Autonomous Exploration of mobile robots in unknown environments

ABSTRACT

The automatic exploration of mobile robots for unknown environments is a hot technology in the field of robots, which can be applied to military reconnaissance, rescue search, resource exploration and other scientific research and commercial fields, and has attracted wide attention in recent years. The basic requirement of mobile robot exploration in unknown environment is to complete the task of constructing unknown environment map without any prior knowledge. Traditional exploration algorithms rely heavily on map information and lack of adaptability to different environments. The existing learning-based control methods have problems such as low learning efficiency, serious dependence on robot models, and are difficult to transfer to the actual environment. In this paper, a decision algorithm based on deep reinforcement learning is proposed. The algorithm uses deep learning method to effectively extract environmental features, and automatically updates the strategy through interaction with the environment. According to the current pose and environment information of mobile robot, the algorithm can output the next target pose of mobile robot. Experiments show that the algorithm has good learning efficiency and adaptability in different environments. Based on the problem of unknown environment exploration in the field of mobile robots, the main work of this paper is as follows:

(1)Real-time mapping of mobile robot is realized based on Karto SLAM algorithm, and path planning of mobile robot is realized based on A* search algorithm. The mapping module generates a two-dimensional grid map according to the characteristics of the surrounding environment, which is input to the decision module. The navigation module conducts path planning according to the optimal target pose calculated by the decision module, and applies A* global path planning and TEB local path planning algorithm to make the mobile robot reach the target pose.

(2)The target point decision of mobile robot is realized based on the improved DQN algorithm. In this paper, convolution neural networks (CNNs) are used to extract environmental features. The three improved schemes of DQN are combined to obtain the D3QN PER algorithm and applied to the unknown environment exploration of mobile robots. The decision module calculates the Q value according to the environment information and the pose of the mobile robot, makes the optimal decision and outputs the next target pose to the navigation module.

(3)The simulation environment based on ROS and Gazebo verifies the effectiveness of the algorithm. Firstly, this paper completes the modeling of different map environments and mobile robots in Gazebo. Secondly, multiple target point decision models are trained based on different reward functions, different training map environments, different deep learning models and different model parameters. At last, under different test maps and different starting positions, this paper verifies the environmental exploration rate, environmental exploration efficiency, average path length and environmental adaptability of each model.

Key words： Autonomous exploration, Optimal decision, Deep reinforcement learning, CNN (Convolutional neural network), DQN (Deep Q-network)

Table of Contents

Chapter 3 4

Exploration System in Unknown Environments Based on Improved DQN 4

- 3.1 Introduction 4
- 3.2 Mapping module based on Karto SLAM 5
- 3.3 Decision Module Based on Improved DQN Algorithm 8
- 3.4 Navigation Module Based on A* Search Algorithm 18
- 3.5 Summary of This Chapter 20

Chapter 4 22

Experimental Analysis 22

- 4.1 Introduction 22
- 4.2 Simulation Experiment Environment Based on ROS and Gazebo 22
- 4.3 Simulation Experiment Workflow 26
- 4.4 Model Training and Testing 28
- 4.4 Conclusion of This Chapter. 37

Chapter 5 38

Conclusion and Future Outlook 38

- 5.1 Conclusion 38
- 5.2 Future Outlook 38

References 39

Chapter 3

Exploration System in Unknown Environments Based on Improved DQN

3.1 Introduction

Currently, some autonomous exploration research based on deep reinforcement learning takes the sensor data of mobile robots and their surrounding environment as input for deep neural networks. These networks directly output control commands for the robots, achieving end-to-end control. The mobile robot receives data from its sensors and can automatically perform actions. While end-to-end approaches are straightforward, they have significant drawbacks.

On one hand, end-to-end control methods do not leverage existing motion control techniques in robotics. Algorithms require extensive training to reach a sufficient level of expertise and performance, meaning they take a long time to converge. On the other hand, due to substantial differences between real and simulated environments, data obtained by sensors in real environments can be vastly different from those in simulated environments. The generalization of deep neural networks in real environments cannot be guaranteed. Furthermore, end-to-end control methods heavily depend on the robot model used during training. Changing the robot model significantly impairs algorithm performance, necessitating model retraining. Therefore, end-to-end deep reinforcement learning methods are not well-suited to addressing autonomous exploration problems in real environments.

Inspired by the work of H. Li et al.[31], which decomposed autonomous exploration into mapping, decision-making, and navigation modules, deep reinforcement learning algorithms are responsible only for decision-making based on the surrounding environment, outputting the next target point's location. The control commands for the mobile robot are generated by the navigation module. This approach greatly reduces the reliance on the mobile robot's model and sensors, effectively utilizing existing mapping and navigation algorithms to simplify the problem. The modular approach facilitates algorithm modification, transfer, and application.

Motivated by this research, this paper constructs an autonomous exploration framework based on mapping, decision-making, and navigation modules, using classical robotics techniques for mobile robot localization, environmental map construction, and path planning. Decision-making is accomplished using a deep reinforcement learning method based on improved DQN. The workflow for solving autonomous exploration problems in this paper is shown in Figure 3.1. The proposed autonomous exploration method is easy to train, can be deployed effectively in real environments, and exhibits strong adaptability to different environments compared to traditional methods.

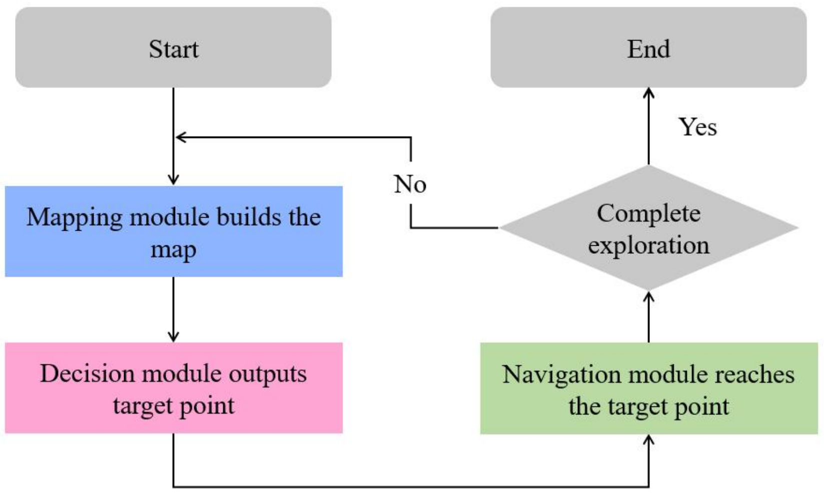

Start
End
Yes
No
Mapping module builds the map
Complete exploration
Decision module outputs target point
Navigation module reaches the target point

Figure 3.1 Navigation and mapping module flow chart

This chapter provides an introduction to the principles and implementations of the mapping module, decision-making module, and navigation module. For the mapping module, the chapter primarily covers the laser SLAM algorithm based on Karto and optimization methods based on the g2o framework. Regarding the decision-making module, the chapter delves into the composition and implementation of the improved DQN algorithm from three aspects: action space partitioning, reward function design, and network architecture design. As for the navigation module, the chapter mainly discusses global path planning based on A* and local path planning based on TEB.

3.2 Mapping module based on Karto SLAM

The core functionality of the mapping module is to construct a grid map of the surrounding environment as the robot moves, a process commonly referred to as Simultaneous Localization and Mapping (SLAM). Because autonomous exploration by mobile robots often takes place in challenging environments with poor lighting conditions, visual SLAM methods may not perform well. Therefore, in this paper, we employ a SLAM method using a laser rangefinder sensor. The functional representation of laser SLAM is as shown in Equation 3.1[31], where u and z represent the control inputs and laser rangefinder observations of the mobile robot, respectively, while m and l represent the environment map and the robot's pose, respectively.

$$m_t, l_t = f_{\text{mapping}}(m_{t-1}, l_{t-1}, u_t, z_t) \tag{3.1}$$

Classic laser SLAM methods can be primarily categorized into filter-based methods [38] and graph-based methods [39]. Filter-based methods utilize various filters or particle filtering techniques to simultaneously estimate the mobile robot's pose and the environment map. Filter-based methods like Extended Kalman Filters (EKF) cannot correct data errors, which means that when mapping large scenes for extended periods, accumulated data errors can lead to suboptimal mapping results. Particle filtering methods, on the other hand, require an increasing number of particles as the map grows, leading to increased memory and computational requirements. Hence, they are also not suitable for constructing large-scale maps.

Graph-based methods represent the SLAM problem using a graph, where nodes in the graph represent the poses of the mobile robot, and edges connecting adjacent nodes represent spatial constraints between two poses. Graph-based SLAM methods typically have a front-end and a back-end structure, as illustrated in Figure 3.2. The front-end is responsible for constructing nodes and edges in the graph, while the back-end adjusts the poses of these nodes to minimize the error between the odometry predictions and laser rangefinder observations. Compared to filter-based methods, graph-based methods tend to have smaller errors and are suitable for constructing large-scale maps.

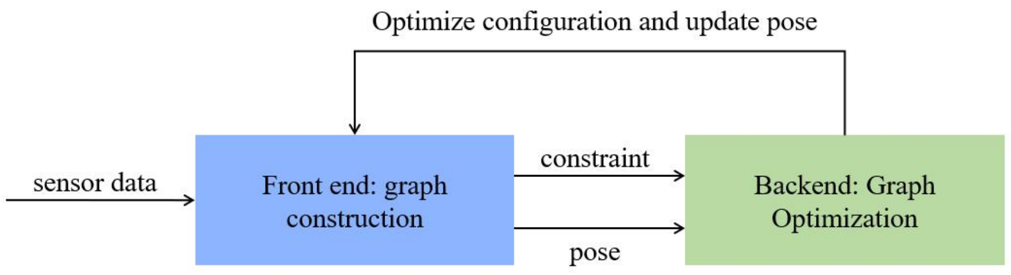

Optimize configuration and update pose
sensor data
Front end: graph construction
constraint
pose
Backend: Graph Optimization

Figure 3.2 Front-end and back-end structure diagram of graph-based SLAM method

Since maps in autonomous exploration scenarios with mobile robots are typically large, this paper utilizes the graph-based Karto SLAM algorithm [39].

3.2.1 Karto SLAM algorithm front end

In the graph-based Karto SLAM method, the front-end is responsible for identifying constraints based on sensor data and constructing nodes and edges in the graph. Whenever the laser rangefinder obtains measurement data, the front-end acquires odometry data and calculates the initial pose based on the previous pose and odometry data. Subsequently, the front-end generates rectangular-shaped regions of interest (Submaps) near the predicted position of the mobile robot based on the odometry data. At the same time, the front-end maps the laser data to different angles with a certain angular resolution and angular offset, creating a lookup table to obtain angle information for the mobile robot. The front-end then maps this lookup table onto the regions of interest with a certain displacement for use during matching. During the matching process, the front-end searches the region to obtain the position with the highest response value.
To improve search efficiency, coarse matching is first performed using low resolution to obtain candidate regions for the mobile robot's position. Subsequently, fine matching is carried out on these candidate regions to obtain an accurate solution for the position. Finally, the front-end adds

nodes and edges to the graph, where nodes represent the poses of keyframes (including the robot's angle and position information).

3.2.2 g2o backend optimization framework

Karto SLAM's front-end constructs a graph that needs to be optimized, consisting of nodes and edges. In this context, nodes represent the variables to be optimized, which can be either the robot's poses or environmental landmarks, while edges represent the constraints between nodes. The back-end of Karto SLAM optimizes this graph by adjusting the poses of various nodes to find the maximum likelihood estimate of the observed information, minimizing the error between the odometry predictions and laser rangefinder observations.

R. Kümmerle and colleagues introduced the g2o graph optimization framework [40], which is a stable, highly scalable, and easily transferable general-purpose framework for graph optimization that can be applied to various graph optimization problems. The g2o framework abstracts the optimization problem of SLAM as a nonlinear least squares problem, representing the optimization problem in a graph format and simplifying it based on the inherent properties of SLAM. The g2o graph optimization framework is known for its high efficiency, versatility, and user-friendliness, making it suitable for implementing the back-end optimization of Karto SLAM.

To ensure that the mapping performance of Karto SLAM meets the basic requirements, this paper conducted mapping simulations in the Gazebo environment with a test map. The model of the real environment in the simulation is depicted in Figure 3.5, and the grid map obtained from the mapping process is shown in Figure 3.6.

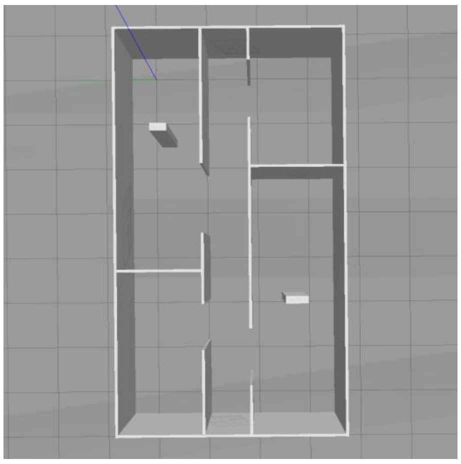


Figure 3.5 Real Environment Model

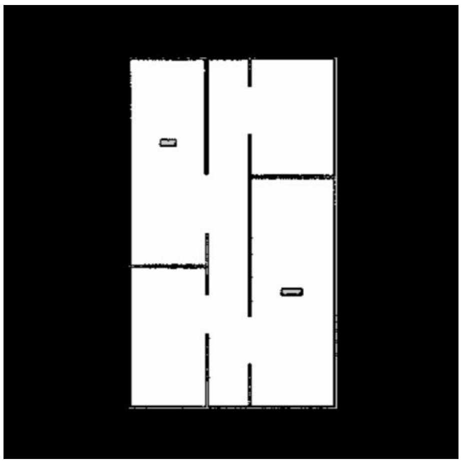


Figure 3.6: Grid Map Constructed by Karto SLAM

After multiple tests, the Karto SLAM algorithm based on g2o optimization has shown good performance and can be used for the mapping module.

