492. [CrossRef]
35. Lin, P.H.; Lin, C.Y.; Hung, C.T.; Chen, J.J.; Liang, J.M. The Autonomous Shopping-Guide Robot in Cashier-Less Convenience Stores. *Proc. Eng. Technol. Innov.* **2020**, *14*, 9–15. [CrossRef]
36. Hollenstein, J.; Auddy, S.; Saveriano, M.; Renaudo, E.; Piater, J. Action noise in off-policy deep reinforcement learning: Impact on exploration and performance. *arXiv* **2022**, arXiv:2206.03787.
37. Huh, D.; Mohapatra, P. Multi-agent Reinforcement Learning: A Comprehensive Survey. *arXiv* **2023**, arXiv:2312.10256.
38. Goodfellow, I.; Bengio, Y.; Courville, A. Deep learning. In *Genetic Programming and Evolvable Machines*; The MIT Press: Cambridge, MA, USA, 2016; Volume 19, pp. 305–307. [CrossRef]
39. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*, 2nd ed.; MIT Press: Cambridge, MA, USA, 2018.