Research on path planning of mobile robot based on improved Deep Q Network

Meng Guan\textsuperscript{1a*}, Fu Xing Yang\textsuperscript{2b}, Ji Chao Jiao\textsuperscript{3c}, Xin Ping Chen\textsuperscript{4d}

\textsuperscript{1}Modern Post College, Beijing University of Posts and Telecommunications, Beijing, China
\textsuperscript{2}School of Artificial Intelligence, Beijing University of Posts and Telecommunications, Beijing, China
\textsuperscript{3}School of Electronic Engineering, Beijing University of Posts and Telecommunications, Beijing, China
\textsuperscript{4}School of Electronic Engineering, Beijing University of Posts and Telecommunications, Beijing, China

bemail: yangfx@bupt.edu.cn, cemail: jiaojichao@gmail.com, demail: xingping.555@bupt.edu.cn

*Correspondence author: aemail: guanmeng@bupt.edu.cn

Abstract: In order to solve the problem of slow convergence and low learning efficiency when mobile robots use ordinary DQN algorithm to plan path in an unknown environment with insufficient prior knowledge, an improved DQN algorithm is proposed. A heuristic reward function is designed to realize continuous reward and solve the problem of slow convergence of DQN algorithm caused by reward sparsity. An adaptive exploration strategy is proposed to solve the problem of balance between exploration and utilization in reinforcement learning process, and improve the efficiency of exploration. Experiments show that the improved DQN algorithm not only has high learning efficiency and fast convergence, but also the planned path is optimal.

1. Introduction
Autonomous navigation of mobile robots has always been a research hot at home and abroad. Path planning is one of the key technologies for autonomous navigation of mobile robots. Its purpose is to find a feasible path from the starting point to the target point as short as possible and avoid obstacles. According to the characteristics and time sequence of the algorithm, path planning algorithms can be divided into traditional path planning algorithms, intelligent bionic algorithms, and path planning algorithms based on reinforcement learning. Common traditional algorithms include dijkstra algorithm, A* algorithm, artificial potential field method, and common intelligent bionic algorithms include ant colony algorithm and genetic algorithm. On the global path planning with a known environment, both traditional algorithms and intelligent bionic algorithms can achieve good path planning results, but the actual road conditions are very complicated. Under this conditions with insufficient prior knowledge, we hope unmanned vehicles can learn autonomously, avoid obstacles and search for suitable paths. Existing traditional algorithms and intelligent bionics are not suitable for obstacle avoidance and path planning in the local environment with insufficient prior knowledge due to their own limitations.
Reinforcement learning is an unsupervised and constantly reactivating algorithm. Different from supervised learning methods, the agent and the environment interact with each other through trial and error to learn online to obtain feedback. According to the feedback, the strategy and action with the greatest reward are selected, without relying on the environment model and prior knowledge, suitable for solving the problem of local path planning of unmanned vehicles in unknown environments. The most commonly used reinforcement learning method in path planning is the Q_learning algorithm. When updating, the strategy that maximizes the Q value is selected. When the traditional Q_learning algorithm initializes the Q value, it is generally set to the mean or random value, so at the beginning, when learning, the convergence speed of the algorithm is very slow. Liu X Y[5] used the combination of e_greedy and Boltzmann exploration to improve the Q_learning algorithm. This algorithm solves the learning problem of the optimal path in a static and non-deterministic environment, considering the step size limit. But there is still the problem of slow convergence. Dong P F[6] added the initial gravitational potential field to improve the Q learning algorithm, initialized the Q value, and improved the path planning speed to a certain extent, but the convergence speed needs to be improved, and the planned path may not be optimal. When searching for paths in a complex environment, the Q_learning algorithm still has major shortcomings. There is a problem of dimensionality disaster, because the Q value needs to be stored in a table. When the environment is complex and there are many states, a large amount of memory is required to store it, which takes up a lot of memory resources of the computer, the value query will also be very slow, resulting in low efficiency. So Deep Q learning Network (DQN) was produced. Volodymyr M[7] used a convolutional neural network to fit the value function. The input is the original pixel and the output is the value function for estimating the future return. This solves the problem of dimensionality disaster, but DQN is when finding a path in a complex unknown environment, there is also the problem of slow convergence.