3.3 Decision Module Based on Improved DQN Algorithm

The function of the decision module is to select the target point that the mobile robot should head to in the next moment based on the robot's pose and the map information of its surroundings. The decision module is the core module for solving autonomous exploration in this paper. It receives input from the mapping module and provides output to the navigation module. The functional expression of the decision module is given in Equation 3.2 [31].

$$g_t = f_{\text{decision}}(l_{0:t}, m_t) \tag{3.2}$$

Where g_t represents the ultimate output of the decision module, $l_{0:t}$ represents the robot's pose from time 0 to t, m_t and represents the environmental map output by the mapping module at time t. Research on autonomous exploration decision-making for mobile robots is predominantly based on boundary methods and heuristic search methods. These approaches, when applied to complex environments, necessitate the construction of numerous rules and constraints, heavily relying on domain expertise, and are labor-intensive and not very versatile. In this paper, we employ the improved DQN method (D3QN PER) from deep reinforcement learning as the decision-making module. Compared to boundary-based methods, this approach can automatically learn visual features for exploration and improve strategies through continuous trial and error. This method avoids the complex process of rule setting and does not depend on stacking exploration rules, which is beneficial for further research on autonomous exploration in more complex scenarios.

3.3.1 D3QN PER Algorithm

In Section 2.4 of this paper, we provided a detailed introduction to DQN and its three improvement algorithms, including Double DQN, Dueling DQN, and Prioritized Experience Replay, explaining their structures, principles, and technical details. The above-mentioned three DQN improvement algorithms can effectively accelerate convergence and enhance algorithm performance, and they do not conflict with each other in terms of network structure. This paper combines these three improved DQN algorithms to create the D3QN PER (Dueling Double DQN with PER) algorithm, leveraging their complementary advantages. The pseudo code for this algorithm is presented in Table 3.1.

Table 3.1 Pseudo Code for D3QN PER Algorithm

```
Input：Stride $\eta$
       mini batch $k$
       Experience replay period $K$
       Experience pool size $N$
       Hyper params $\alpha$ $\beta$
Process:
```

```
Initialize the experience pool $\mathcal{H} = \emptyset, \Delta = 0, p_1 = 1$
Observe $S_0$, Choose $A_0 \sim \pi_\theta(S_0)$
for $t = 1$ to $T$ do
    Observe $S_t, R_t, \gamma_t$
    Based on maximum priority $p_t = \max_{i<t} p_i$ Store$(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$ into $\mathcal{H}$
    if $t \equiv 0 \bmod K$ then
        for $j = 1$ to $k$ do
            Sample $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$
            Calculate importance sampling weights $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
            Calculate TD error:
                $\delta_j = R_j + \gamma_j Q_{\text{target}}\left(S_j, arg\ max_a Q(S_j, a)\right) - Q(S_{j-1}, A_{j-1})$
            Update priority $p_j \leftarrow |\delta_j|$
            Accumulate bias $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$
        end for
        Update network weights $\theta \leftarrow \theta + \eta \cdot \Delta$ ,Reset $\Delta = 0$
        Update the Target network periodically $\theta_{\text{target}} \leftarrow \theta$
    end if
    Choose action $A_t \sim \pi_\theta(S_t)$
end for
```

Output: Action-value function Q

3.3.2 Action Space Partitioning

The task of the decision module is to select a point on the map as the next target point for the mobile robot to move to. Algorithms based on the DQN framework are suitable for solving decision problems with discrete action spaces but are not suitable for continuous action space problems.

In theory, the environment map consists of countless points, and the action space is infinitely large. However, the mapping module can generate a grid map based on the surrounding environment information, and the number of points in the grid map is determined by the size of the environment and the mapping resolution. Nevertheless, there are still many points in the map, often reaching tens of thousands or even hundreds of thousands, resulting in an excessively large action space. Considering that the model of the mobile robot can cover several points in terms of area, the effect of the robot moving to a target point is similar to the effect produced when the robot moves to a point within a certain neighborhood of the target point, and the difference can be neglected. Therefore, to further reduce the action space, this paper performs sampling operations on the grid map, and the action space is composed of these sampled points, as shown in Figure 3.7. In the figure, green points represent sampled points in open areas, red points represent sampled points too close to obstacles, and blue points represent sampled points in unknown areas.

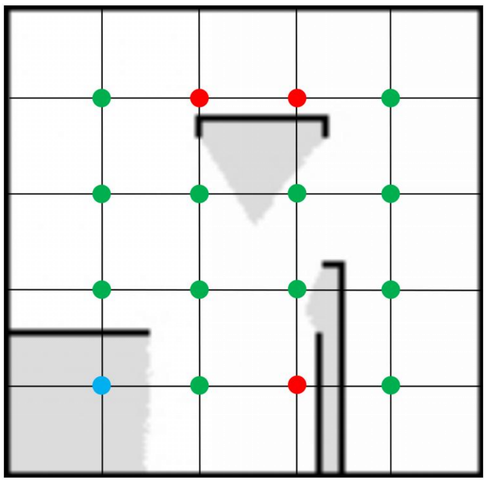


Figure 3.7 Sampling Operation on Grid Map

The expression for the size of the action space $N_{\mathcal{A}}$ is shown in Equation 3.3, where l_{width} and l_{height} are the width and height of the real environment, and T_{width} and T_{height} are the sampling intervals in the width and height directions, respectively.

$$N_{\mathcal{A}} = \frac{l_{\text{width}} \cdot l_{\text{height}}}{T_{\text{width}} \cdot T_{\text{height}}} \tag{3.3}$$

Action numbering $n_a \in [0, N_{\mathcal{A}} - 1]$ corresponds one-to-one with the sampling points on the grid map.

3.3.3 Reward Function Design

The purpose of autonomous exploration for a mobile robot is to construct an environment map that is as similar as possible to the real environment. In practical applications, such as robotic vacuum cleaners and rescue robots, the goal is to complete the maximum amount of work within limited battery life. Therefore, the mobile robot is designed to choose the shortest path to explore the environment as thoroughly as possible. The objective function for autonomous exploration can be represented as Equation 3.4.

$$c(\hat{m}, x_{t=0:T}) = \min_{u_{t=0:T}} \| m - \hat{m} \|^2 + L(x_{t=0:T}) \tag{3.4}$$

Here, $\hat{m}$ represents the estimated map, m is the true map, and $L(x_{t=0:T})$ is the path length during exploration from time $t = 0$ to $t = T$. It is worth noting that 'L' does not refer to time but rather the number of times the robot moves to a target point. However, since the environment map is unknown, the true value of the map cannot be accurately determined, making it challenging to find the minimum value of this objective function.

In this paper, two reward functions r_t are designed from two perspectives.

(1) Reward Function Based on Map Entropy

Influenced by information-based decision-making methods, this paper uses the Shannon entropy of the map to measure the difference between the constructed map and the real environment. Map entropy can represent the uncertainty of the map, which decreases as the map completion level increases, reflecting the rise in map completion. The formula for map entropy is given in Equation 3.5.

$$H(m) = -\sum_i \sum_j p(m_{i,j}) \log p(m_{i,j}) \tag{3.5}$$

Where $p(m_{i,j})$ is the occupancy probability of the cross grid in the i row and j column. The mapping module transfers map information to the decision module in the form of a 2D array, with each grid having three possible states: unknown, free, and occupied, corresponding to values of -1, 0, $\theta \in (0, 255]$ in the corresponding position data in the 2D array. A higher θ represents a higher probability that the point is occupied.

$$p(m_{i,j}) = \frac{\theta}{255} \tag{3.6}$$

Considering that during the mapping process, the mobile robot consumes battery power when it rotates in place, the angle rotated by the mobile robot is also considered in the cost of the robot's movement. The cost expression for the robot's path is given in Equation 3.7.

$$S(x_{t-1}, x_t) = k_1 L(x_{t-1}, x_t) + k_2 R(x_{t-1}, x_t) \tag{3.7}$$

Here, $R(x_{t-1}, x_t)$ represents the total angle rotated by the mobile robot during exploration, and k_1, k_2 are hyperparameters that balances the path length and the rotation angle. It ensures a more balanced contribution of these two factors to the reward function. Finally, the reward function is expressed as Equation 3.8.

$$r_t = \alpha\big(H(m_{t-1}) - H(m_t) - \beta S(x_{t-1}, x_t)\big) \tag{3.8}$$

Here, α is a coefficient to make the training more stable, and β is a hyperparameter that balances the change in map entropy with the path traveled by the mobile robot, ensuring a more balanced contribution of these two factors.

(2) Reward Function Based on the Number of Known Points

During the process where the mobile robot moves from its current position to the next target point, the mapping module updates the state information of each grid point based on the principles of laser SLAM, such as known, unknown, and occupied. As the robot explores, it gradually converts unknown points into known points and updates map information. The more known points there are in the map, the higher the exploration completion level. Therefore, this paper measures the exploration completion rate and efficiency based on the number of known points and the rate of change of known points.

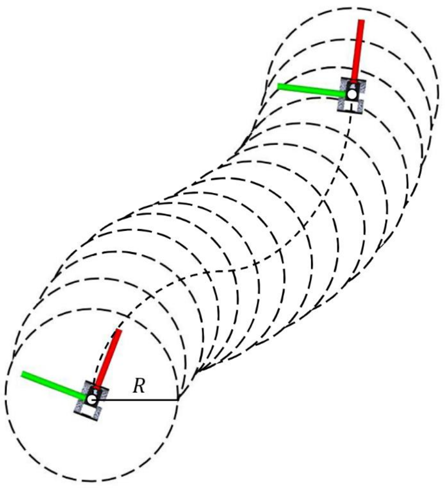


Figure 3.8 Schematic Diagram of Laser Radar Scanned Area

The expression of the reward function is given in Equation 3.9, where p represents the number of known points in the constructed map, T_{SLAM} is the resolution set in the Karto SLAM mapping module, S is the cost of the robot's path, and R is the effective detection range of the laser radar.

$$r_t = \frac{(p_t - p_{t-1}) T_{\text{SLAM}}^2}{\gamma \pi R^2 + 2\pi R S(x_{t-1}, x_t)} \tag{3.9}$$

It is worth noting that when the mobile robot repeatedly traverses an already explored area, the denominator in Equation 3.9 no longer accurately represents the area scanned by the laser radar. The calculated area will be larger than the true area, resulting in a smaller reward value. However, in this paper, the mobile robot is expected to explore as large an area as possible by taking the shortest path and is not expected to revisit already explored areas. In such cases, it should receive smaller rewards, so the reward function does not need to be adjusted for this situation.

In addition, to shorten the training time and make the training safer, this paper has designed heuristic reward functions for several special cases, which are applicable to the two types of reward functions mentioned above.

A. Before Navigation Starts:

$$\begin{cases} r_t = -1, & \text{The next target point is in an unknown area} \\ r_t = -1, & \text{The next target point is within the obstacle} \end{cases} \tag{3.10}$$

B. After Navigation Starts:

$$\begin{cases} r_t = -1, & \text{Navigation failed} \\ r_t = -1, & \text{Collision} \\ r_t = -1, & \text{The mobile robot is not movin} \end{cases} \tag{3.11}$$

C. Mapping Completion:

$$r_t = 1, \text{ Known area ratio } \rho > 0.90 \tag{3.12}$$

3.3.4 Network Architecture Design

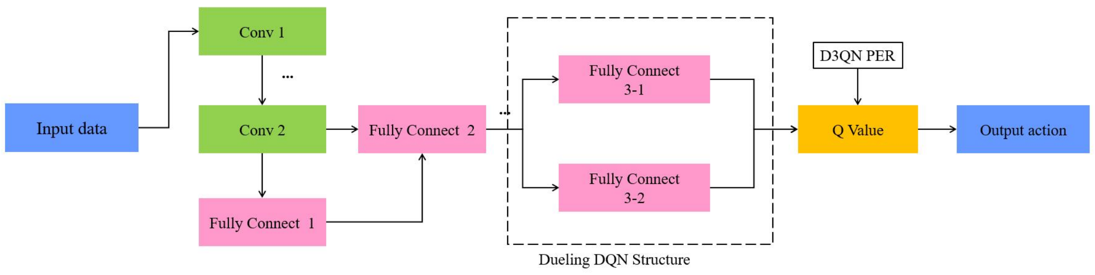


Figure 3.9 Overall Network Architecture

The overall architecture of the deep reinforcement learning decision model based on the improved DQN algorithm in this paper is shown in Figure 3.9. Input data goes through convolutional layers and fully connected layers to extract deep features, and the D3QN PER algorithm is used to calculate the Q-values, which are then used to output actions. This subsection will provide a detailed description of the network model's structure and components in terms of network input, feature extraction, and D3QN PER settings.

