J. Shanghai Jiao Tong Univ. (Sci.), 2024, 29(4): 601-612
https://doi.org/10.1007/s12204-024-2732-1

# Multi-Agent Path Planning Method Based on Improved Deep Q-Network in Dynamic Environments

*LI Shuyi* (李舒逸), *LI Minzhe* (李旻哲), *JING Zhongliang** (敬忠良)
(School of Aeronautics and Astronautics, Shanghai Jiao Tong University, Shanghai 200240, China)



**Abstract:** The multi-agent path planning problem presents significant challenges in dynamic environments, primarily due to the ever-changing positions of obstacles and the complex interactions between agents' actions. These factors contribute to a tendency for the solution to converge slowly, and in some cases, diverge altogether. In addressing this issue, this paper introduces a novel approach utilizing a double dueling deep Q-network (D3QN), tailored for dynamic multi-agent environments. A novel reward function based on multi-agent positional constraints is designed, and a training strategy based on incremental learning is performed to achieve collaborative path planning of multiple agents. Moreover, the greedy and Boltzmann probability selection policy is introduced for action selection and avoiding convergence to local extremum. To match radar and image sensors, a convolutional neural network - long short-term memory (CNN-LSTM) architecture is constructed to extract the feature of multi-source measurement as the input of the D3QN. The algorithm's efficacy and reliability are validated in a simulated environment, utilizing robot operating system and Gazebo. The simulation results show that the proposed algorithm provides a real-time solution for path planning tasks in dynamic scenarios. In terms of the average success rate and accuracy, the proposed method is superior to other deep learning algorithms, and the convergence speed is also improved.
**Keywords:** multi-agent, path planning, deep reinforcement learning, deep Q-network
**CLC number:** TP242.6 **Document code:** A

## 0 Introduction

Multi-agent path planning in dynamic environments has attracted significant attention. This issue holds significant implications in many practical applications, such as collaborative navigation of autonomous vehicle fleets in complex environments, coordination of unmanned aerial vehicles in swarms, and cooperative operations of multiple robots[1-3]. In these scenarios, each agent needs to plan a safe and effective path in real time based on the dynamic changes in the environment and the actions of other agents. Although significant progress has been made in autonomous path planning of agents[4], it still remains a challenging task in dynamic environments. First, the dynamic nature of the environment means that the positions of obstacles may change at any moment, necessitating real-time path planning methods. Second, the positions and actions of multiple agents interact, which can cause planned paths to collide or become invalid. Furthermore, as the number of agents in the environment increases, these interactions become more intense. Due to these difficulties, existing multi-agent path planning methods in dynamic environments often face problems such as divergence and low convergence speed[5], implying that the path planning of agents could be unstable, and an effective path within an acceptable time frame might not be found, or the path found may not be optimal.

In the field of multi-agent path planning in dynamic environments, existing research mainly includes graph search-based methods[6], optimization-based methods[7], and probability model-based methods[8]. While these methods are widely applied to static path planning, they often perform poorly in dynamic environments and fail to effectively handle large amounts of real-time data and unknown factors[9]. In response, many scholars have conducted extensive research and achieved certain results, such as local path planning methods based on neural network (NN)[10], particle swarm optimization (PSO)[11], and reinforcement learning (RL)[12]. These methods, to some extent, have overcome the problem of insufficient prior environmental information and significantly improved the adaptability

**Received:** 2023-06-12 **Accepted:** 2023-10-12
**Foundation item:** the National Natural Science Foundation of China (Nos. 61673262 and 50779033), the National GF Basic Research Program (No. JCKY2021110B134), and the Fundamental Research Funds for the Central Universities
∗**E-mail:** zljing@sjtu.edu.cn

of agents in complex environments. They can obtain better paths, but they struggle to converge when facing high-dimensional, nonlinear problems such as multiple agents and continuous action spaces.

In recent path planning research, deep reinforcement learning (DRL) has received widespread attention and been widely used in path planning tasks[13]. RL enables agents to acquire optimal strategies via iterative interactions with the environment, founded on a trial-and-error principle[14]. This methodology aligns well with the capacities of deep neural networks, particularly in addressing non-linear and continuous challenges[15]. Consequently, there has been a notable increase in the application of DRL within navigation studies. Yan et al.[16] introduced a DRL technique predicated on situational maps, contributing to the field's understanding of spatial recognition in drone path planning. Ruan et al.[17] furthered this domain by implementing double dueling deep Q-network (D3QN) based on a convolutional residual network. Their primary objective was the extraction of depth information from images for autonomous vehicle navigation. Ruan et al.'s work not only emphasized the significance of image information but also suggested potential benefits of integrating varied data modalities. In the context of multi-agent navigation, DRL has proven effective in elucidating complex agent interactions. For instance, distributed DRL algorithms excel in managing intricate inter-agent relationships, facilitating collaborative navigation in complex scenarios[18]. However, despite some progress, there is still a lack of methods for multi-agent path planning in dynamic environments. Designing effective algorithms to handle multi-agent sensor data and interactions among agents, allowing multi-agents to effectively learn and make decisions in partially observable environments, remains a challenge.

In this paper, an improved D3QN-based multi-agent path planning method is proposed to face the dynamic scenario, where the state space dimension of the multiple agents is high and the interaction between multiple agents is prone to divergence. A new reward function involving the interaction of multiple robot positions and movements is designed, and a new action selection policy encompassing both the greedy and Boltzmann probability selection methods is adopted for the multi-agent control. This policy is designed to prevent convergence to local extremes and ensure a more holistic path selection process. In addition, a convolutional neural network - long short-term memory (CNN-LSTM) network is constructed so as to handle radar and image sensors data. This network extracts the features from multi-source measurements as the input for the dueling deep Q-network. The solutions are finally obtained based on incremental learning framework through model training process. The approach leverages the experiences gained in simpler environments for application in more complex scenarios.

The paper is organized as follows. Section 1 introduces the multi-agent model and D3QN algorithm framework. Section 2 introduces the details of the path planning method. Simulation and experimental results are discussed in Section 3. Finally, Section 4 concludes the paper and outlines our future plans.

## 1 Multi-Agent Model and Algorithm Framework

### 1.1 Multi-Agent Model

The multi-agent model proposed in this paper consists of multiple agents equipped with autonomous control capabilities. These agents can either converge on a shared target within the environment or independently pursue different targets. They engage in communication to establish a coordinated multi-agent system, as shown in Fig. 1(a). To establish a reference point for the coordinates, we designate the initial position of one agent as the origin, with the coordinates of other agents expressed relative to this origin. As depicted in Fig. 1(b), the single-agent model employs a differential motion model to acquire information in the simulation environment, including its own state $(x_t, y_t, v_t, \omega_t, d_t, \theta_t)$, where $(x_t, y_t)$ represents the position of the agent in the multi-agent coordinate system, $(v_t, \omega_t)$ are respectively the velocity and angular velocity of the agent at the current moment, $(d_t, \theta_t)$ are respectively the distance and angle between the agent and the target, and $t$ is the runtime of the agent.