In response to the above problems, this paper proposes an improved DQN path planning method. Based on the heuristic function of A* algorithm and the idea of artificial potential field method, a continuous heuristic reward function is designed, which greatly accelerates the learning efficiency of mobile robots. DQN with heuristic reward greatly speeds up the learning efficiency of mobile robot and improves the convergence of the algorithm. The improved e-greedy adaptive exploration strategy solves the problem of balance between exploration and utilization, and improves the exploration efficiency.

2. Improved DQN mobile robot path planning model

2.1 DQN algorithm with target network

The Q_learning algorithm takes action \( a \) based on the current state to interact with the environment, and evaluates the feedback value, evaluates the action, uses the \( Q(s, a) \) of state-action \((s, a)\) as the action evaluation function, and stores the Q values of different state-action pairs to form a Q table. Repeating the actions in all states, updating the Q table, and then selecting the action that maximizes the Q value in this table to learn the best strategy. The Q value calculation formula is as follows:

\[
Q(s, a) = r + \gamma \max_{\hat{a}} Q(s', \hat{a})
\]

(1)

Among them, \( r \) is the immediate reward value of the current state \( s \) and action \( a \), and \( \gamma \) is the discount factor. \( Q(\hat{s}, \hat{a}) \) which represents the Q value generated by executing possible future actions in the current state \( s \) to enter the next state. The update formula of Q value is:

\[
Q(s, a) \leftarrow Q(s, a) + \alpha \left[ r + \gamma \max_{\hat{a}} Q(s', \hat{a}) - Q(s, a) \right]
\]

(2)

Among them, \( \alpha \) is the learning rate, \( \gamma \) is the attenuation factor, and \( Q(s, a) \) is the evaluation function for performing action \( a \) in state \( s \).

The deep Q learning algorithm is based on the Q learning algorithm by adding a neural network to fit the value function, input state and action, output Q value, and continuously train to obtain a neural network with good parameters. Then we only need to input the corresponding and you can find the optimal action you want.

There is a problem in using neural networks to estimate the Q value: the nonlinearity of the deep
network causes the algorithm to not converge and become unstable during the training process. Generally, the empirical playback mechanism and the DQN algorithm with the target network are used to solve the problem. The DQN with the target network is in the original add a neural network with exactly the same structure to the neural network. The original neural network is called the evaluation network, and the new network is called the target network. During the learning process, the evaluation network parameters are updated, and the target network parameters are not updated, and a certain number of updates are completed, then the parameters of the target network are updated to reduce the correlation between calculating the target Q value and updating the Q network parameters. The return value during the time period when the target network parameters are not changed is relatively fixed, which increases the stability of learning. The DQN algorithm with target network is shown in Figure 1:

![DQN with target network](image)

**2.2 Adaptive exploration strategy design**

An improved $\varepsilon$–greedy greedy strategy was proposed to solve the problem of exploration and utilization. Generally, the greedy strategy parameters $\varepsilon$ are set to specific values. During the reinforcement learning process, if the mobile robot can quickly find the target and gradually converge, then the knowledge of the path to the target point should be used more to reduce the remaining “useless” environment exploration. If the environment is too complicated to find the target point, the mobile robot learns bad knowledge. At this time, it is necessary to encourage the mobile robot to explore the unknown environment and try to find the target point. So the parameters also need to be adjusted adaptively. Zhao Y[4] use information entropy to define a measure of state importance, which measures the degree of association between states and goals, which is used to adaptively adjust the balance between exploration and utilization in the learning process and improve learning efficiency.