(1) Network Input

The mapping module provides map data of the surrounding environment, and the robot's position and pose can be obtained from odometry data. It is evident that the current position of the mobile robot has a significant impact on the selection of the next target point. In addition, influenced by boundary-based exploration algorithms, this paper also takes the boundaries between known and unknown areas as network inputs to serve as prior knowledge to expedite training. Therefore, the input to the deep reinforcement learning model in this paper consists of images that are of the same size, not cropped, and are a combination of the grid map, boundary information, and current pose, as shown in Figures 3.11, 3.12, and 3.13, respectively.

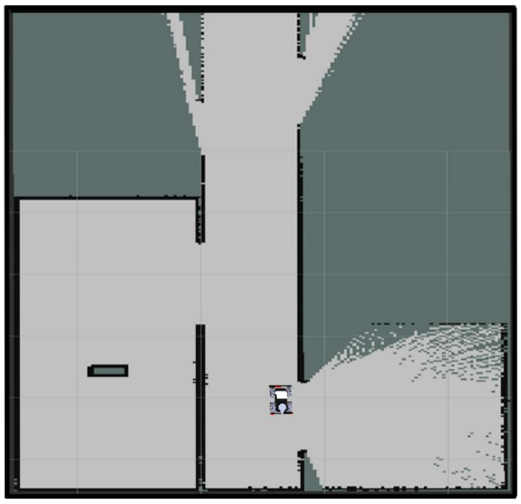


Figure 3.10 Mobile robot and environment map

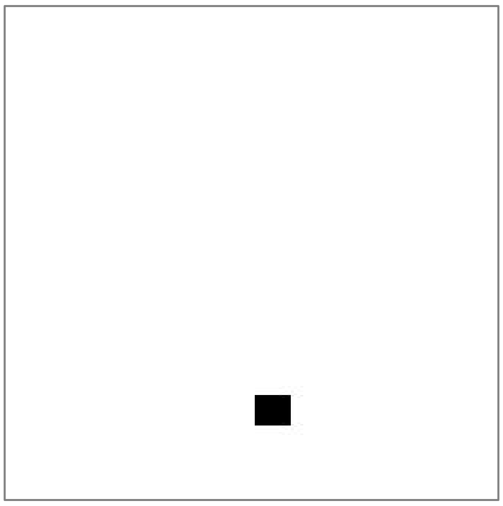


Figure 3.11 Position and posture of mobile robot

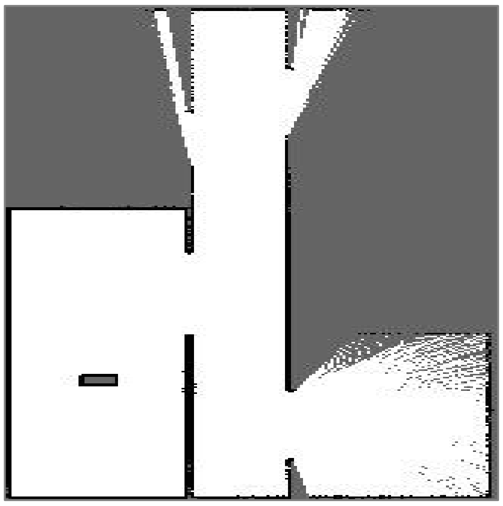


Figure 3.12 Environmental grid map

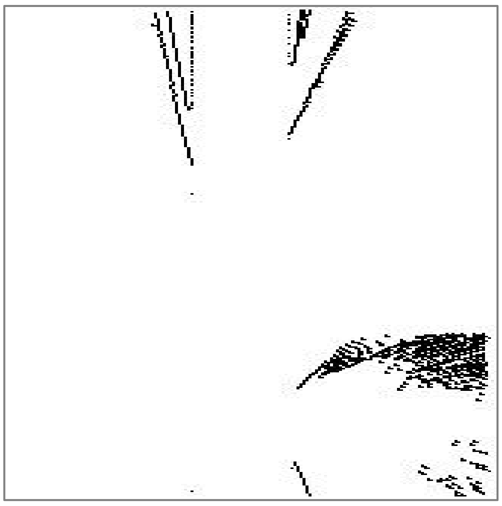


Figure 3.13 Boundary information

(2) Feature Extraction

Convolutional neural networks are the most widely used feature extractors. In this paper, several convolutional layer structures are used for training, and different network structure parameters yield significant differences in model performance. Here, a set of parameters is presented to illustrate the general idea of the paper. Two convolutional layers are used in this paper, namely Convolutional Layer 1 and Convolutional Layer 2, with the convolutional layer parameters as shown in Table 3.2. After each convolutional layer, there are activation function layers and max-pooling layers. After passing through the convolutional layers, the width and height of the input data are both reduced to half of their original sizes.

Table 3.2 Convolution layer parameters

Network layer name	Height	Width	Number of input	Number of output	Step size	Filling method

			channels	channels		
Convolutional layer 1	5	5	3	32	1	SAME
Convolutional layer 2	5	5	32	64	1	SAME

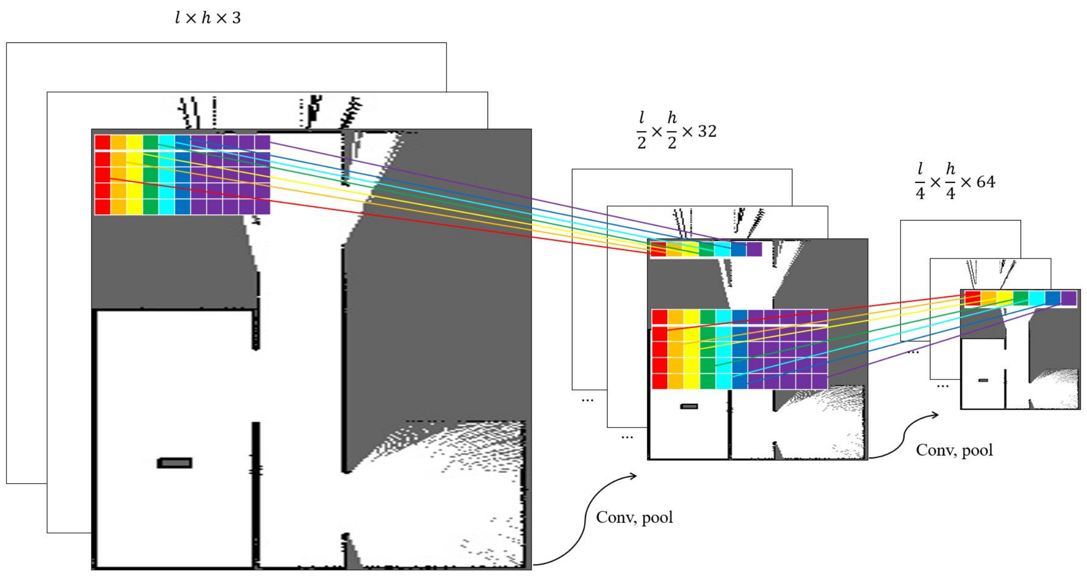

$l \times h \times 3$
$\frac{l}{2} \times \frac{h}{2} \times 32$
$\frac{l}{4} \times \frac{h}{4} \times 64$
Conv, pool
Conv, pool

Figure3.14 Convolution and Pooling Process Illustration

After the convolutional layers, this paper employs three fully connected layers. The third fully connected layer follows the structure of Dueling DQN, splitting into two branches to calculate the state-value function and action-advantage function. The parameters for the fully connected layers are shown in Table 3.3. Here, $N_{\mathcal{A}}$ represents the size of the action space, l_{width} and l_{height} represent the width and height of the environment map, respectively, while T_{SLAM} is the resolution of the mapping (without sampling).

Table 3.3 Fully connected layer parameters

Network layer name	Input dimension	Output dimension
Fully connected layer 1	$4 \cdot \frac{l_{\text{width}} \cdot l_{\text{height}}}{T_{\text{SLAM}}^2}$	512
Fully connected layer 2	512	256
Fully connected layer 3-1	256	1
Fully connected layer 3-2	256	$N_{\mathcal{A}}$

(3) D3QN PER Configuration

The decision module in this paper calculates Q-values and updates the network using the D3QN PER algorithm. The structure and principles of the D3QN PER model have been detailed in Sections 2.4 and 3.3.1. This section will only introduce the parameter settings during training. The parameter settings for the D3QN algorithm are presented in Table 3.4, while the parameter settings for the PER algorithm are shown in Table 3.5.

Table 3.4 D3QN algorithm parameter table

Parameter	Meaning	Value
γ	discount factor	0.90
Learning_rate	learning rate	0.001
ϵ_{start}	Greedy strategy starting parameters	0.5
ϵ_{final}	Greedy strategy termination parameters	0.05
Num_action	Number of actions	$N_{\mathcal{A}}$
Replay_size	memory pool size	2000
Num_training	training times	10000
Update_steps	Target network update steps	100
Num_batch	mini-batch size	32

To expedite convergence, this paper sets the ϵ-greedy exploration parameter ϵ to linearly decrease from its initial value ϵ_{start} as training steps progress until $\epsilon = \epsilon_{final}$. The expression is given by Equation 3.13.

$$\epsilon = \epsilon - \frac{\epsilon_{\text{start}} - \epsilon_{\text{final}}}{\text{Num\_training}} \tag{3.13}$$

Table 3.5 PER algorithm parameter table

Params	meaning	Value
ε_{PER}	The probability of being replayed when TD error is 0	0.0001
α_{PER}	priority parameter	0.6
β_{init}	Adjust the degree of deviation	0.4

The parameter β_{PER}, which controls the PER algorithm's importance sampling adjustment, also linearly decreases with training steps, as shown in Equation 3.14.

$$\beta_{\text{PER}} = \beta_{\text{PER}} + \frac{1.0 - \beta_{\text{init}}}{\text{Num\_training}} \tag{3.14}$$

Once the Q-values are calculated, the decision network computes the loss function $\mathcal{L}$

according to Equation 3.15 and uses the mean squared error (MSE) minimization to update the model parameters θ, where r, s, a represent rewards, states, and actions, respectively.

$$\mathcal{L} = \left(r_t + \gamma Q\left(s_{t+1}, \arg\max_{a'} Q\left(s_{t+1}, a'; \theta\right); \theta^-\right) - Q(s, a; \theta)\right)^2 \tag{3.15}$$

(4) Target Point Coordinate Transformation:

After the decision is made, the deep reinforcement learning model based on the D3QN PER algorithm outputs the action's index based on the maximum Q-value. The decision module then converts the action index n_a into the coordinates of the target point (x, y, z), calculated as shown in Equation 3.16.

$$\begin{cases} x = \frac{n_a}{l_{\text{height}}} T_{\text{height}} + x_0 \\ y = \left(n_a - \left\lfloor \frac{n_a}{l_{\text{width}}} \right\rfloor l_{\text{width}}\right) T_{\text{width}} + y_0 \\ z = 0 \end{cases} \tag{3.16}$$

Where (x_0, y_0) represents the initial coordinates of the map. In this paper, a holonomic model is used for training and testing the mobile robot's motion, so the orientation of the target point is not critical. To ensure algorithm generality and versatility for different motion models, this paper uses the vector between the target point and the current position as the Euler angle $(\beta_{\text{pitch}}, \beta_{\text{roll}}, \beta_{yaw})$, as shown in Equation 3.17.

$$\begin{cases} \beta_{\text{pitch}} = \beta_{\text{roll}} = 0 \\ \beta_{yaw} = \arctan \frac{y - y_{\text{r}}}{x - x_{\text{r}}} \end{cases} \tag{3.17}$$

The decision module then passes array $(\text{x}, \text{y}, \text{z}, \beta_{\text{pitch}}, \beta_{\text{roll}}, \beta_{\text{yaw}})$ to the navigation module, which plans the path from the current point to the target point. The geometric relationship between the target point and the mobile robot's pose is illustrated in Figure 3.15.

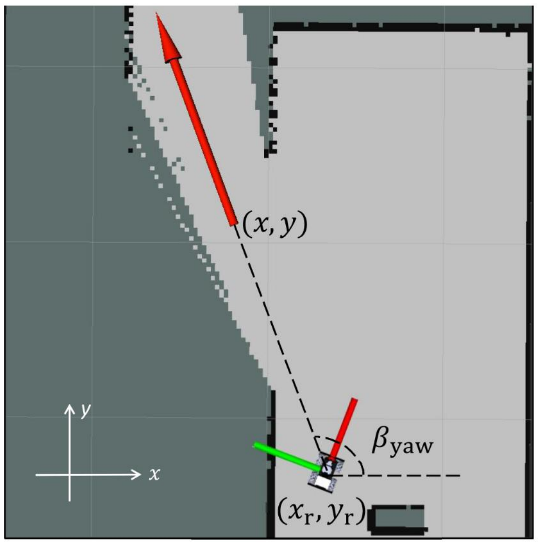


Figure 3.15 Geometric Relationship Between Target Point and Mobile Robot Pose

3.4 Navigation Module Based on A* Search Algorithm

The core function of the navigation module is to plan a path and output control commands based on the current pose of the mobile robot and the position of the next target point, allowing the robot to reach the target location. The functional expression of the navigation module is shown in Equation 3.18 [31], where τ represents the planned path, m and l represent the environment map and the robot's pose, and g represents the next target point output by the decision module.

$$\tau_{t:t+T} = f_{\text{planning}}(l_{0:t}, g, m_t) \tag{3.18}$$

Existing mature navigation algorithms often consist of two parts: global path planning and local path planning. Global path planning determines the shortest path from the current position to the target position based on the robot's current pose and known map information. Local path planning modifies the path based on real-time sensor data and ultimately outputs the robot's linear and angular velocities. This paper uses the widely used A* search algorithm [41] for global path planning and the Timed-Elastic Band (TEB) method [42] for local path planning. Compared to Pure Pursuit [43], the TEB method dynamically modifies the local path near the mobile robot in real-time, effectively avoiding dynamic obstacles in the map and preventing collisions.