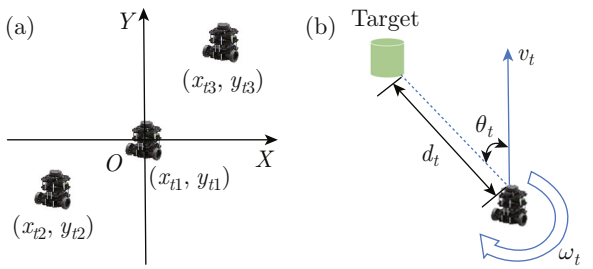


Fig. 1 Multi-agent model. (a) Multi-agent coordinate system; (b) Agent self-state.

### 1.2 Algorithm Framework

The entire algorithm framework can be viewed as an interactive process between the environment and a multi-agent system. The environment generates data from various sensors, and the agents process this multi-modal data using a path planning algorithm, which in turn produces actions that are subsequently executed within the environment, as depicted in Fig. 2. The data processing procedure involves several steps as follows.

(1) Data preprocessing. Transforming raw image, LiDAR, and self-status data into formats suitable for input into deep neural networks.

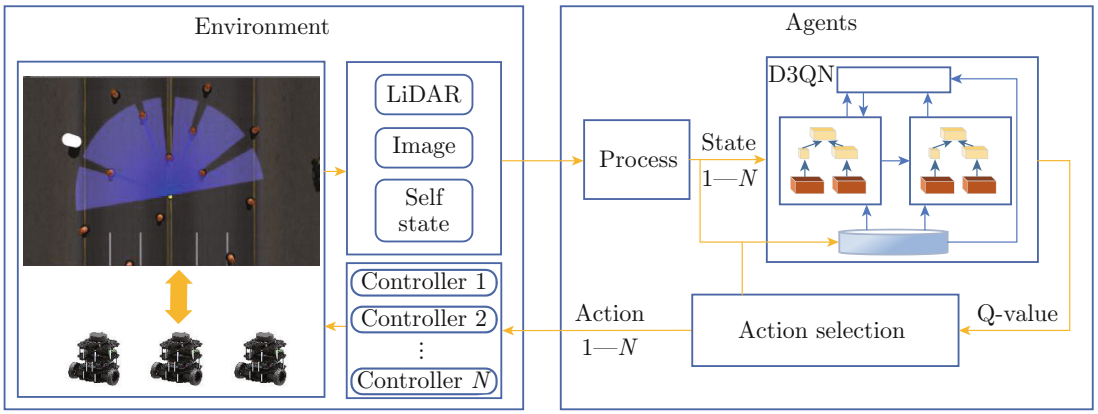


Fig. 2 Multi-agent path planning algorithm framework.

(2) CNN-LSTM network. The CNN and LSTM separately further process image information, LiDAR information, and self-status information.

(3) D3QN network. Action Q-values are computed through neural networks, and the neural network parameters are trained and updated using prioritized experience replay.

(4) Action selection strategy. The action Q-values outputted by the neural network are assessed by the action selection module, resulting in the determination of actions for each intelligent agent within the environment.

## 2 DRL-Based Path Planning

In this section, we will delve into deep reinforcement learning and provide specific details of the multi-agent path planning algorithm.

### 2.1 Deep Reinforcement Learning

Reinforcement learning defines the decision-maker as an agent, and everything outside the agent is defined as the environment. The goal of the agent is to maximize cumulative rewards, and it receives reward values as feedback signals for training through interaction with the environment. The interaction process between the agent and the environment can be modeled as a Markov decision process (MDP) composed of basic elements $S$, $A$, $R$ and $P$, where $S$ is the environmental state, $A$ is the action taken by the agent, $R$ is the acquired reward value, and $P$ is the state transition probability. The agent's policy is a mapping from the state space to the action space. When the state is $s_t \in S$, the agent takes action $a_t \in A$, then transitions to the next state $s_{t+1}$ according to the state transition probability $P$, and at the same time accepts feedback of the reward value $r_t \in R$ from the environment.

DRL combines deep learning with RL. In DRL, deep neural networks are used as models to estimate or optimize the policy or value function in RL problems. The learning process of DRL mainly includes the following steps.

(1) Initialization. Choose an initial policy and an initial value function. The agent operates in the environment based on its current policy. These operations may change the state of the environment and provide rewards for the agent.

(2) Learning. The agent employs observed states, performed actions, and received rewards to update its policy and value function. The parameters of the neural network are updated using the stochastic gradient descent method. Based on the new value function, the agent generates a new policy.

(3) Repetition. Repeat the steps of interaction, learning, and policy improvement until the policy and value function converge or a predetermined number of iterations is reached.

### 2.2 D3QN Network

In 2013, Mnih et al.[19] introduced the deep Q-network (DQN), which is founded on Q-learning and leverages a convolutional neural network to represent the action value function. This enables the DQN to learn control policies directly from environmental inputs.

The DQN algorithm employs a deep neural network to approximate the value function, providing Q-network samples through Q-learning to bring the Q-value closer to the target Q-value. A loss function is defined, and network parameters are updated through backpropagation using gradient descent.

The target Q-value is defined as

$$Q(s,a;w) = r_{t+1} + \gamma \max_{a'} Q(s',a';w^-), \qquad (1)$$

where $s$ denotes the current state within the environment, $a$ stands for the action taken by the agent in state $s$, $w$ refers to the weights of the neural network, $\gamma$ is the discount factor, $r_{t+1}$ is the reward value obtained at time $t+1$, and $w^-$ is the target network weight parameter. By starting from the state $s_t = s$, taking the first step to perform action $a_t = a$, and then executing

the strategy $\pi$, a sequence of rewards $(r_{t+1}, r_{t+2}, \cdots)$ as well as state $s'$, action $a'$ are obtained. Additionally, $\max_{a'} Q$ represents the action with the maximum Q-value. In this work, the agent's goal is to use the Q-value to obtain cumulative rewards from the environment and execute the best strategy through $\max_{a'} Q$.

The loss function for DQN is denoted as

$$L(w) = E[(r_{t+1} + \gamma \max_{a'} Q(s', a'; w^-) - Q(s, a; w))^2], \tag{2}$$

where $E[\cdot]$ represents the expected value. The expected value in this loss function averages over all possible state transitions.