$\varepsilon$–greedy policy follows the principle of selecting the best action with a high probability and selecting randomly with a small probability. The action is randomly selected under the probability of less than $\varepsilon$, and the action corresponding to the maximum action value $Q$ is selected under the probability of greater than $\varepsilon$. Assume that the random number between 0 and 1 is $\theta$, The mathematical expression is:

$$A(s) \leftarrow \begin{cases} \arg \max_a Q(s,a), & 0 \leq \theta \leq \varepsilon \\ \text{rand}(A), & \varepsilon < \theta < 1 \end{cases}$$ (3)

In the reinforcement learning, the result of the interaction between the agent and environment is fed back by the reward value. Therefore, the mapping relationship between the reward and the $\varepsilon$ parameter can be established to adaptively adjust the parameter and balance the relationship between exploration and utilization. The improved $\varepsilon$–greedy mathematical expression of the parameter is as follows:

$$\varepsilon = \begin{cases} \varepsilon_0 e^{\Delta R_n}, & \Delta R_n \leq 0 \\ \varepsilon_0 + (1 - \varepsilon_0) e^{\Delta R_n}, & \Delta R_n > 0 \end{cases}$$ (4)

The value $\varepsilon$ shouldn’t be too large or too small, $\varepsilon \in [\varepsilon_{\text{min}}, \varepsilon_{\text{max}}]$ if $\varepsilon < \varepsilon_{\text{min}}$, than $\varepsilon = \varepsilon_{\text{min}}$; if
\[ \varepsilon > \varepsilon_{\text{max}}, \quad \text{than} \quad \varepsilon = \varepsilon_{\text{max}}. \]

\[ \Delta R_n = R_n - R_{n-1} \]

Indicates the difference between the reward value of the \( n \) round and the \( n-1 \) round, \( E \in (0,1) \), which is a constant greater than 0 and less than 1. In this way, as each iteration of learning progresses, the parameters adapt themselves to fit the scale of random exploration by the robot in each learning.

### 2.3 Heuristic reward function design

To solve the problem of sparse rewards, the article designs a heuristic continuous reward. Reinforcement learning has always revolved around rewards. The reward generated by interaction with the environment is an important parameter. The setting quality directly affects the convergence speed of the algorithm. In path planning, when the general reinforcement learning method sets the reward function, most of the settings are sparse values, as shown in equation (5):

\[
R = \begin{cases} 
R_1, & \text{Reach the target point} \\
R_2, & \text{Hit an obstacle} \\
R_3, & \text{Intermediate state}
\end{cases}
\]

These reward values are all a constant. Usually when the agent reaches the target point, it is given a reasonable positive reward \( R_1 > 0 \), when it encounters an obstacle, it is given a negative reward value \( R_2 < 0 \). The intermediate state reward is generally set to \( R_3 = 0 \) or set to Some satisfied \( R_1 < R_3 < R_2 \) constant. The feedback information is relatively small. Most of the reinforcement learning has a slow convergence speed and does not have heuristic functions, it is not smart. Especially when the environment is complex, the agent is difficult to reach the target point, hits a wall everywhere, and receives negative results. The rewards and learning are all bad knowledge, which is difficult to converge and lingers in a certain area. Setting appropriate continuous rewards can speed up the learning process of the agent. Marashi M et al. [9] added digital feedback in the learning process of the agent, analyzed the received environment map, and provided the extracted information to the agent, which improved learning speed, while the disadvantage is that it requires perceptual analysis of the environment, which is more troublesome. Starting from the reward function itself, We designs a continuous reward function covering the intermediate state, sets reasonable and continuous rewards, and solves the problem of sparse rewards.

The A* algorithm [1] is an efficient optimal path optimization algorithm and the most commonly used path search algorithm. The key of the A* algorithm is the cost function, which is used to calculate the priority level of each node:

\[
f(n) = g(n) + h(n)
\]

Among them, \( g(n) \) is the compensation for the distance from the node to the start point, and \( h(n) \) is the estimated compensation for the distance from the node to the end point. Generally speaking, it is expressed by Euclidean distance. \( h(n) \) is the key to heuristic searching. Its setting can adjust the speed and accuracy of the algorithm. \( f(n) \) is the node cost function, which represents the comprehensive priority of each node.

Therefore, we can learn from the idea of A* algorithm and set up a heuristic reward function for reinforcement learning, so that each state (node) also has a comprehensive reward, instead of a sparse reward, solving the problem of convergence difficulties and slow learning.