3.4.1 Global Path Planning Based on A* Search Algorithm

The A* algorithm is an optimization of the breadth-first search algorithm and can calculate the shortest path from the current position to the target position in a static map. A* is a heuristic algorithm, and its key lies in calculating the priority of each node through an estimation function. Equation 3.19 represents the expression of the estimation function, where s represents the node currently being expanded, $f(s)$ represents the evaluation value of reaching the target grid point via the node from the start point, and $g(s)$ represents the cost of moving from the start point to grid

$$f(s) = g(s) + h(s) \tag{3.19}$$

By adjusting the heuristic function, the algorithm's speed and accuracy can be balanced. A smaller heuristic function leans toward accuracy, while a larger heuristic function leans toward speed. Heuristic functions can use Manhattan distance, diagonal distance, Euclidean distance, etc. When the heuristic function is always 0, the A* algorithm degenerates into the Dijkstra algorithm. The A* algorithm prioritizes selecting nodes with the smallest $f(s)$ value as the next node to be explored. The A* algorithm uses two lists to store nodes to be explored and nodes that have already been explored.

3.4.2 Local Path Planning Based on the TEB Algorithm

When a mobile robot is navigating, it converts the global path into a trajectory through local path planning, tracks the trajectory, and eventually reaches the target point. S. Quinlan et al. [44]

proposed a method based on Elastic Bands (EB) to reduce the gap between the global path planning and the actual traversed trajectory. The main idea of this method is to treat the actual trajectory of the mobile robot as an elastic band. The initial shape of the elastic band is the path generated by global planning. When the laser radar detects obstacles or moving objects, it introduces two virtual forces: internal contraction force and external repulsion force. Under the interaction of these two forces, the elastic band deforms, generating a local path with sufficient distance from obstacles to avoid collisions. The elastic band can be described as a sequence of intermediate poses of the robot, as mathematically expressed in Equation 3.20 [44].

$$Q = \{(x_i, y_i, \beta_i)^T\}_{i=0\ldots n}, n \in \mathbb{N} \tag{3.20}$$

x_i, y_i, β_i represent the robot's horizontal and vertical coordinates and orientation.Traditional elastic band methods only consider geometric constraints on the global path and do not take into account the robot's dynamic constraints (such as velocity, acceleration, etc.). C. Roesmann et al. [42] introduced time information into the elastic band method and proposed the Timed Elastic Band (TEB) method. This method can generate trajectories with explicit time dependence, allowing real-time generation of commands for the mobile robot's lower-level motion controller. The time sequence of the Timed Elastic Band τ is shown in Equation 3.21 [42], where ΔT represents the time interval between adjacent robot poses.

$$\tau = \{\Delta T_i\}_{i=0\ldots n-1}, n \in \mathbb{N} \tag{3.21}$$

The pose sequence formed by x_i, y_i, β_i and the time sequence formed by Δ T can be intuitively represented in Figure 3.16.

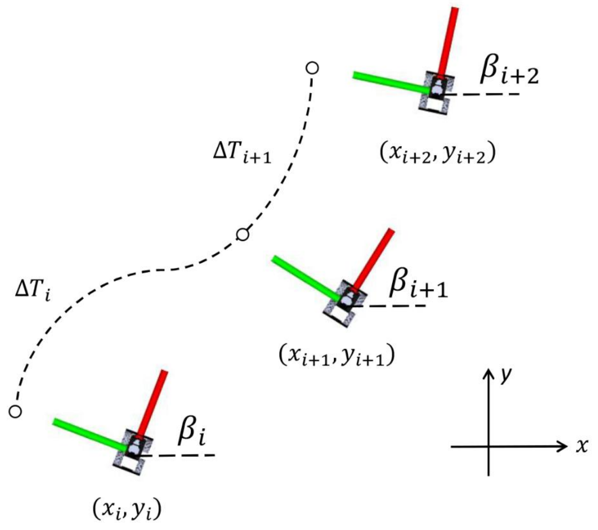


Figure 3.16 Schematic Diagram of Pose Sequence and Time Sequence

The timed elastic band is defined as a tuple of pose sequences and time sequences. Its core idea is to adjust and optimize the TEB through weighted multi-objective real-time optimization. The optimization objectives are obtained by summing the weighted dynamic constraints during the mobile robot's motion process. The mathematical expression is shown in Equation 3.22 [42].

$$B^* = \underset{B}{\operatorname{argmin}} \sum_k \gamma_k f_k(B) \tag{3.22}$$

Where: B^* represents the optimized timed elastic band. γ_k represents the weights of individual components. $f_k(B)$ represents the objective functions of various dynamic constraints, such as constraints related to obstacles and waypoints, time constraints, velocity and acceleration constraints, non-holonomic constraints, etc.

To ensure that the navigation results of A* global path planning and TEB local path planning meet the basic requirements, this thesis conducted mobile robot navigation simulations in the Gazebo environment on a test map. In Figures 3.18 and 3.19, the blue lines represent trajectories obtained from global path planning, while the red lines represent trajectories obtained from local path planning. Figure 3.20 shows the actual route taken during this navigation.

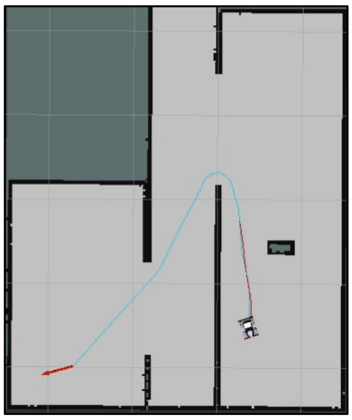


Figure 3.18 Initial navigation status

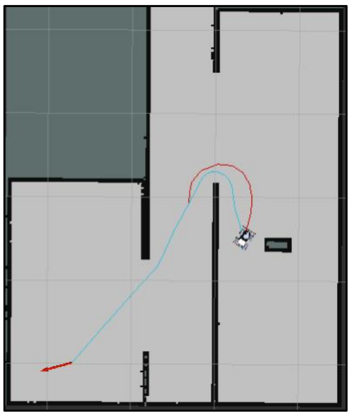


Figure 3.19 Intermediate navigation process

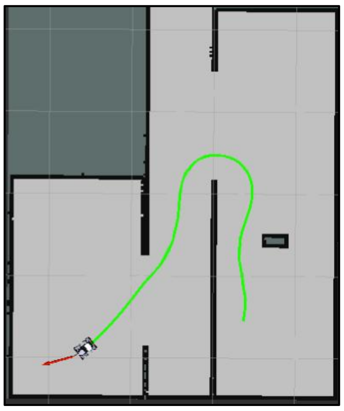


Figure 3.20 Actual navigation route

After multiple tests, the mobile robot can successfully reach the target point in the vast majority of cases. Only in cases where the single-step navigation distance is too far is there a small probability of failure. This design helps avoid low training efficiency and poor model performance due to navigation failures.

3.5 Summary of This Chapter

In this chapter, we first used the graph-based Karto SLAM algorithm and g2o backend optimization to create the mapping module, which can construct a grid map of the surrounding environment using laser point cloud data. Then, we employed the improved DQN (D3QN PER) deep reinforcement learning algorithm to form the decision module, allowing for optimal decision-making based on the environmental map and robot pose, resulting in the output of the coordinates of the next target point. Finally, we used A* global path planning and TEB local path planning to build the navigation module, which can output the mobile robot's base motion control commands, enabling it to reach the target point. This chapter constitutes the core of the entire thesis, constructing an autonomous exploration system based on deep reinforcement learning. The system is highly modular, flexible to use, and suitable for transfer to various applications.

Chapter 4
Experimental Analysis

4.1 Introduction

To simplify training and reduce training time, the training of deep reinforcement models is typically conducted in a simulation environment. This chapter addresses the autonomous exploration problem of mobile robots, relying primarily on three modules: mapping, decision-making, and navigation. The simulation environment must provide the input data required by these three modules and strive to closely resemble the data found in a real-world environment. Firstly, this chapter selects appropriate simulation software and completes the simulation modeling of the environmental map and the mobile robot. Then, it proceeds to train the autonomous exploration method proposed in Chapter 3, based on the improved DQN algorithm. Multiple target decision models are trained with different reward functions, different training map environments, different deep learning models, and various model parameters. These models are then horizontally compared to validate their convergence and performance, including metrics such as environment exploration rate, environment exploration efficiency, average path length, and environmental adaptability. Finally, this chapter deploys real-time mapping algorithms, navigation algorithms, and deep reinforcement learning models trained in the simulation environment on a physical mobile robot validation platform. It tests the mapping performance of the unknown environment autonomous exploration algorithm in a real-world environment.

4.2 Simulation Experiment Environment Based on ROS and Gazebo

4.2.1 Introduction to ROS

ROS (Robot Operating System) originated from the STAIR and Personal Robotics projects at Stanford University and is currently maintained and managed by the Open Source Robotics Foundation (OSRF). ROS is a cross-platform open-source software development tool designed for robotic applications. It provides a standard software platform and various robot modules for developers in the field of robotics. ROS offers tools for startup, introspection, debugging, visualization, plotting, recording, and playback, which accelerate the progress of development teams, making projects easier to maintain, contribute to, and reuse. The ROS ecosystem is well-established, and this paper can directly use ROS interfaces for sensors like laser rangefinders without the need to write hardware-related code.

The core of ROS is its message-passing system, typically referred to as a "pipeline." ROS designates different applications as nodes, such as mapping nodes and navigation nodes. These nodes can communicate and exchange data through three methods: Topic, Service, and Action. The data communication between nodes is illustrated in Figure 4.1.

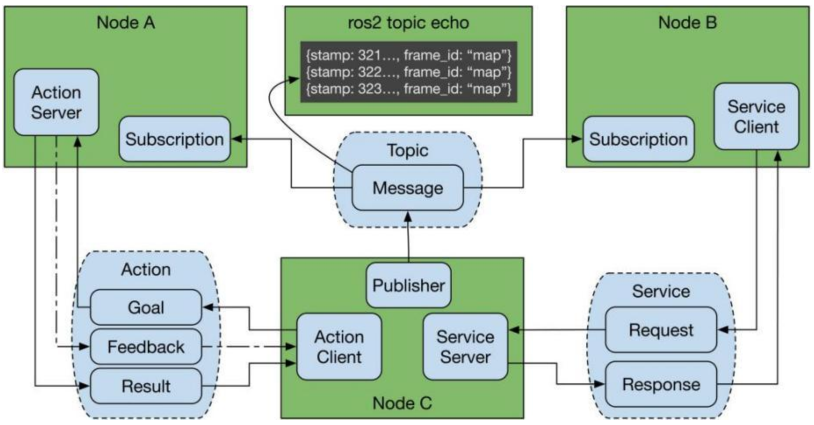


Figure 4.1 Communication between nodes using topics, services, and actions [47]

There is a substantial amount of real-time data exchange among the mapping, navigation, and decision modules of the mobile robot discussed in this paper. For instance, the mapping module acquires data from sensors like LiDAR and odometers, the decision module outputs action IDs to the navigation module, and the navigation module provides velocity and angular velocity information to the mobile robot base, among other interactions. Due to the numerous sensors and modules involved, the data exchange during the autonomous exploration of the mobile robot can be quite complex. ROS provides an excellent platform for managing data communication, greatly simplifying communication and data exchange between modules. Users only need to write nodes, define communication methods, and specify data formats to establish connections between nodes, making it convenient, concise, and conducive to algorithm development.

4.2.2 Introduction to Gazebo

This paper demands that the simulation environment closely resembles reality to ensure that the deep reinforcement learning models trained here will not exhibit significant performance variations when deployed in real-world scenarios. Gazebo is an open-source 3D robot simulation software known for its robust physics engine, high-quality graphics, and user-friendly programming and graphical interfaces. Gazebo's generic model is based on the open-source physics engine (ODE) and the OpenGL GLUT library for modeling robot joints, sensors, and bodies, effectively mapping to real-world physics. Gazebo excels at simulating physical properties of the environment, including mass, inertia, velocity, acceleration, friction, lighting, and more, creating effects and interactions closely resembling those in the real world. Gazebo offers a rich variety of models, encompassing joint types such as revolute, prismatic, planar, floating, and fixed joints, as well as attributes for robot bodies like shape, color, material, collision ranges, and inertia. Moreover, users can integrate their own mechanical models created with software like Solidworks or AutoCAD using the URDF (Unified Robot Description Format) and incorporate them into the simulation environment. Gazebo is equipped with a wide array of simulation sensors, including laser rangefinders, IMUs, cameras, GPS, sonar, contact sensors, and more. These sensors analyze the simulation environment and generate data approximating real-world physical sensor data.