When DQN calculates the target Q-value, each step obtains the maximum Q-value through the $\varepsilon$-greedy algorithm for solving, leading to an overestimation of the value function. To solve this problem, the double deep Q-network (DDQN) algorithm was proposed[20]. This algorithm uses a double network structure in the target Q function to separate the selection and calculation of the target Q value, using different Q-networks for each. This helps to reduce the overestimation of the value function. The target Q-value of DDQN is

$$Q(s, a; w) = r_{t+1} + \gamma Q(s', \arg\max_{a'} Q(s', a'; w); w^-), \tag{3}$$

and the loss function is denoted as

$$L(w) = E[r_{t+1} + \gamma Q(s', \arg\max_{a'} Q(s', a'; w); w^-) - Q(s, a; w))^2]. \tag{4}$$

Dueling DQN builds upon the DQN algorithm by training two sub-neural networks simultaneously during the training phase[21]. At the beginning of training, dueling DQN selects the action corresponding to the maximum Q-value, which can introduce some instability in the training process. To address this, the algorithm splits its architecture into two main components: sub-network $V(s; w; \beta)$ and sub-network $A(s, a; w, \alpha)$. Sub-network $V$ is designed to be action-independent and focuses on predicting state values with higher Q-values. On the other hand, sub-network $A$ is responsible for estimating relative advantages among different actions and corresponding action values.

The loss function of the dueling DQN method follows

$$L(w) = E[(r_{t+1} + \gamma Q(s', \mathrm{argmax}_{a'} Q(s', a'; w, \alpha, \beta) - Q(s, a; w))^2], \tag{5}$$

where $\alpha$ and $\beta$ respectively represent the network parameters of sub-network $A$ and sub-network $V$.

The D3QN algorithm[22] combines DQN, DDQN, and dueling network algorithms. Instead of directly outputting Q-values, the neural network outputs both the state value function and the advantage function. Research indicates that D3QN can achieve superior performance. The final output expression is as follows:

$$Q(s', a'; w, \alpha, \beta) = V(s; w; \beta) + A(s, a; w, \alpha) - \frac{1}{|A|} \sum_{a'} (s, a'; w, \alpha), \tag{6}$$

where $|A|$ is the action dimension.

### 2.3 Prioritized Replay Mechanism

The prioritized experience replay mechanism is a technique employed in DRL to improve the efficiency and stability of Q-learning algorithms, such as DQN. It involves the management of a replay buffer that stores a history of experiences, including state transitions, actions, rewards, and the resulting next states, encountered by an agent during its interactions within the environment.

In contrast to standard experience replay, prioritized experience replay assigns priority values to experiences based on their temporal-difference (TD) errors. These errors indicate the difference between the current Q-value estimate and the target Q-value for a given experience. Experiences with higher TD errors, signifying greater potential for learning, are assigned higher probabilities of being selected and replayed during the training process[23]. This prioritization mechanism helps the agent focus on experiences that are more informative and can lead to more effective learning.

### 2.4 Network Structure

To accurately estimate the state-action value function, we have designed the D3QN architecture as illustrated in Fig. 3. This architecture is tailored to the requirements of multi-modal sensor data obtained by multiple agents. The preprocessed multi-modal sensor data, comprising self-state information, LiDAR data, and image data, is input into a CNN-LSTM network, serving as the state representation $s$. In this work, the CNN component extracts relevant features from the image data, while the LSTM component captures timing features from LiDAR and self-state information. The outputs of the CNN-LSTM network are reshaped and concatenated to form a multi-modal fused feature vector, which is subsequently fed into the dueling network for further processing.

The dueling network consists of two sequences of fully connected (FC) layers: one for estimating the state value and the other for estimating the advantage for each action, respectively. In the FC1 layer, there are two FC sub-layers, each composed of 512 nodes. The FC2 layer is divided into two sub-layers, with 1 node and 9 nodes, respectively. In the FC3 layer, a single FC sub-layer consists of 9 nodes. All layers utilize the ReLU activation function. The network ultimately outputs the Q-values of all current candidate actions.

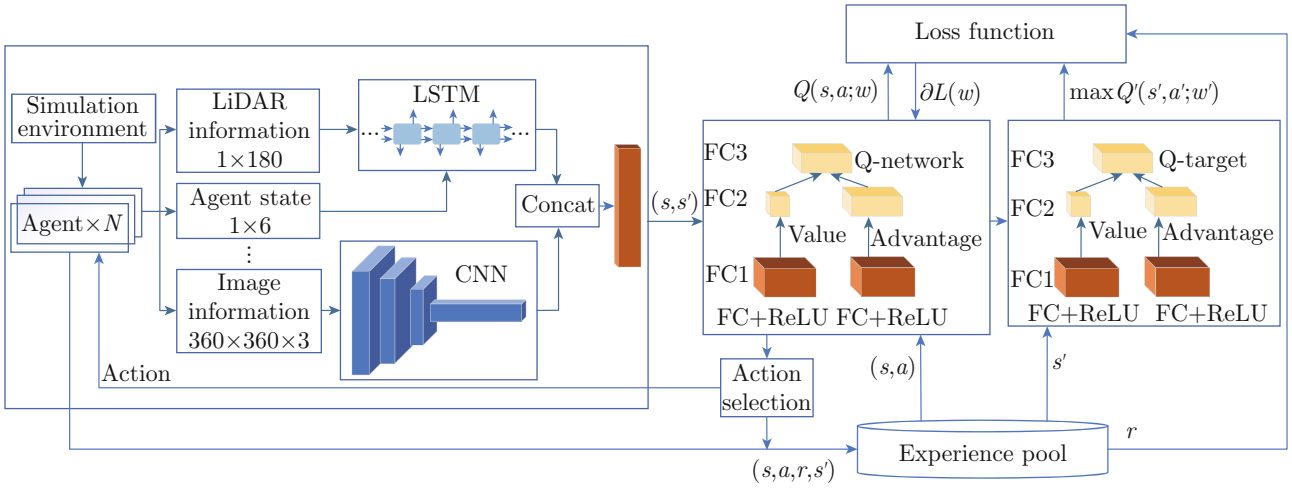


Fig. 3 Structure of D3QN.

Simultaneously, information about the agent's current state, image, LiDAR data, reward value, and more for the current step is stored in the experience pool, where prioritized experience replay assigns priority values to experiences based on their TD errors.

The network structure also introduces a target value network $Q'(s', a'; w')$ whose parameters are copied from the current network $Q(s, a; w)$ every $N$ steps. Since it is independent of the current network, the calculated target label $y(r, s') = \gamma \max Q'(s', a'; w')$ will be updated more stably.

Lastly, the D3QN algorithm fits the action value function through the neural network and selects actions based on the action value function outputted by the neural network. The D3QN parameters are given in Table 1. This design ensures the robustness and adaptability of the algorithm in complex multi-agent systems.

**Table 1 D3QN parameters**

| Parameter | Meaning | Value |
|---|---|---|
| Learning_rate | Learning rate | 0.000 1 |
| $\gamma$ | Decay factor | 0.99 |
| Action_num | Action space size | 9 |
| Num_train | Number of training | 50 000 |
| Max _num | Maximum number of episodes per training round | 500 |

## 2.5 Data Preprocessing

**Agent's Self-States Data** The self-states of an agent include its position $(x_t, y_t)$, current speed $v_t$, angular speed $\omega_t$, distance $d_t$ to the target location, and angle $\theta_t$ to the target location. In the simulation environment, these data are calculated from the outputs of the simulated inertial measurement unit (IMU) sensor and are ultimately combined into an array $(x_t, y_t, v_t, \omega_t, d_t, \theta_t)$.