First, we assume a continuous comprehensive reward function is:

\[
R = \begin{cases} 
R_q, & \text{Reach the target point} \\
R_a, & \text{Hit an obstacle} \\
R_{(a,s)} + R_{(n,s)}, & \text{Intermediate state}
\end{cases}
\]

Among them, \( R_a \) represents the reward received when encountering an obstacle, \( R_q \) represents the reward received when reaching the target point. When the obstacle is encountered or the target point is reached, the agent is in the termination state, so it can be set as a constant. The focus is to set the reward function of the middle state, we set it as a continuous heuristic reward function, which \( R_{(a,s)} \) represents the instant reward of the state \( s \) and \( R_{(n,s)} \) the estimated reward of the state. So below we will focus on how to implement the heuristic reward function of the intermediate state.

For instant rewards \( R_{(a,s)} \), the intermediate state is generally set to zero. In order to better
encourage the robot to explore and find the target point faster, we set a stage reward, and the value is determined by the Manhattan distance between the target point and the agent $r_g$. When $r_g$ a certain threshold is reached, we give stage reward:

$$R_{(a,s)} = \begin{cases} m_1, & 0 < r_g \leq d_1 \\ m_2, & d_1 < r_g \leq d_2 = 2d_1 \\ \vdots \\ m_n, & d_{n-1} < r_g \leq d_n = 2d_{n-1} \\ 0, & d_n < r_g \end{cases}$$  \hspace{1cm} (8)

$d_1$ is a reasonable distance threshold, $\rho_a = d_n$. If the grid environment is $x \times x$, it is set by experience $\frac{1}{4}x$, if it is a decimal, it can be rounded up.

For heuristic rewards $R_{(n,s)}$, we use artificial potential field method to judge each state and give a comprehensive reward.

The artificial potential field method [2] is a simple and effective method in robot path planning. Its basic idea is to regard the movement of an object as an electric charge. The movement is driven by the gravitational potential field and the repulsive potential field. An attraction, the obstacle produces a repulsive force on the robot within the range of its repulsive force, and the resultant force affects the movement of the object. Design heuristic reward function based on artificial potential field method. We hope the greater the reward when the agent is close to the target point, and the closer to the obstacle, the smaller the expected reward. In traditional artificial potential field methods, the closer to the target point, the greater the gravity, and the closer to the obstacle, the greater the repulsion. Therefore, it is necessary to transform the artificial potential field method and design a reward function suitable for reinforcement learning.

The gravity reward function is defined as:

$$R_{\text{att}} = \begin{cases} \delta, & \frac{1}{r_{ag}} \leq \rho_a \\ 0, & r_{ag} > \rho_a \end{cases}$$  \hspace{1cm} (9)

Among them, $\delta$ is the reward scale factor, $\rho_a$ is the gravitational influence distance of the target point, and $r_{ag}$ is the Manhattan distance between the current state of the agent and the target point. In the actual situation, $r_{ag}$ should be expressed by Euclidean distance, measured by the lidar, infrared equipment or camera of the mobile robot. For the convenience of research, the simulation experiment is conducted in a grid environment. The agent action space is set up, down, left, and right, so $r_{ag}$ is expressed by Manhattan distance, recorded as:

$$r_{ag} = |q_{ax} - q_{goals}| + |q_{ay} - q_{goals}|$$  \hspace{1cm} (10)

The repulsion reward (penalty) function is:

$$R_{\text{req}} = \begin{cases} -\beta \frac{1}{r_{io}^2} \left( \frac{1}{r_{io}} - \frac{1}{\rho} \right), & r_{io} \leq \rho \\ 0, & r_{io} > \rho \end{cases}$$  \hspace{1cm} (11)

Among them, $\beta$ is the penalty scale factor, $\rho$ representing the distance of the obstacle's influence, which is the obstacle distance in the agent's field of view that can affect the agent's path. $r_{io}$ represents the Manhattan distance between the current state of the agent and an obstacle within the radius of influence, which is recorded as:

$$r_{io} = |q_{ax} - q_{iax}| + |q_{ay} - q_{iay}|$$  \hspace{1cm} (12)

The repulsion function of all obstacles is $\sum R_{\text{req}}$, then $R_{(n,s)} = R_{\text{att}} + \sum R_{\text{req}}$.