Furthermore, Gazebo offers an extensive array of simulation plugins, including elevator plugins, aerodynamics plugins, fluid dynamics plugins, gravity compensation plugins, encompassing plugins, flash plugins, contact plugins, drive plugins, and more. Users can employ these plugins to customize and enhance the simulation environment as needed. Various plugins endow the robot with specific physical properties. It is worth emphasizing that Gazebo includes ROS plugins, often acting as a ROS node to publish various robot data and provide various services. Users can achieve high-quality robot simulation effects and conduct various experiments simply by defining parameters such as materials, geometry, dynamics, and noise in the environment.

4.2.3 Building the Simulation Experimental Environment

(1) Construction of the Environmental Map

During the training process, the robot needs to continuously move and map in the environment, learning through trial and error. The choice of the training environment has a decisive impact on the final performance of the model. Therefore, this paper conducts training using both simple and complex maps. The simple map, as shown in Figure 4.3, has fewer obstacles and wider aisles, while the complex map, as shown in Figure 4.4, has more obstacles, narrower aisles, and "dead-end" paths. Map modeling was performed using the Building Editor built into the Gazebo simulation software.

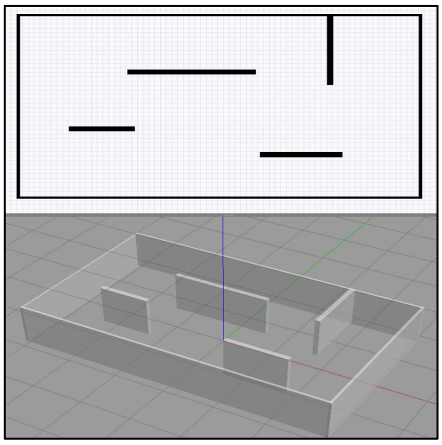


Figure 4.3: A Simple Training Map

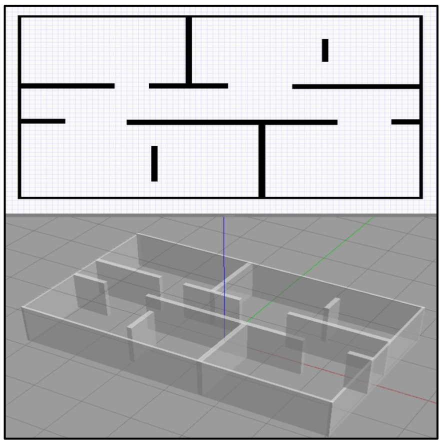


Figure 4.4: A Complex Training Map

(2) Construction of the Mobile Robot

This paper uses Solidworks for the mechanical design of the omnidirectional mobile robot model, including the mecanum wheels, LiDAR, and chassis, as shown in Figures 4.5, 4.6, and 4.7, respectively.

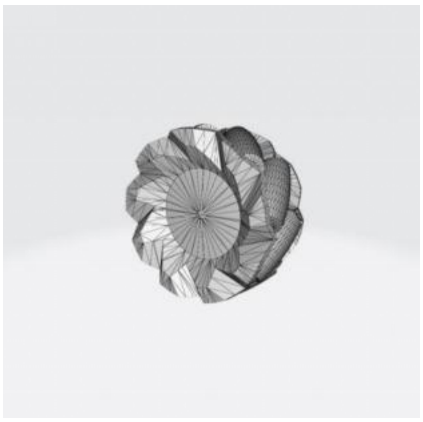


Figure 4.5: Mecanum Wheel Mechanical Structure Diagram

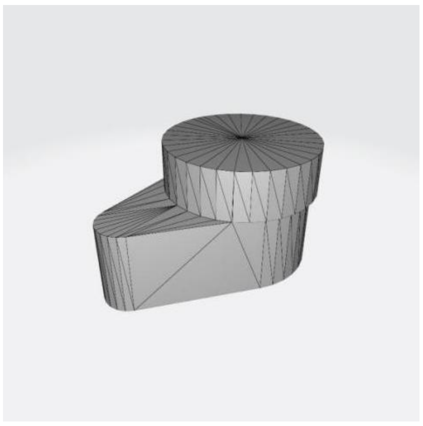


Figure 4.6: LiDAR Mechanical Structure Diagram

Through the SW2URDF plugin, coordinate frames are named, joint types are set, and inertia parameters are configured to output the mechanical structure model in the URDF (Unified Robot Description Format). The URDF file is then converted to the xacro format and configured with the necessary plugins. The libgazebo_ros_planar_move.so library corresponds to the planar movement plugin, enabling omnidirectional movement of the mecanum wheels and the publishing of odometry data. The libgazebo_ros_p3d.so library corresponds to the ground truth pose plugin, providing access to the true pose of the mobile robot and its components. The libgazebo_ros_bumper.so library corresponds to the collision detection plugin, allowing the detection of forces applied to the contact sensors and determining if the robot has collided. Contact sensors are deployed on the front and rear bumpers of the robot, with the bumpers only having collision properties and no mass or inertia, ensuring they do not interfere with the robot's motion, as shown in Figure 4.8.

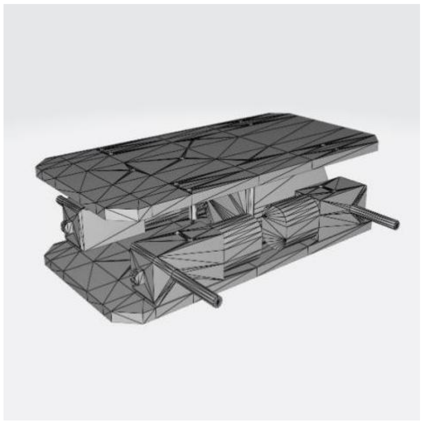


Figure 4.7 Mechanical structure of the chassis

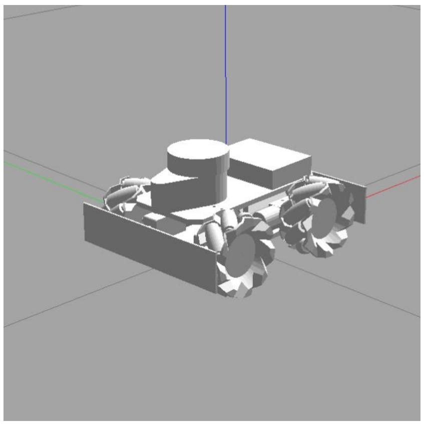


Figure 4.8 Omnidirectional mobile robot model

After loading the model into Gazebo via ROS, you can visualize the transformation relationships between the coordinate frames of the mobile robot using the Rqt visualization plugin, as shown in Figure 4.9. In this visualization, the odometry coordinate frame serves as the parent coordinate frame of the base coordinate frame, and the coordinate frames of the two collision sensors, the motion controller, the LiDAR, and the four mecanum wheels are all child coordinate frames of the base coordinate frame.

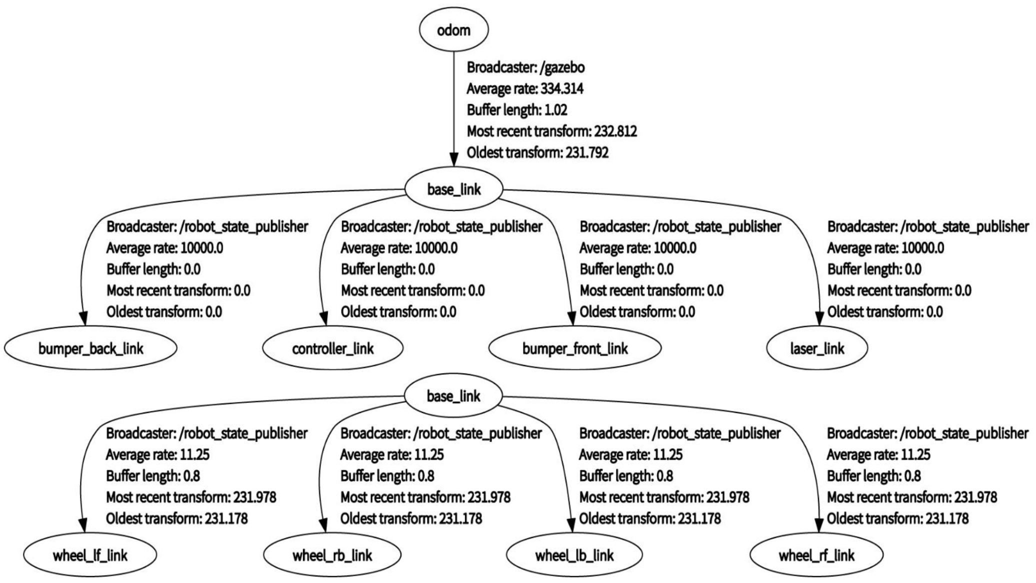


Figure 4.9 The TF tree of the mobile robot

4.3 Simulation Experiment Workflow

4.3.1 Overall System Workflow

In Chapter 3, this paper introduced the functions and implementations of the mapping, decision-making, and navigation modules individually. This section describes the overall process of autonomous exploration for mobile robots. The workflow of the entire mobile robot autonomous exploration system is shown in Figure 4.10. The blue part constitutes the mapping module, the pink part constitutes the decision-making module, and the green part constitutes the navigation module. In each training step, the mobile robot collects odometry data and LiDAR point cloud data. The Karto SLAM mapping module utilizes this data to construct maps and sends the map data to the decision-making module. The decision-making module calculates Q-values using a convolutional neural network, selects actions based on an ϵ-greedy strategy, and converts actions into target point coordinates, which are then sent to the navigation module. The navigation module plans a global path using the A* algorithm and converts it into motion control commands for the robot's base using the TEB algorithm. The robot moves around in the environment, continuously receives new sensor data, and keeps building new maps, forming a loop. To make the training safer and more stable, the loop is terminated when the decision-making module outputs a next target point that is in an unknown area or too close to obstacles. Similarly, if the robot collides during navigation or fails to find a valid path for an extended period, the loop is also terminated. At this point, the simulation environment is reset, the navigation and mapping nodes are restarted, obstacles in Gazebo return to their original positions, the robot's velocity is reset to zero and returned to the initial position, then a new round of training begins, re-entering the loop.

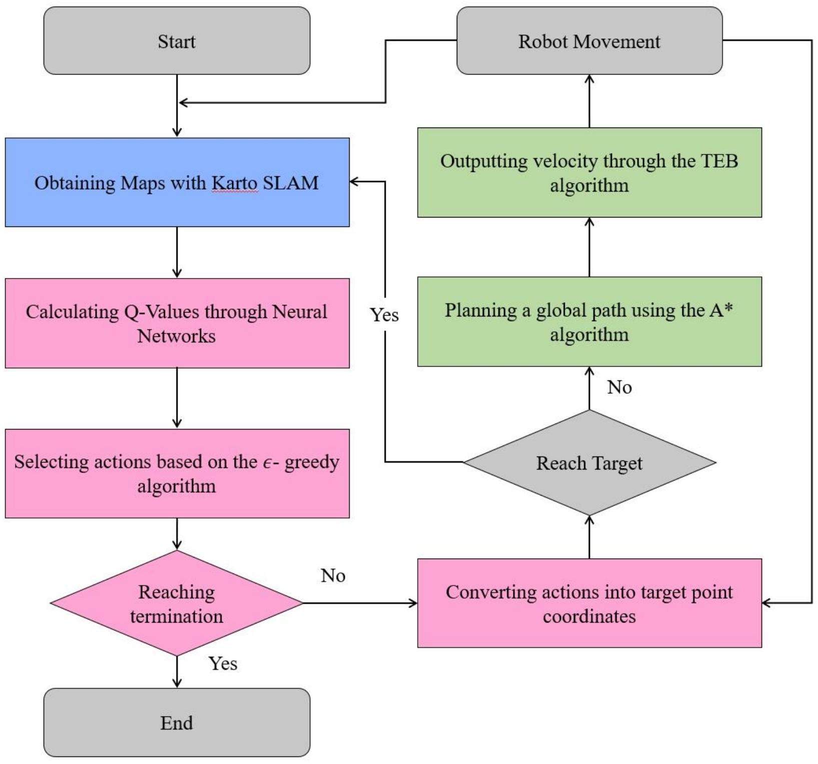

Start
Robot Movement
Obtaining Maps with Karto SLAM
Outputting velocity through the TEB algorithm
Calculating Q-Values through Neural Networks
Yes
Planning a global path using the A* algorithm
No
Selecting actions based on the ϵ- greedy algorithm
Reach Target
No
Reaching termination
Converting actions into target point coordinates
Yes
End

Figure 4.10 Autonomous exploration system flow chart

4.3.2 Model Evaluation Metrics

In this paper, environment exploration rate, average path cost, and exploration efficiency are used as evaluation metrics to measure the effectiveness of the autonomous exploration algorithm. The environment exploration rate measures the completeness of autonomous exploration by defining it as in Equation 4.1, where p_t is the number of known points in the map at time t. l is the true size of the environment map, and T_{SLAM} is the mapping resolution.

$$\rho_t = p_t \frac{l_{\text{width}} \cdot l_{\text{height}}}{T_{\text{SLAM}}^2} \tag{4.1}$$