**Image Data** In the simulation environment, the agent's simulated camera captures 800 × 600 color images. To facilitate convolution operations, the input image size usually needs the same width and height. In this experiment, we cropped and adjusted the input data to 360 × 360 RGB images, i.e., the dimensions are 360 × 360 × 3, as shown in Fig. 4.

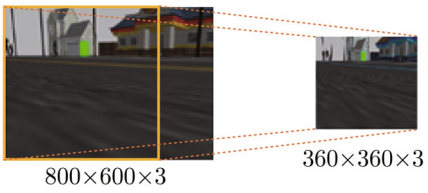


Fig. 4 Image preprocess.

**LiDAR Data** The LiDAR sensor is employed to measure the distance between the agent and the surrounding environment objects. In this simulation environment, the sensor provides a 1 × 180 array at each time step, representing the distances to obstacles at every 1° within a range of 180°, with the robot itself as the reference point. The detection range spans from 0 m to 15 m, and the data is collected at a frequency of 50 Hz.

## 2.6 CNN-LSTM Network

The CNN-LSTM network is utilized to process preprocessed multimodal data. In this setup, image data is fed into the CNN, while LiDAR data and self-state information are directed to the LSTM network. The outputs of both the CNN and LSTM networks are subsequently integrated into the control network, also known as the dueling network. The CNN-LSTM structure is shown in Fig. 5.

**CNN** We employ image data to discern whether the agent encounters a target or an obstacle. CNNs are utilized to extract important visual details like edges,

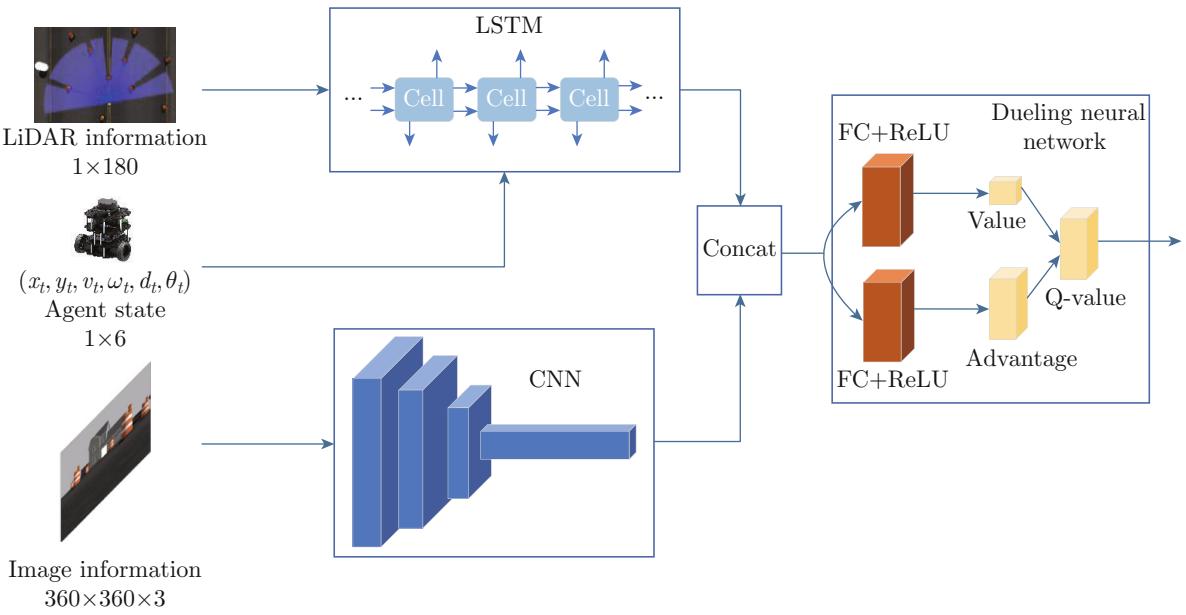


Fig. 5 CNN-LSTM network.

textures, and colors from RGB images. These features aid the robot in navigation and obstacle avoidance. We combine these visual cues with other data to make informed decisions about navigation and avoiding obstacles. Our design includes a three-layered CNN with the layers named CONV1, CONV2, and CONV3. The specific details for each layer are provided in Table 2. Each convolution layer employs the ReLU activation function.

**Table 2 CNN parameters**

| Convolution layer | Input dimension | Number of filter | Filter size | Activation function |
|---|---|---|---|---|
| CONV1 | $360 \times 360 \times 3$ | 32 | $8 \times 8 \times 3$ | ReLU |
| CONV2 | $90 \times 90 \times 32$ | 64 | $3 \times 3 \times 3$ | ReLU |
| CONV3 | $45 \times 45 \times 64$ | 64 | $1 \times 1 \times 3$ | ReLU |

**LSTM** The LSTM network specializes in learning from temporal features, utilizing both the robot's state information and LiDAR data. Specifically, we utilize a 4-dimensional state vector representing the robot's attributes like position, speed, and direction, combined with a 180-dimensional LiDAR data set, which details obstacle distances and positions. This mixed feature vector serves as the input to our LSTM network. As the LSTM network processes this data, it maintains a memory of past patterns, thus refining its predictive capabilities. Once processed, the LSTM network's outputs are merged with those from the CNN, guiding navigation decisions. Network parameters are outlined in Table 3.

**Table 3 LSTM parameters**

| Parameter | Meaning | Value |
|---|---|---|
| Num_units | Number of LSTM cells per time step | 512 |
| Activation | Nonlinear activation function, ReLU | |
| Forget_bias | Forgetting gate bias | 1.0 |

## 2.7 Reward Function Setting

The setting of the reward function is shown in Table 4. In our multi-agent path planning algorithm, we utilize a mobile robot as the validation platform. In the simulation environment, mobile robots can communicate to acquire distances between each other. The following scenarios are given positive rewards: ① when the distance between the robot and the corresponding target point $d$ decreases; ② when the target point is reached. The following scenarios are given negative rewards: ① when the robot moves away from the corresponding target point; ② when the distance to an

**Table 4 Reward function**

| Situation | Reward |
|---|---|
| $d < 0.5\,\mathrm{m}$ | $+20$ |
| Close to target ($d_{\mathrm{last}} > d$) | $+0.15(d_{\mathrm{last}} - d)$ |
| Away from the target ($d_{\mathrm{last}} < d$) | $+0.15(d_{\mathrm{last}} - d)$ |
| Close to obstacles ($l_{i_{\mathrm{last}}} > l_i | l_i \leqslant 1.5\,\mathrm{m}$) | $-0.01(l_{i_{\mathrm{last}}} - l_i)$ |
| $l_i < 0.5\,\mathrm{m}$ | $-3$ |
| $l_{\mathrm{a}} < 0.5\,\mathrm{m}$ | $-3$ |
| Take action | $-0.01$ |

obstacle $l_i$ is less than 0.5 m, indicating a collision has occurred; ③ when the distance to an obstacle $l_i$ is within a certain range and continues to decrease, leaving space to evade dynamic obstacles; ④ when the distance to another agent $l_{\mathrm{a}}$ is within a certain range, affecting the other agent's work.