The heuristic reward function in other states of the design is:

$$R = R_{\text{att}} + \sum R_{\text{req}} + R_{(a,s)}$$  \hspace{1cm} (13)

Finally, the reward function we set is:

$$R = \begin{cases} R_q, & \text{Reach the target point} \\ R_o, & \text{Hit an obstacle} \\ R_{\text{att}} + \sum R_{\text{req}} + R_{(a,s)}, & \text{Intermediate state} \end{cases}$$  \hspace{1cm} (14)

So we get the heuristic reward function.
2.4 Action space design and network parameter update

The movable action space of the mobile robot is set to four actions, namely A={up, down, left, right}. In order to balance exploration and utilization, an adaptive $\epsilon$-greedy strategy is used to select actions.

The parameter update of the target network and the evaluation network need to set a certain time interval, and the evaluation network parameters $W$ are updated in real time. The parameter update standard of the target network is: when the number of training steps reaches 200 steps and it is updated once after 5 rounds, the network will be evaluated. The parameter is copied to the target parameter, that is $w \leftarrow W$. The following experiment will verify the improved DQN path planning model based on python.

3. Experiment and analysis

This paper conducts experiments on ordinary DQN, DQN with adaptive exploration strategy, DQN with improved heuristic reward, DQN path planning algorithm with adaptive exploration strategy and heuristic reward, which are abbreviated as: Algorithm 1, Algorithm 2, Algorithm 3, Algorithm 4. The rewards and exploration strategies corresponding to each algorithm are shown in Table 1.

| Algorithm                        | Reward      | Explore strategies |
|----------------------------------|-------------|--------------------|
| Algorithm 1: Ordinary DQN       | Sparse      | $\epsilon$-greedy |
| Algorithm 2: DQN with adaptive exploration strategy | Sparse      | Improve $\epsilon$-greedy |
| Algorithm 3: DQN with heuristic reward function | Continuous  | $\epsilon$-greedy |
| Algorithm 4: DQN with heuristic reward and adaptive exploration strategy | Continuous  | Improve $\epsilon$-greedy |

3.1 Environment construction

Rasterize the explored real environment, and create a simulation environment based on the Tkinter of python module. As shown in Figure 2, the red grid represents the agent, the blue grid represents the obstacle, the yellow circle represents the target point, and the white part is the passable area. We need to plan the path from the starting point in the upper left corner to the yellow target in the lower right corner.

![Figure 2. Example of grid environment map](image)

3.2 Experiment comparison in simulation environment

Experiment in a grid environment of 10*10 and 25*25. Each grid has a width of 20. When it hits an obstacle or reaches the target point, the round ends. The agent returns to the starting point and starts the next round of exploration. The maximum number of round iterations ends the exploration. The parameter settings are shown in Table 2:

| Parameters                  | Values |
|-----------------------------|--------|
| Maximum number of iterations M | 500/3000 |
Discount factor $\gamma$ \hspace{1cm} 0.9
Learning rate $\alpha$ \hspace{1cm} 0.01
Reward scale factor $\delta$ \hspace{1cm} 100
Penalty scale factor $\beta$ \hspace{1cm} 100
Obstacle influence radius $\rho$ \hspace{1cm} 40
Target point reward \hspace{1cm} 10
Obstacle penalty \hspace{1cm} -1

Heuristic reward function settings:

In a 10*10 grid environment, set the number of phase intervals to 3, $d_1 = 40$ and $m=1$, then the heuristic reward function is shown in equation 15.

$$R = \begin{cases} 
\delta \frac{1}{r_g} + \sum R_{req} + 3m & 0 < r_g \leq 40 \\
\delta \frac{1}{r_g} + \sum R_{req} + 2m & 40 < r_g \leq 80 \\
\delta \frac{1}{r_g} + \sum R_{req} + m & 80 < r_g \leq 160 \\
0 & 0160 < r_g 
\end{cases} \tag{15}$$

$$R_{req} = \left\{ \begin{array}{c}
-\beta \frac{1}{r_o} \left( \frac{1}{r_o} - \frac{1}{\rho} \right), r_o \leq \rho = 40 \\
0, & r_o > \rho = 40
\end{array} \right. \tag{16}$$