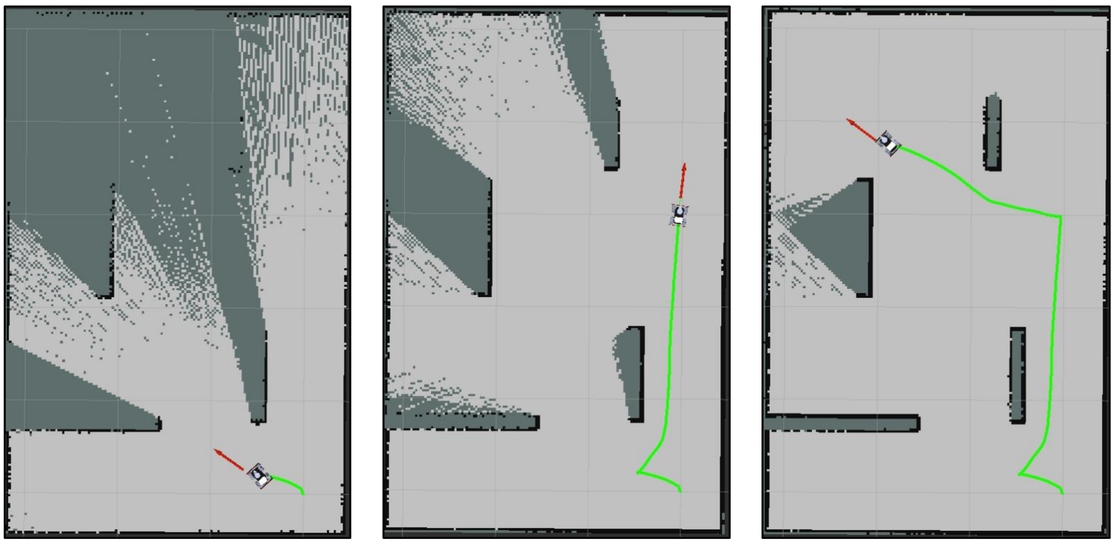
The path cost of the mobile robot was defined in Section 3.3.3, and its expression is given in Equation 3.7. It is not feasible for the mobile robot to explore the environment blindly without considering the burden, and the path cost can be used to measure the effectiveness of the robot's movements. Generally, as the environment exploration rate increases, the path traveled by the robot becomes longer, resulting in higher path cost. There is often a trade-off between environment exploration rate and path cost, and they cannot always be simultaneously satisfied. In this paper, we define exploration efficiency as the reduction in map entropy per unit path cost, which serves as a compromise metric to assess the effectiveness of autonomous exploration. The definition of exploration efficiency is given in Equation 4.2, where *H* represents the map entropy.

$$\eta = \frac{\sum_i (H(T) - H(0))_i}{\sum_i S(x_{i-1}, x_i)} \tag{4.2}$$

4.4 Model Training and Testing

4.4.1 Training of the Simple Model

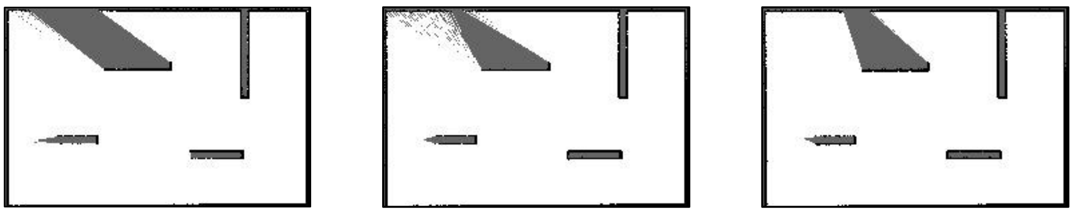
In order to intuitively confirm that the proposed unknown environment exploration system based on improved DQN in this paper can truly solve the problem of autonomous exploration for mobile robots, training was initially conducted in the simple environment shown in Figure 4.3 in Section 4.2.3. Initially, the algorithm frequently selected target points in the unknown area and was unable to complete exploration. After 1000 rounds of training, the mobile robot frequently moved within the area, and the probability of selecting target points in the unknown area decreased significantly. After 3000 rounds of training, it can be observed that the algorithm can successfully complete exploration in the vast majority of cases, and the process of successful exploration is shown in Figure 4.11.



Δs =0.523814737797	Δs =2.041010475918	Δs =3.41800570488
ρ =0.372653792579	ρ =0.75225635480	ρ =0.932908679605
(Step1)	(Step2)	(Step3)

Figure 4.11 Schematic diagram of the training process of a simple map

In the figure, an episode of training is completed in three steps. Δs indicates the path cost, and ρ represents the exploration rate. In the training on a simple map, the exploration is completed when the parameter is set to $\rho > 0.90$. An example of the map constructed when exploration is successfully completed is shown in Figure 4.12.



(a) (b) (c)

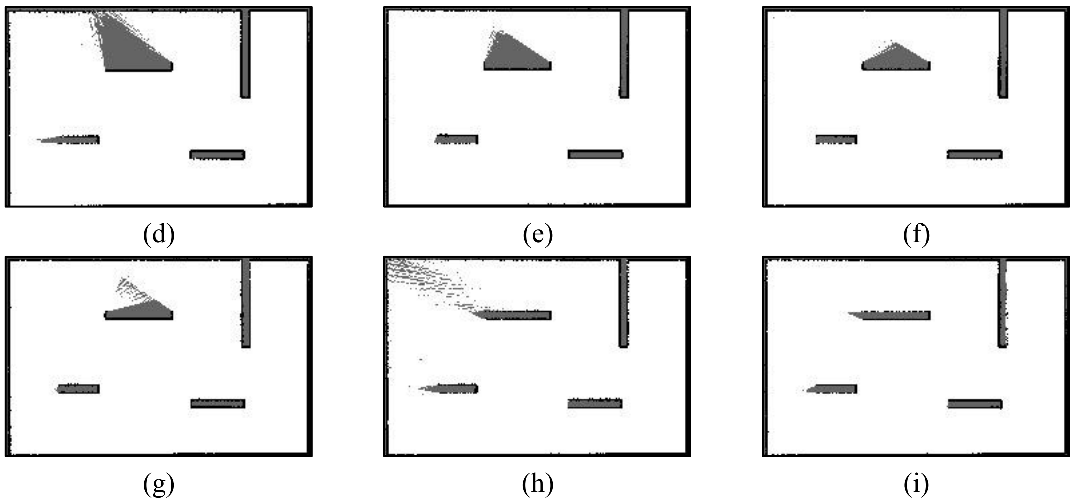


Figure4.12 Map constructed during training on a simple map

The model based on the map entropy reward function tends to converge in a simple environment after about 6000 rounds, with the loss function approaching 0, as shown in Figure 4.13. The model based on the known point reward function tends to converge at around 4500 rounds, converging faster than the former, but with greater fluctuations in the loss function, as shown in Figure 4.14. In the early stages of training, the mobile robot tends to choose a large number of target points within unknown areas, resulting in fewer effective movements and slower convergence of the loss function in the early stages of training.

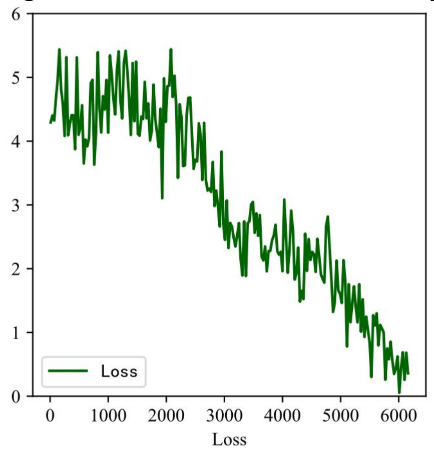


Figure 4.13 The loss function of the map entropy model on a simple map

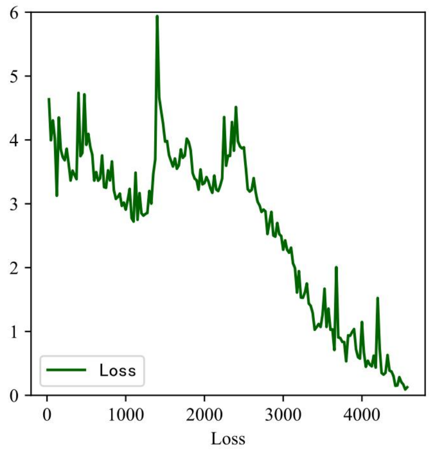


Figure 4.14 The loss function of the known point model on a simple map

4.4.2 Testing of the Simple Model

This paper tested the D3QN PER deep learning model trained in a simple map environment and compared its performance with a boundary-based algorithm in the training environment. The average and variance of environmental exploration rate, path cost, and exploration efficiency for both algorithms are shown in Figure 4.1. By comparing the data, the boundary-based method outperformed in terms of environmental exploration rate and path cost, while the D3QN PER algorithm excelled in terms of exploration efficiency. Additionally, the variance of the three performance parameters for the boundary-based algorithm was significantly lower than that of the D3QN PER algorithm, indicating greater stability. Overall, traditional boundary-based algorithms demonstrated more advantages.

Table 4.1 Test results on simple training map

method		Environment exploration rate	Path cost	Exploration efficiency
D3QN PER algorithm	variance	0.921	3.419	**1634.39**
	average value	0.0357	3.9441	142.4689
Boundary based algorithm	variance	**0.938**	**2.813**	1535.51
	average value	0.0127	0.5733	80.2736

Following this, the paper conducted tests on the model in complex maps to observe its adaptability to complex environmental maps. Unfortunately, the model was unable to explore successfully in complex environments and often became trapped in corners, failing to fully explore the environment. Figures 4.15 and 4.16 depict two typical examples of failed training, where the mobile robot oscillated repeatedly for an extended period, making training ineffective.

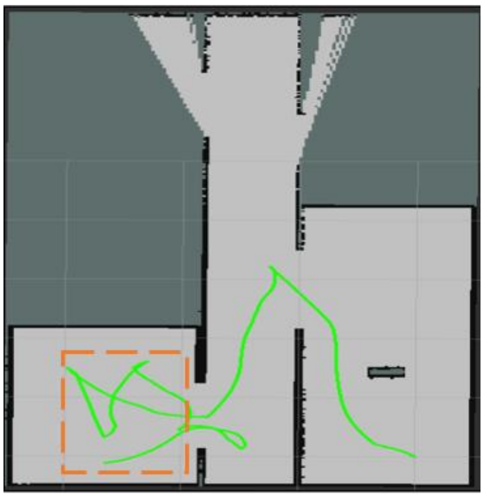


Figure 4.15 The robot vibrates in the lower left corner of the map

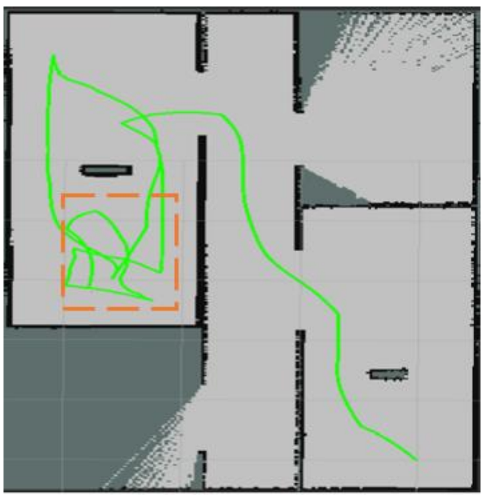


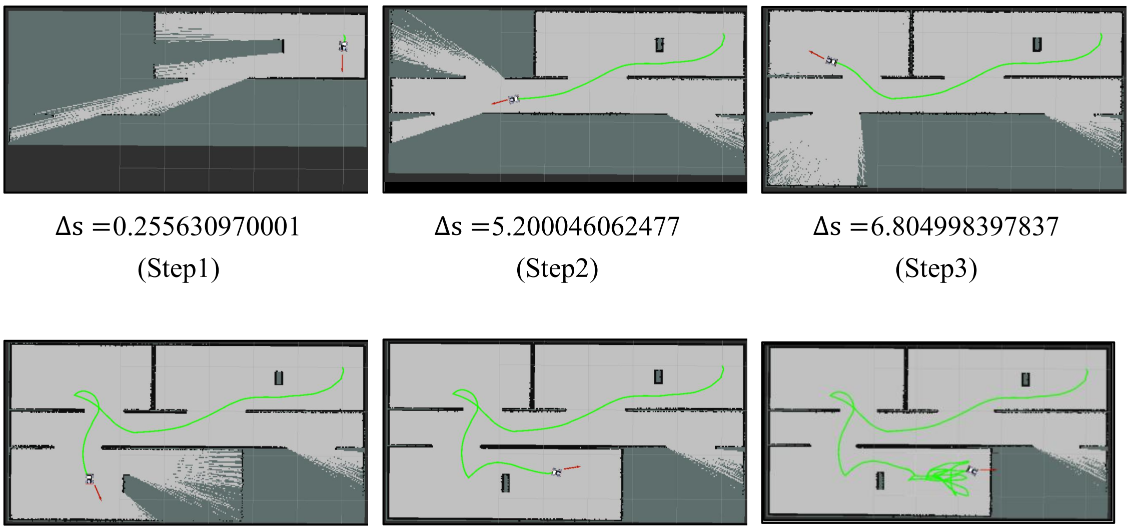
Figure 4.16 The robot vibrates in the upper left corner of the map

Upon analysis, it was determined that the simple maps lack the characteristic of "dead ends." During the training process in simple maps, the mobile robot did not have enough opportunities to learn how to handle "dead end" situations, which resulted in its inability to explore successfully in complex environments. Therefore, it can be inferred that the adaptability of the model to different environments is largely related to the environmental characteristics of the training maps. The more complex the training maps are, with a variety of valuable strategies to learn, the stronger the model's ability to autonomously explore different environments.