In Table 4, $l_i$ is the distance between the agent and the $i$th obstacle at the current moment, and $l_{i_{\mathrm{last}}}$ is the distance between the agent and the $i$th obstacle at the previous moment; $d$ and $d_{\mathrm{last}}$ are the distances between the agent and the corresponding target and the target point between the current time and the previous time, respectively. Additionally, to incentivize the agent to minimize the number of actions taken to reach the endpoint, a negative reward is assigned for each new action it takes. The final reward value is computed as the average reward value for each agent.

### 2.8 Action Space Design

In the simulation environment of this paper, we employ differential-drive mobile robots as the agents. These robots have the capability to perform various movements by controlling their linear velocity and angular velocity. The design of the action space is illustrated in Table 5, comprising nine distinct actions, each executed within a single time step.

**Table 5 Action space**

| Action | $v_t/(\mathrm{m}\cdot\mathrm{s}^{-1})$ | $\omega_t/(\mathrm{rad}\cdot\mathrm{s}^{-1})$ |
|---|---|---|
| 0 | 1.0 | 0.0 |
| 1 | 1.0 | −0.5 |
| 2 | 1.0 | 0.5 |
| 3 | 0.5 | 0.0 |
| 4 | 0.5 | −0.5 |
| 5 | 0.5 | 0.5 |
| 6 | 0.0 | 0.0 |
| 7 | 0.0 | −0.5 |
| 8 | 0.0 | 0.5 |

### 2.9 Action Selection Strategy

If an agent constantly chooses actions based on the maximum Q-value, it may encounter challenges in exploring unfamiliar areas and discovering more efficient paths. To address the limitations, this paper combines the $\varepsilon$-greedy method with Boltzmann probabilities. This amalgamation enables the agent to introduce more randomness into its exploration during the training process and prevents it from becoming trapped in local optima[24]. The action selection strategy is as follows:

Action selection strategy
**Input**:
$Q$: Expected return Q value for each action;
episode_num: Total number of episodes in the current training;
Max_num: Maximum number of steps in an episode.
**Output**:
Action $a_i$.
**Action selection**:
Generate $p$ randomly, $p \in (0,1)$
If episode_num < Max_num/2:
$a = \mathrm{Boltzman}(A)$ // $A$ represents the action space
Else:

$$a = \begin{cases} \mathrm{rand}(A), & p < \varepsilon \\ \mathrm{argmax}\, Q(s,a_i), & p \geqslant \varepsilon \end{cases}$$

Endif

During the early stages of training, the focus is placed on exploration, which relies more on a Boltzmann probability selection strategy, as shown in

$$P(a_k|s) = \frac{\exp\left[-\dfrac{Q\{s,a_k\}}{\kappa_{\mathrm{B}}T}\right]}{\displaystyle\sum_{k=1}^{K}\exp\left[-\frac{Q(s,a_k)}{\kappa_{\mathrm{B}}T}\right]}, \quad k=1,2,\cdots,K, \tag{7}$$

where $P(a_k|s)$ signifies the probability of the agent selecting action $a_k$ in state $s$, $\kappa_{\mathrm{B}}$ stands for the Boltzmann constant, and $T$ represents the temperature.

In the later stages, once the model has explored a sufficient amount of the state space and learned effective strategies, the focus shifts to exploiting these strategies. At this time, the action selection policy uses a greedy algorithm to choose the action currently evaluated as the best. The epsilon value in the epsilon-greedy algorithm is set to a very low value, such as 0.1, to ensure that the optimal action is chosen in most cases. The $\varepsilon$ value initiates at 1.0 and gradually decreases, but it does not go below 0.1:

$$\varepsilon = \begin{cases} 1.0\times 0.99^n, & \varepsilon > 0.1 \\ 0.1, & \varepsilon \leqslant 0.1 \end{cases}, \tag{8}$$

where $n$ denotes episode_num.

## 3 Simulation

In this study, we have established an algorithm simulation validation environment using robot operating system (ROS) and Gazebo. ROS, a premier robot software framework, seamlessly integrates with Gazebo, a simulator renowned for its physically realistic simulations and customizable settings. Gazebo ensures that algorithm performance closely mirrors reality and provides robust and repeatable validation, reducing the risks associated with initial tests on real hardware. Developers have the flexibility to adjust physical parameters in Gazebo, making use of both pre-existing

and custom robotic components to create realistic environmental simulations[25]. During the model training phase, agents gather data, including LiDAR, visual images, and self-states information, which is processed and fed to the neural network. The primary challenge for the robotic agents during training is navigating around obstacles and reaching their designated targets.

### 3.1 Simulation Environments

To attain improved training outcomes, we have configured four distinct simulation environments for multiple rounds of training and introduced a scene resembling a real-world setting to evaluate the algorithm's effectiveness.

**Empty World** In this environment as shown in Fig. 6(a), we have set up an empty world with no obstacles, featuring only three agents navigating within it. The primary objective here is to train the agents in recognizing target objects and successfully reaching specified targets.

**Static Obstacle World** Building upon the empty world, we have introduced 14 static roadblocks as obstacles, as depicted in Fig. 6(b). When the LiDAR feedback data indicates a minimum distance to an obstacle ($l_i < 0.5\,\mathrm{m}$), it is considered a collision. The mobile robot undergoes multiple rounds of training in this static obstacle world to develop the ability to recognize and approach targets while avoiding obstacles.

**Dynamic Obstacle World** Expanding upon the static obstacle world, we have created a dynamic obstacle world by adding roadblocks with different actions and motion trajectories, as shown in Fig. 6(c). This dynamic environment includes mobile robots, static obstacles (10 static barriers) and dynamic obstacles (7 moving barriers).

**Pedestrian World** To assess the trained model's generalization capabilities, we have replaced the types of obstacles and constructed a pedestrian world based on the previous environments. In this scenario, 15 pedestrian models with collision volumes continuously traverse the environment, as shown in Fig. 6(d).

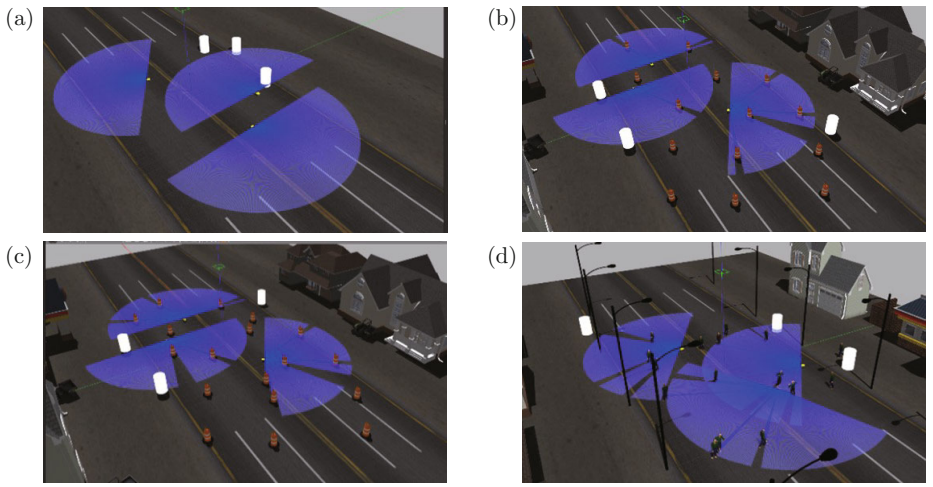


Fig. 6 Simulation environments. (a) Empty world; (b) Static obstacle world; (c) Dynamic obstacle world; (d) Pedestrian world.

In the simulation environment, the trajectories of dynamic roadblocks and pedestrians (obstacles) are defined through files. Each obstacle follows a specific back-and-forth movement pattern, with each having its own unique trajectory. These movement patterns can be adjusted before the environment begins, including parameters like movement speed and starting and ending positions of obstacles.

During training, the starting and target positions of the agents are randomly set in each iteration, while the movement patterns of obstacles within the environment remain consistent. During testing, the movement patterns of obstacles in the dynamic obstacle environment remain unchanged, and the pedestrian environment represents a novel dynamic obstacle environment utilized to assess the model's performance.

### 3.2 Training Process

The path planning method for multiple agents designed based on DRL needs to train neural network parameters so that the network can accurately estimate state-action values, enabling the robot to successfully carry out navigation and obstacle avoidance tasks. The training environment is set as a square area of $30\,\mathrm{m} \times 30\,\mathrm{m}$. During training, the environment is initialized as follows:

$$\left.\begin{aligned} &X_{\mathrm{ra}} \in [-15.0, 15.0]\,\mathrm{m} \\ &Y_{\mathrm{ra}} \in [-15.0, 15.0]\,\mathrm{m} \\ &X_a \in [-15.0, 15.0]\,\mathrm{m} \\ &X_i \in [-15.0, 15.0]\,\mathrm{m} \\ &Y_i \in [-10.0, 10.0]\,\mathrm{m} \\ &Y_a \in [-20.0, -10.0) \cup (10.0, 20.0]\,\mathrm{m} \end{aligned}\right\}, \tag{9}$$

where $X_{ra}$ and $Y_{ra}$ represent the initial position of agent $a$, $X_a$ and $Y_a$ represent the initial position of agent $a$'s corresponding targets, and $X_i$ and $Y_i$ represent the position of the $i$th obstacle.

At the start of each training round, the robot's starting position and target position are randomly assigned within the specified range, ensuring that the final training results can handle different situations. During each training round, if agents encounter no obstacles, they continue exploring within the maximum run times. If they encounter an obstacle, the environment resets, and the agents recommence their exploration from the beginning. As the number of explorations accumulates, the agents progressively develop better route planning skills to avoid obstacles and reach the target point.

The training process incorporates the concept of incremental learning, allowing for the continuous evolution of the trained model. This procedure is executed in three phases:

**Round One: Target Recognition and Navigation** The first round is trained in the empty world. The primary goal is to train the agents to recognize target objects and reach specified targets accurately. This environment provides a foundation for understanding basic navigation.

**Round Two: Static Obstacle Avoidance** The second round of training builds upon the model obtained in the first round, aiming to enhance the agent's obstacle avoidance capabilities. Agents explore in static obstacles world, striving to reach their destinations while avoiding all obstacles. In this round of training, since the model from the first round already possesses the ability to find the target, the initial actions do not need to be entirely random and the initial value of $\varepsilon$ in Eq. (8) remains at 0.8.

**Round Three: Dynamic Obstacle Handling** During the third training round, we transition to the dynamic obstacle world. Agents have already acquired navigation and obstacle avoidance skills from the previous rounds, and this round presents them with the challenge of dynamic obstacles. In order to accommodate this transition, the initial value of $\varepsilon$ in Eq. (8) was adjusted to 0.6, allowing for a more dynamic exploration strategy.

### 3.3 Result Analysis

To evaluate the algorithm's effectiveness, we conducted two sets of experiments. In Experiment 1, we compared the performance of DQN, DDQN and D3QN in four different scenarios. All of these groups utilized a CNN-LSTM network along with a new reward function and action selection strategy. In Experiment 2, three control groups were used: ① D3QN with a new reward function and new action selection strategy (D3QN); ② D3QN without the new action selection strategy (D3QN2); ③ D3QN without reward and the new action selection strategy (D3QN3).

**Experiment 1** To assess the model's reliability and adaptability, we tested the model in four environments. Each agent autonomously selected actions based on its own policy network. We conducted 20 rounds of testing for each environment, and the results are presented in Table 6. The table displays the average success rate and average reward achieved by three agents over 20 testing rounds for each respective model. Environment 0 represents the empty world, Environment 1 features static obstacles, Environment 2 involves dynamic obstacles, and Environment 3 simulates a pedestrian scenario.

**Table 6 Average success rate and average reward for DQN, DDQN and D3QN**

| Environment | Algorithm | Average success rate/% | Average reward |
|---|---|---|---|
| 0 | DQN | 95.00 | 19.95 |
| 0 | DDQN | 96.67 | 20.02 |
| 0 | D3QN | 96.67 | 20.10 |
| 1 | DQN | 85.00 | 19.52 |
| 1 | DDQN | 88.33 | 19.85 |
| 1 | D3QN | 93.33 | 20.01 |
| 2 | DQN | 80.00 | 19.57 |
| 2 | DDQN | 86.67 | 19.68 |
| 2 | D3QN | 90.00 | 19.98 |
| 3 | DQN | 75.00 | 19.38 |
| 3 | DDQN | 81.67 | 19.41 |
| 3 | D3QN | 88.33 | 19.67 |

In the static obstacle world, as shown in Fig. 7(a), the DQN algorithm converged around the 350th training episode, the DDQN algorithm converged around the 200th episode, and the D3QN algorithm converged around the 180th episode. In the initial phase, the DDQN algorithm exhibited faster convergence, but D3QN eventually surpassed it. This improvement is attributed to the D3QN algorithm's ability to learn the Q function more effectively, thanks to the separation of the single-stream Q-network.