4.4.3 Training of the Complex Model

During the training in the complex map shown in Figure 4.4 in Section 4.2.3, it became evident that the model's convergence speed significantly decreased, and several issues were exposed. The most severe problem was that the mobile robot remained stuck in corners during training, performing small-range movements within local regions and continuously oscillating. This continued until the robot selected a next target point very close to its current position, with the positional difference falling within the endpoint error range of the navigation module. In this scenario, navigation was completed without the robot moving, triggering the condition to exit the training loop, and the robot received the lowest single-step reward. A typical example of a failed training instance is shown in Figure 4.17.

Through observation, it was discovered that during the oscillations, the mobile robot received negative rewards for each step because it did not further map construction. Therefore, it is hypothesized in this paper that the reason for the robot's short-distance movements is an attempt to reduce path cost and obtain larger single-step rewards.



Δs =0.255630970001 (Step1)

Δs =5.200046062477 (Step2)

Δs =6.804998397837 (Step3)

Δs =9.310574531562 (Step4)

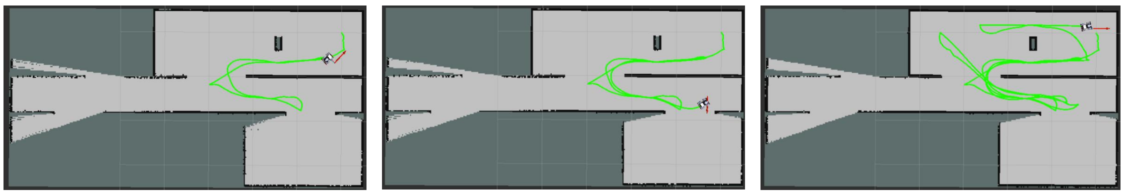
Δs =11.42890930189 (Step5)

局部震荡，探索失败 ρ =0.819472556257

Figure 4.17 Local Oscillation Phenomenon in the Complex Training Map

Based on this observation, this paper adjusted the parameters in Equation 3.7 and the parameters in Equation 3.8 from Section 3.3.3. This was done to reduce the impact of the mobility cost in the reward function, encouraging the mobile robot to make longer-distance movement attempts and increasing the likelihood of the robot escaping from local corners. Following these improvements, the oscillation behavior of the mobile robot was alleviated. Through observation, it was noted that the single-step movement distance of the robot tended to increase compared to before, but this did not result in an overall improvement in training performance.

The issue of the robot oscillating in local corners transformed into a roaming problem. After exploring some areas, the robot began to repeat previously traversed paths within known regions, moving back and forth until it reached the maximum allowable number of steps for a single episode, at which point it would exit the loop. The roaming problem within the known regions is illustrated in Figure 4.18, and under these conditions, the model's convergence became extremely slow, making it unable to successfully complete training.



$\Delta s = 10.5039416162$ (Step4) $\Delta s = 15.177540145$ (Step5) Roaming phenomenon, exploration failure $\rho = 0.63235823742$

Figure 4.18 Roaming Behavior of the Robot in a Complex Training Map

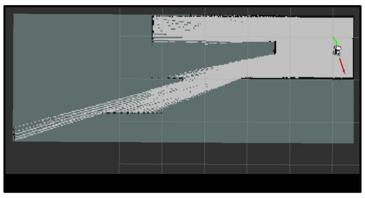
Boundaries represent the intersection between known and unknown areas, and boundary information is crucial for a mobile robot's exploration into unknown regions. Prolonged observations have shown that the roaming problem can occur due to the mobile robot's insufficient sensitivity to boundaries. This paper addresses the issue of insufficient boundary sensitivity from three perspectives.

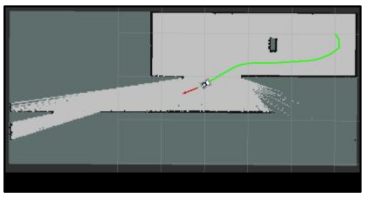
Firstly, as described in detail in Section 3.3.4, boundary information is input to the deep learning network in the form of images for feature extraction. In this regard, the paper deepens the structure of the convolutional layers, leveraging the mature ResNet [46] residual network architecture to enhance the model's extraction of deep-level features from input images, fundamentally solving the problem of inadequate boundary sensitivity.

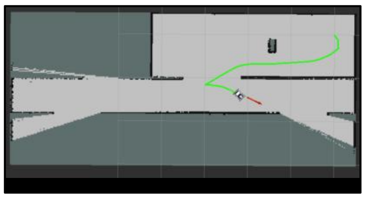
Secondly, target points may fall into the unknown area near the boundary and would receive rewards according to existing rules, breaking the loop and wasting an opportunity for the robot to explore the boundary effectively. This paper modifies the existing rules to allow the robot to navigate to the unknown area within a certain neighborhood range near the boundary.

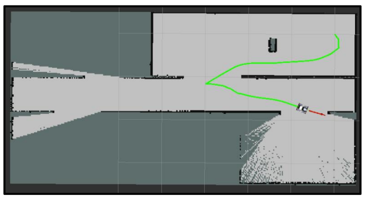
Thirdly, during the robot's roaming, minimal changes occur in the map, leading to the frequent selection of target points near the robot's last location. As a result, the robot moves repeatedly between a few points. This paper inputs the robot's current position and the neighborhood range around the last position into the deep learning network as obstacles to reduce the probability of target points repeating the robot's previous poses.

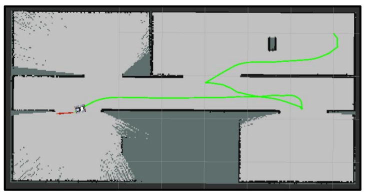
Through extensive parameter tuning and experimentation, the training performance of the model has been improved significantly, leading to a substantial increase in the probability of successful exploration by the mobile robot. Figure 4.19 illustrates the process of a successful exploration by the robot.


Δs =0.335100650787
(Step1)


Δs =3.61359786987
(Step2)


Δs =4.446269512182
(Step3)


Δs =5.937387466432
(Step4)


Δs =11.45266056068
(Step5)

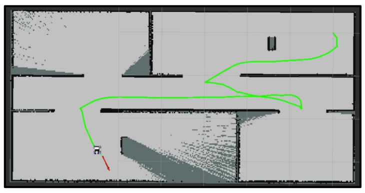
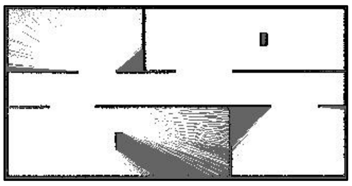
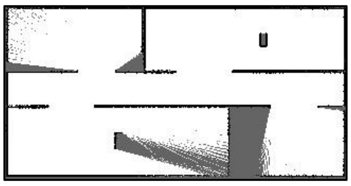
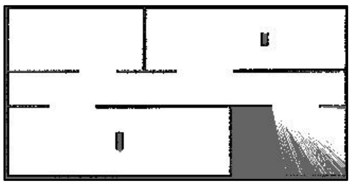

Δs =12.51754665378
(Step6)

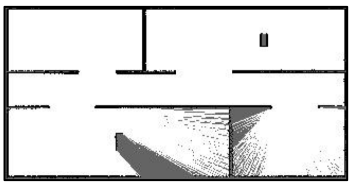
Figure 4.19 Example of successful exploration in complex training environment

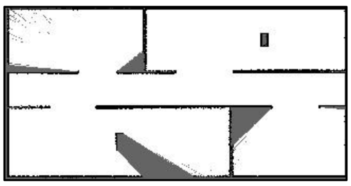
The map constructed when the mobile robot successfully completes exploration in a complex training environment is illustrated in Figure 4.12. Here, Δs refers to the cost of travel distance, ρ represents the exploration rate, and in the context of a complex training environment, $\rho > 0.85$ is considered as achieving exploration completion.

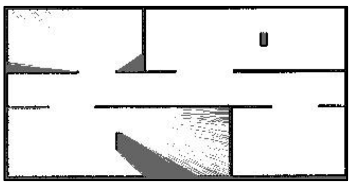

Δs =22.6640652882
ρ =0.850041671628
(a)

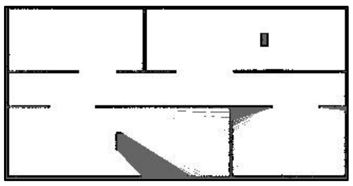

Δs =12.0262676978
ρ =0.863257530658
(b)

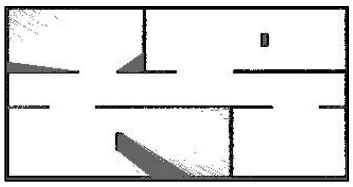

Δs =17.0564537272
ρ =0.873258721276
(c)


Δs =13.9339936272
ρ =0.904512441957
(d)


Δs =6.51231896241
ρ =0.918502202643
(e)


Δs =20.0224306865
ρ =0.919037980712
(f)


$\Delta s = 18.9558915319$
$\rho = 0.938861769258$
(g)


$\Delta s = 14.7221078873$
$\rho = 0.9499126118$
(h)

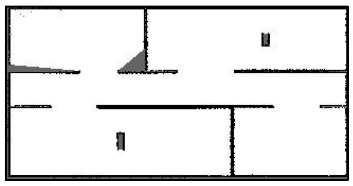

$\Delta s = 17.4604082548$
$\boldsymbol{\rho} = \mathbf{0.97490772711}$
(i)

Figure 4.20 Map constructed during complex map training process

Among them, (e) represents the instance during the training process when the robot achieved the minimum cost of movement for a successful exploration. The robot's movement trajectory is shown in Figure 4.21, with a concise and clear purpose. The exploration effectiveness, to some extent, exceeds human-operated levels. This is the ultimate goal expected to be achieved by the unknown environment exploration system presented in this paper. Based on this trajectory, various model parameters were further fine-tuned in this study.

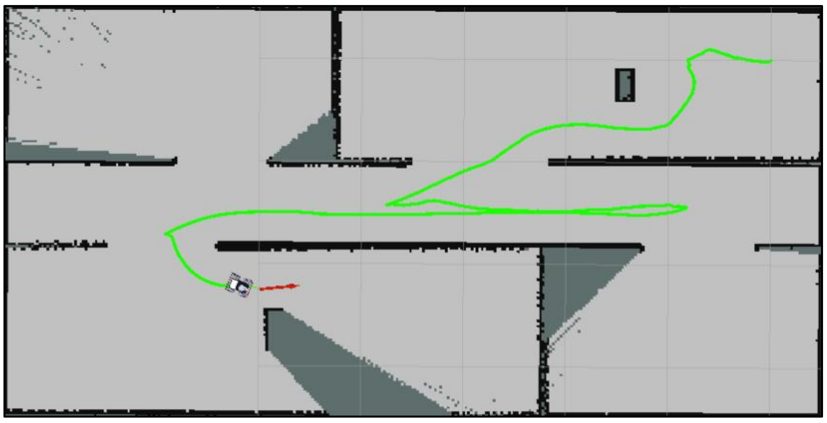


Figure 4.21 Exploration trajectory of shortest path cost

4.4.4 Testing the Complex Model

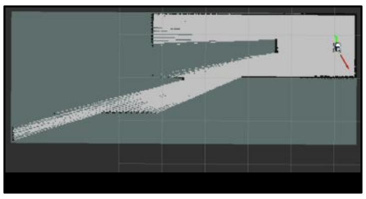
(1) Testing D3QN PER Model on the Training Map

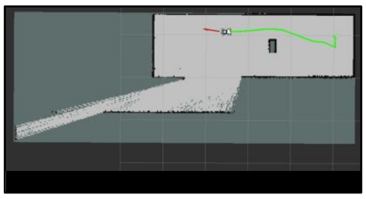
In this section, the D3QN PER deep learning model trained in a complex map environment is tested and compared side by side with boundary-based algorithms in the training environment to evaluate their performance. The average and variance of environment exploration rate, path cost, and exploration efficiency for both algorithms are shown in Figure 4.2. By comparing the data, it is evident that the boundary-based method outperforms in terms of environment exploration rate and path cost, while the D3QN PER algorithm excels in exploration efficiency. Additionally, the variance of performance parameters for the boundary-based algorithm is significantly lower than that of the D3QN PER algorithm, making it more stable. It's worth noting that both algorithms have path costs much higher than the trajectory shown in Figure 4.21.