In the dynamic world as shown in Fig. 7(b), the DQN algorithm converged around the 400th episode, the DDQN algorithm converged around the 330th episode, and the D3QN algorithm converged around the 180th episode. By comparing the second round of training to the first, it is evident that the D3QN algorithm has better adaptability and convergence speed in complex environments.

The main issue with DQN is its instability in estimating Q-values, which can result in estimation biases and convergence difficulties. This instability arises due to the updates of neural network parameters during the training process. In DDQN, the two Q-networks independently update their weights, reducing the risk

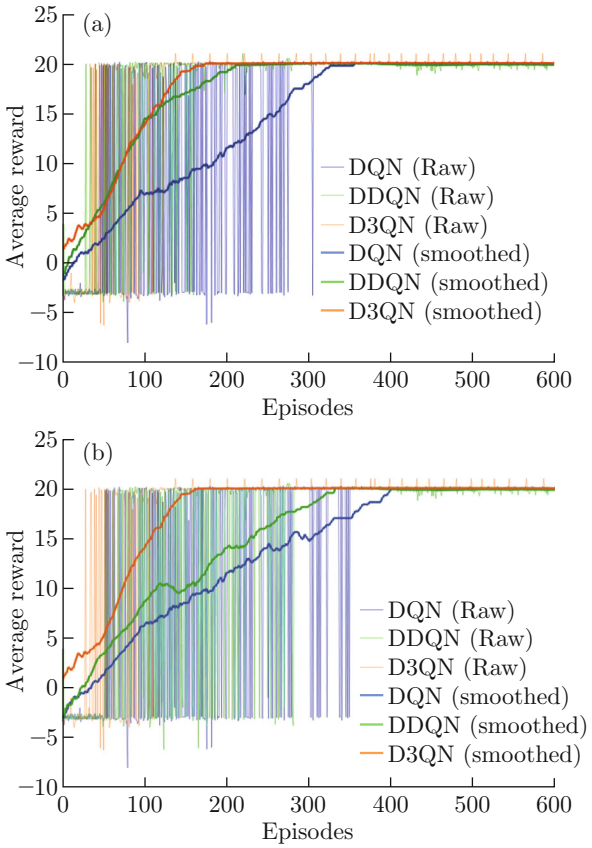


Fig. 7 Average reward of DQN, DDQN and D3QN. (a) Static world training results; (b) Dynamic world training results. "Raw" denotes the unprocessed distribution of average reward values during the training process, while "smoothed" represents the data after applying a smoothing technique.

of overestimating action values, thereby enhancing stability and convergence speed. However, some redundancy in learning still exists as both networks share certain underlying feature representations. The separate architecture of D3QN further mitigates this redundancy in learning. Moreover, due to the decomposition of Q-values, the network becomes more sensitive to variations in different actions. This heightened sensitivity allows for better dampening of reward fluctuations and avoidance of unstable learning processes, ultimately leading to improved convergence speed and stability of the network.

Table 6 presents a comparison of the average success rates and average reward values of the three algorithms tested in four different environments. The average success rate indicates the probability of an agent successfully navigating obstacles and reaching the target point in 20 trials of each round. The average reward value represents the mean reward obtained by each agent upon successfully reaching the target, excluding rewards obtained during failures. Given that the reward function was designed to account for the influence of the distance to the target point at each moment and the number of movements on the final reward, the magnitude of the average reward value can indicate the accuracy of the path planning.

From Table 6, it is evident that as the complexity of the environment increases, the average success rates of all three algorithms steadily decrease. However, when dealing with intricate dynamic environments, the D3QN algorithm exhibits a notable advantage. The success rate of the DQN algorithm drops significantly after the environment becomes more complex, indicating its poor stability. In contrast, the D3QN algorithm achieves the highest accuracy. This is attributed to the fact that D3QN decomposes the value function of each action into a state-value function and an advantage function. This decomposition helps to more accurately capture state values and the effects of different actions, thereby enhancing the network's ability to acquire useful information.

Furthermore, when testing models trained in dynamic world environments within pedestrian environments, the D3QN algorithm maintains satisfactory accuracy and average success rates. This phenomenon underscores the stability and generalization prowess inherent in our algorithm.

**Experiment 2** In Experiment 2, we aimed to validate the effectiveness of the new reward function and action selection strategy. For this purpose, we established three control groups. The first group remained consistent with the D3QN from Experiment 1. The second group employed a greedy algorithm for action selection, while the third group omitted a term that considered other agents in the reward function and simultaneously adopted the greedy algorithm for action selection.

Action selection strategy for D3QN2 and D3QN3
Generate $p$ randomly, $p \in (0, 1)$
If $p < \varepsilon$:
    $a = \text{rand}(A)$
Else:
    $a = \max Q(s, a)$
Endif

Experiment 2 followed the same training protocol as Experiment 1. The models were trained for three rounds in Environments 0, 1, and 2, and subsequently tested in Environments 0, 1, 2, and 3. The outcomes are presented in Fig. 8 and Table 7.

Figure 8 illustrates the results of training D3QN, D3QN2, and D3QN3 in a dynamic world. It can be observed that even after multiple training iterations, D3QN is still prone to collisions. This is attributed

**Table 7 Average success rate and average reward for D3QN, D3QN2 and D3QN3**

| Environment | Algorithm | Average success rate/% | Average reward |
|---|---|---|---|
| 0 | D3QN | 96.67 | 20.10 |
| 0 | D3QN2 | 96.67 | 20.04 |
| 0 | D3QN3 | 93.33 | 19.87 |
| 1 | D3QN | 93.33 | 20.01 |
| 1 | D3QN2 | 90.00 | 19.97 |
| 1 | D3QN3 | 86.67 | 19.54 |
| 2 | D3QN | 90.00 | 19.98 |
| 2 | D3QN2 | 88.33 | 19.90 |
| 2 | D3QN3 | 85.00 | 19.35 |
| 3 | D3QN | 88.33 | 19.67 |
| 3 | D3QN2 | 88.33 | 19.65 |
| 3 | D3QN3 | 81.67 | 19.23 |

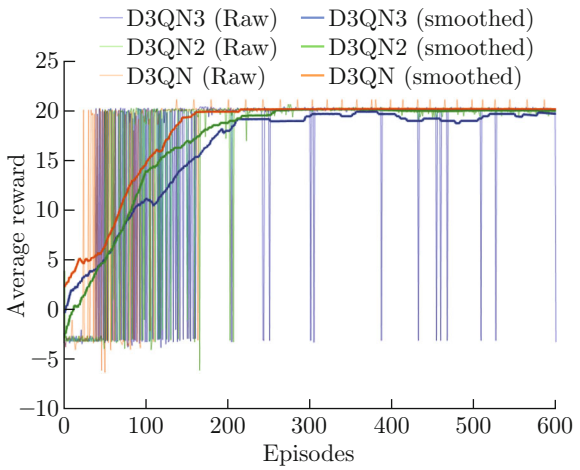


Fig. 8 Average reward of D3QN, D3QN2 and D3QN3.

to the fact that the agents did not recognize each other as obstacles during the earlier training stages, resulting in a possibility of collisions and subsequent task failures. In the initial phases, D3QN employs the Boltzmann-Greedy algorithm for action selection, which introduces randomness to achieve a better balance between exploration and exploitation. This renders D3QN more advantageous in the initial exploration phase and helps prevent falling into local optima.

Table 7 reveals that D3QN3 exhibits the lowest average success rate and precision. Following a refinement of the reward function, D3QN2 shows a notable improvement in average success rate. Though enhancing action selection strategy in D3QN yields less conspicuous improvements in success rate and precision, it primarily serves to enhance the convergence of the algorithm.

## 4 Conclusion

This work proposes a multi-agent path planning algorithm based on the D3QN. A new reward function is designed based on multi-agent location constraints to address the mutual influences among agents. A greedy policy with Boltzmann probability is introduced for action selection. In addition, we constructed a CNN-LSTM network to extract features from multi-modal sensor measurements. Finally, the proposed algorithm is validated in both static and dynamic simulation environments, and the simulations results show that it outperforms previous methods in terms of accuracy and convergence speed.

**Conflict of Interest** The authors declare that they have no conflict of interest.

## References

[1] ARADI S. Survey of deep reinforcement learning for motion planning of autonomous vehicles [J]. *IEEE Transactions on Intelligent Transportation Systems*, 2022, **23**(2): 740-759.

[2] ZHOU W H, LIU Z H, LI J, et al. Multi-target tracking for unmanned aerial vehicle swarms using deep reinforcement learning [J]. *Neurocomputing*, 2021, **466**: 285-297.

[3] HAN R H, CHEN S D, HAO Q. Cooperative multi-robot navigation in dynamic environment with deep reinforcement learning [C]//*2020 IEEE International Conference on Robotics and Automation*. Paris: IEEE, 2020: 448-454.

[4] SÁNCHEZ-IBÁÑEZ J R, PÉREZ-DEL-PULGAR C J, GARCÍA-CEREZO A. Path planning for autonomous mobile robots: A review [J]. *Sensors*, 2021, **21**(23): 7898.

[5] CHAE S W, SEO Y W, LEE K C. Task difficulty and team diversity on team creativity: Multi-agent simulation approach [J]. *Computers in Human Behavior*, 2015, **42**: 83-92.

[6] MA H. Graph-based multi-robot path finding and planning [J]. *Current Robotics Reports*, 2022, **3**(3): 77-84.

[7] POUDEL S, ARAFAT M Y, MOH S. Bio-inspired optimization-based path planning algorithms in unmanned aerial vehicles: A survey [J]. *Sensors*, 2023, **23**(6): 3051.

[8] HUANG J, JI Z H, XIAO S, et al. Multi-agent vehicle formation control based on mpc and particle swarm optimization algorithm [C]//*2022 IEEE 6th Information Technology and Mechatronics Engineering Conference*. Chongqing: IEEE, 2022: 288-292.

[9] GAO J L, YE W J, GUO J, et al. Deep reinforcement learning for indoor mobile robot path planning [J]. *Sensors*, 2020, **20**(19): 5493.

[10] PATLE B K, BABU L G, PANDEY A, et al. A review: On path planning strategies for navigation of mobile robot [J]. *Defence Technology*, 2019, **15**(4): 582-606.

[11] SALAMAT B, TONELLO A M. A modelling approach to generate representative UAV trajectories using PSO [C]//*2019 27th European Signal Processing Conference*. A Coruna: IEEE, 2019: 1-5.

[12] BATTOCLETTI G, URBAN R, GODIO S, et al. RL-based path planning for autonomous aerial vehicles in unknown environments [C]//*AIAA AVIATION 2021 FORUM*. Online: AIAA, 2021: 3016.

[13] ZHU K, ZHANG T. Deep reinforcement learning based mobile robot navigation: A review [J]. *Tsinghua Science and Technology*, 2021, **26**(5): 674-691.

[14] GARAFFA L C, BASSO M, KONZEN A A, et al. Reinforcement learning for mobile robotics exploration: A survey [J]. *IEEE Transactions on Neural Networks and Learning Systems*, 2023, **34**(8): 3796-3810.

[15] LIU F, CHEN C, LI Z H, et al. Research on path planning of robot based on deep reinforcement learning [C]//*2020 39th Chinese Control Conference*. Shenyang: IEEE, 2020: 3730-3734.

[16] YAN C, XIANG X J, WANG C. Towards real-time path planning through deep reinforcement learning for a UAV in dynamic environments [J]. *Journal of Intelligent & Robotic Systems*, 2020, **98**(2): 297-309.

[17] RUAN X G, LIN C L, HUANG J, et al. Obstacle avoidance navigation method for robot based on deep reinforcement learning [C]//*2022 IEEE 6th Information Technology and Mechatronics Engineering Conference*. Chongqing: IEEE, 2022: 1633-1637.

[18] HU Z W, CONG S C, SONG T K, et al. AirScope: Mobile robots-assisted cooperative indoor air quality sensing by distributed deep reinforcement learning [J]. *IEEE Internet of Things Journal*, 2020, **7**(9): 9189-9200.

[19] MNIH V, KAVUKCUOGLU K, SILVER D, et al. Playing Atari with deep reinforcement learning [DB/OL]. (2013-12-19). http://arxiv.org/abs/1312.5602

[20] VAN HASSELT H, GUEZ A, SILVER D. Deep reinforcement learning with double Q-Learning [C]//*Thirtieth AAAI Conference on Artificial Intelligence*. Phoenix: ACM, 2016: 2094-2100.

[21] SEWAK M. Deep Q Network (DQN), Double DQN, and Dueling DQN: A step towards general artificial intelligence [M]//Deep reinforcement learning: Frontiers of artificial intelligence. Singapore: Springer, 2019: 95-108.

[22] PENG B Y, SUN Q, LI S E, et al. End-to-end autonomous driving through dueling double deep Q-network [J]. *Automotive Innovation*, 2021, **4**(3): 328-337.

[23] SCHAUL T, QUAN J, ANTONOGLOU I, et al. Prioritized experience replay [DB/OL]. (2015-11-18). http://arxiv.org/abs/1511.05952

[24] CHAUHAN R, GHANSHALA K K, JOSHI R C. Convolutional neural network (CNN) for image detection and recognition [C]//*2018 First International Conference on Secure Cyber Computing and Communication*. Jalandhar: IEEE, 2018: 278-282.

[25] MEGALINGAM R K, R A, HEMATEJAANIRUDHBABU D, et al. Implementation of a Person Following Robot in ROS-gazebo platform [C]//*2022 International Conference for Advancement in Technology*. Goa: IEEE, 2022: 1-5.