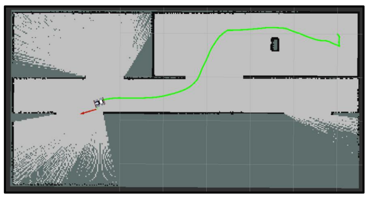
Table 4.2 Test results on complex training maps

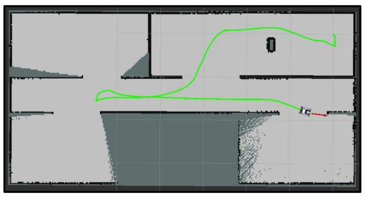
method		Environment exploration rate	Path cost	Exploration efficiency
D3QN PER algorithm	average value	0.909	16.419	**1124.39**
	variance	0.0610	6.2620	330.8614
Boundary based algorithm	average value	**0.921**	**14.813**	926.98
	variance	0.0192	2.5733	180.2736

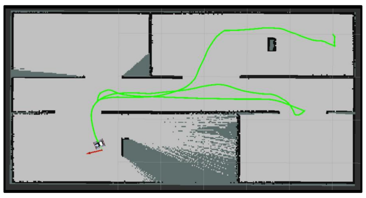
The testing results of the D3QN PER algorithm on the complex training map are depicted in Figure 4.22, where the green line represents the trajectory of the robot.


Δs =0.292018443346
(Step1)


Δs =2.940917491917
(Step2)


Δs =6.680819988251
(Step3)


Δs =11.72324371342
(Step4)


Δs =15.93091120178
(Step5)

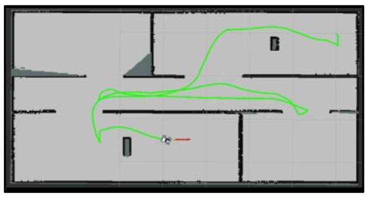

Δs =16.78376007088
ρ =0.919966662698

Figure 4.22 Test results of D3QN PER model

(2) Generalization Testing of the D3QN PER Model

In this paper, we employed the modeling approach outlined in Section 4.2.3 to create multiple test maps, as shown in Figure 4.23. These maps were used to evaluate the performance and effectiveness of the D3QN PER model in various test map scenarios. Given that the D3QN PER model includes fully connected layers, the size of the action space was fixed during training, meaning that the model can adapt to a maximum map size.

For smaller maps with different shapes, the mapping module filled the maps to match the dimensions of the largest map using a padding approach. The regions that were filled in the process were treated as obstacles during navigation, effectively addressing the adaptation issue of the D3QN PER model to maps with varying shapes.

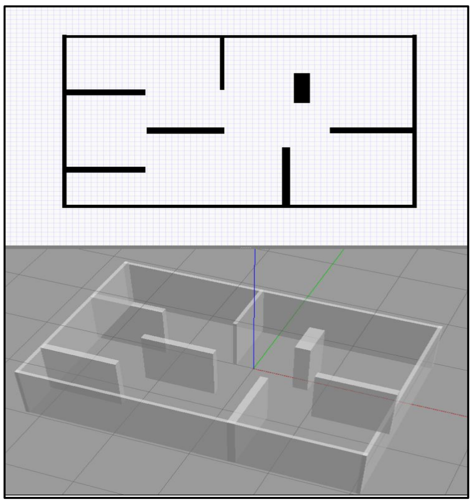


Figure 4.23 Example diagram of test environment

In order to validate the model's generalization capability, this paper tested and conducted a comparative analysis of the D3QN PER deep learning model trained in complex map environments and a boundary-based algorithm on a test map. Taking the test map shown in Figure 4.23 as an example, the mobile robot starts from different random initial points and explores the environment. After multiple random tests, the average values and variances of the environment exploration rate, path cost, and exploration efficiency for both algorithms are shown in Figure 4.3.

It can be observed that the D3QN PER model achieved an average environment exploration rate of 84.9% in unfamiliar map environments without any modifications, which generally meets the requirements for autonomous robot exploration. Through cross-data comparison, the boundary-based method outperforms in terms of environment exploration rate, path cost, and exploration efficiency, exhibiting lower variance and greater stability. The D3QN PER algorithm, when transferred to the test map, cannot be directly compared with the performance of the boundary-based algorithm and can only achieve preliminary autonomous exploration tasks.

Table 4.3 Test results on the test map

method		Environment exploration rate	Path cost	Exploration efficiency
D3QN PER algorithm	average value	0.849	21.274	694.93
	variance	0.0312	5.9253	284.8621
Boundary based algorithm	average value	0.917	19.823	913.21
	variance	0.0120	2.1373	190.2810

4.4 Conclusion of This Chapter.

This section represents the experimental validation component of the overall thesis. Firstly, the chapter establishes a simulation environment for the autonomous exploration problem of mobile robots using the ROS (Robot Operating System) and Gazebo simulation software. Subsequently, the chapter outlines the general workflow and implementation details of the system within the simulation environment, defining three performance evaluation metrics: environment exploration rate, path cost, and exploration efficiency.

The chapter then begins with the training of simple models, gradually addressing issues encountered during the training process. It covers training and testing of both simple and complex models. Through an extensive series of experiments, the study demonstrates that the proposed autonomous exploration system based on improved DQN exhibits a degree of generalization, capable of autonomously exploring environments without prior knowledge.

This chapter's experimental validation provides robust empirical support for the research, confirming the effectiveness and applicability of the proposed methods.

Chapter 5
Conclusion and Future Outlook

5.1 Conclusion

A. We have successfully established a modular and versatile simulation environment based on ROS and Gazebo. The navigation, mapping, and decision-making modules within this environment can not only be applied to address autonomous exploration challenges but also allow for algorithm modifications, recombination, and transfer to tackle various other problems.

B. In this study, we delved into the fundamental theories related to deep reinforcement learning, including principles of convolutional neural networks, temporal difference algorithms, and DQN within the realm of deep reinforcement learning. This solid theoretical foundation serves as a basis for future exploration.

C. The proposed autonomous exploration algorithm based on improved DQN can autonomously explore environments without the need for complex rule-based configurations and demonstrates effective performance across a wide range of map environments.

D. Our method based on improved DQN exhibits good adaptability to different maps, reducing the heavy reliance on specific map environments that traditional algorithms often require, highlighting its versatility.A.

5.2 Future Outlook

A. While theoretically, deep reinforcement learning methods can address autonomous exploration in complex environments, practical challenges such as time-consuming training, convergence issues, and suboptimal model performance still persist.

B. The modular system structure in this paper has not entirely eliminated the dependence on the robot model. When transferring the system model between different robots, such as changing from a holonomic model to an Ackermann model, discrepancies in path planning by the navigation module may lead to variations in exploration effectiveness.

C. The algorithm presented in this paper makes decisions only when reaching a target point and acquiring a new environmental map. This leads to discontinuous robot movement and potential stuttering when exploring, as it lacks continuous decision-making during movement.

D. This paper focused on implementing the improved DQN algorithm based on convolutional neural networks without extensive exploration of deep learning model architectures. Modifying model structures or using alternative deep learning models like fully convolutional networks may potentially result in faster convergence and improved model performance

.

References

[1] Yamauchi B. A frontier-based approach for autonomous exploration[C]//Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97.'Towards New Computational Principles for Robotics and Automation'. IEEE, 1997: 146-151.

[2] Freda L, Oriolo G. Frontier-based probabilistic strategies for sensor-based exploration[C]//Proceedings of the 2005 IEEE International Conference on Robotics and Automation. IEEE, 2005: 3881-3887.

[3] Wettach J, Berns K. Dynamic Frontier Based Exploration with a Mobile Indoor Robot[C]//ISR/ROBOTIK. 2010: 1-8.

[4] Mobarhani A, Nazari S, Tamjidi A H, et al. Histogram based frontier exploration[C]//2011 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2011: 1128-1133.

[5] Keidar M, Kaminka G A. Efficient frontier detection for robot exploration[J]. The International Journal of Robotics Research, 2014, 33(2): 215-236.

[6] Umari H, Mukhopadhyay S. Autonomous robotic exploration based on multiple rapidly-exploring randomized trees[C]//2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2017: 1396-1402.

[7] Bircher A, Kamel M, Alexis K, et al. Receding horizon" next-best-view" planner for 3d exploration[C]//2016 IEEE international conference on robotics and automation (ICRA). IEEE, 2016: 1462-1468.

[8] Qiao W, Fang Z, Si B. Sample-based Frontier detection for autonomous robot exploration[C]//2018 IEEE International Conference on Robotics and Biomimetics (ROBIO). IEEE, 2018: 1165-1170.

[9] Gao W, Booker M, Adiwahono A, et al. An improved frontier-based approach for autonomous exploration[C]//2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV). IEEE, 2018: 292-297.

[10] Fang B, Ding J, Wang Z. Autonomous robotic exploration based on frontier point optimization and multistep path planning[J]. IEEE Access, 2019, 7: 46104-46113.

[11] Bourgault F, Makarenko A A, Williams S B, et al. Information based adaptive robotic exploration[C]//IEEE/RSJ international conference on intelligent robots and systems. IEEE, 2002, 1: 540-545.

[12] Julian B J, Karaman S, Rus D. On mutual information-based control of range sensing robots for mapping applications[J]. The International Journal of Robotics Research, 2014, 33(10): 1375-1392.

[13] Francis G, Ott L, Marchant R, et al. Occupancy map building through bayesian exploration[J]. The International Journal of Robotics Research, 2019, 38(7): 769-792.

[14] González-Banos H H, Latombe J C. Navigation strategies for exploring indoor

environments[J]. The International Journal of Robotics Research, 2002, 21(10-11): 829-848.

[15] Ila V, Porta J M, Andrade-Cetto J. Information-based compact pose SLAM[J]. IEEE Transactions on Robotics, 2009, 26(1): 78-93.

[16] Juliá M, Gil A, Reinoso O. A comparison of path planning strategies for autonomous exploration and mapping of unknown environments[J]. Autonomous Robots, 2012, 33(4): 427-444.

[17] Monahan G E. State of the art—a survey of partially observable Markov decision processes: theory, models, and algorithms[J]. Management science, 1982, 28(1): 1-16.

[18] Bai S, Chen F, Englot B. Toward autonomous mapping and exploration for mobile robots through deep supervised learning[C]//2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2017: 2379-2384.

[19] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[J]. Advances in neural information processing systems, 2012, 25.

[20] Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning[J]. arXiv preprint arXiv:1312.5602, 2013.

[21] Silver D, Huang A, Maddison C J, et al. Mastering the game of Go with deep neural networks and tree search[J]. nature, 2016, 529(7587): 484-489.

[22] Tai L, Liu M. Mobile robots exploration through cnn-based reinforcement learning[J]. Robotics and biomimetics, 2016, 3(1): 1-8.

[23] Mirowski P, Pascanu R, Viola F, et al. Learning to navigate in complex environments[J]. arXiv preprint arXiv:1611.03673, 2016.

[24] Niroui F, Zhang K, Kashino Z, et al. Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments[J]. IEEE Robotics and Automation Letters, 2019, 4(2): 610-617.

[25] Pathak D, Agrawal P, Efros A A, et al. Curiosity-driven exploration by self-supervised prediction[C]//International conference on machine learning. PMLR, 2017: 2778-2787.

[26] Li H, Zhang Q, Zhao D. Deep reinforcement learning-based automatic exploration for navigation in unknown environment[J]. IEEE transactions on neural networks and learning systems, 2019, 31(6): 2064-2076.

[27] O’Neill J, Pleydell-Bouverie B, Dupret D, et al. Play it again: reactivation of waking experience and memory[J]. Trends in neurosciences, 2010, 33(5): 220-229.

[28] Van Hasselt H, Guez A, Silver D. Deep reinforcement learning with double q-learning[C]//Proceedings of the AAAI conference on artificial intelligence. 2016, 30(1).

[29] Wang Z, Schaul T, Hessel M, et al. Dueling network architectures for deep reinforcement learning[C]//International conference on machine learning. PMLR, 2016: 1995-2003.

[30] Schaul T, Quan J, Antonoglou I, et al. Prioritized experience replay[J]. arXiv preprint arXiv:1511.05952, 2015.

[31] Dong H, Ding Z, et al. Deep Reinforcement Learning[M]. Springer Singapore, 2020.

[32] Long J, Shelhamer E, Darrell T. Fully convolutional networks for semantic segmentation[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2015: 3431-3440.

[33] Montemerlo M, Thrun S, Koller D, et al. FastSLAM: A factored solution to the simultaneous localization and mapping problem[J]. Aaai/iaai, 2002, 593598.

[34] Konolige K, Grisetti G, Kümmerle R, et al. Efficient sparse pose adjustment for 2D mapping[C]//2010 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2010: 22-29.

[35] Kümmerle R, Grisetti G, Strasdat H, et al. g 2 o: A general framework for graph optimization[C]//2011 IEEE International Conference on Robotics and Automation. IEEE, 2011: 3607-3613.

[36] Yao J, Lin C, Xie X, et al. Path planning for virtual human motion using improved A* star algorithm[C]//2010 Seventh international conference on information technology: new generations. IEEE, 2010: 1154-1158.

[37] Rösmann C, Feiten W, Wösch T, et al. Trajectory modification considering dynamic constraints of autonomous robots[C]//ROBOTIK 2012; 7th German Conference on Robotics. VDE, 2012: 1-6.

[38] Amidi O, Thorpe C E. Integrated mobile robot control[C]//Mobile Robots V. International Society for Optics and Photonics, 1991, 1388: 504-523.

[39] Quinlan S, Khatib O. Elastic bands: Connecting path planning and control[C]//[1993] Proceedings IEEE International Conference on Robotics and Automation. IEEE, 1993: 802-807.

[40] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv:1409.1556, 2014.

[41] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

[42] Macenski S, Foote T, Gerkey B, et al. Robot Operating System 2: Design, architecture, and uses in the wild[J]. Science Robotics, 2022, 7(66): eabm6074.