In a 25*25 environment, set the number of phase intervals to 4, $d_1 = 40$ and $m=1$, then the heuristic reward function is shown in equation 17.

$$R = \begin{cases} 
\delta \frac{1}{r_g} + \sum R_{req} + 4m & 0 < r_g \leq 50 \\
\delta \frac{1}{r_g} + \sum R_{req} + 3m50 < r_g \leq 100 \\
\delta \frac{1}{r_g} + \sum R_{req} + 2m & 100 < r_g \leq 200 \\
\delta \frac{1}{r_g} + \sum R_{req} + m & 200 < r_g \leq 400 \\
0 & 0400 < r_g
\end{cases} \tag{17}$$

$\sum R_{req}$ Calculation is the same as equation 16.

Adaptive exploration strategy setting: set $\varepsilon_0 = 0.9$, $E=0.99$, the adaptive exploration strategy is shown in equation 18.

$$\varepsilon = \begin{cases} 
0.9 \times 0.9^{|\Delta R_n|}, & \Delta R_n \leq 0 \\
0.9 + 0.1 \times 0.9^{|\Delta R_n|}, & \Delta R_n > 0
\end{cases} \tag{18}$$

Among them, $\varepsilon_{min} = 0.6$, $\varepsilon_{max} = 0.95$, $\Delta R_n$ is the difference between the reward value of the $n$ round and the $n - 1$ round.

In the 10*10 grid environment, 500 rounds of iteration, the experimental results are shown in Table 3.

| Algorithm | The first round to find the target | Elapsed time (s) | Shortest path | 500 iterations total elapsed time (s) |
|---|---|---|---|---|
| Algorithm 1 | 76 | 36.67 | 19 | 80.78 |
| Algorithm 2 | 64 | 28.21 | 18 | 79.96 |
| Algorithm 3 | 10 | 3.26 | 18 | 63.09 |
| Algorithm 4 | 9 | 2.77 | 18 | 60.21 |

The algorithm 4 (DQN with heuristic reward and adaptive exploration strategy) proposed in the article has the highest path planning efficiency from the experimental results that, and the shortest path is planned. From the point of view of the first round of finding the target and its time consumption, Algorithm 1 and Algorithm 2 require 76 and 64 rounds for the first search respectively, which takes 36.67s and 28.21s respectively, and the total time is 80.78s and 79.96s respectively; Algorithm 3 and Algorithm 4 both set up heuristic rewards. It only takes 10 rounds and 9 rounds to find the target point,
which takes 3.26s and 2.77s, and the total time is 63.09s and 60.21s. Shown that setting up heuristic rewards can greatly improve the efficiency of the algorithm. This is because in an environment with no prior information, after setting heuristic continuous rewards, there are comprehensive rewards in each state, and the agent is not blindly exploring.

We visualize the final optimal path, from left to right are the path planning results of algorithm 1, algorithm 2, algorithm 3, and algorithm 4, as shown in the figure 3.

![Figure 3. Visualization results of 10*10 grid environment pathfinding](image)

The path planned by Algorithm 1 is not optimal from table 3, which is 19 steps, and three improved algorithms all planned the shortest path to reach the target point, which is 18 steps. Why the path display is different in picture 3 is because we take the path to the target point the first time we find it when we save the shortest path, and the path library is not updated, so the specific path may be different during visualization. From left to right, from top to bottom are the path planning results of Algorithm 1, Algorithm 2, Algorithm 3, and Algorithm 4.

In a 25*25 grid environment, iterated 3000 rounds, the experimental results are shown in Table 4, and the pathfinding results are visualized in Figure 4.

![Figure 4. Visualization results of 25*25 grid environment pathfinding](image)

The four algorithms can find the shortest path in 3000 rounds, and the shortest path is 48 steps. The path display is different because when saving the shortest path, the path to the target point for the first time will prevail, so the path during visualization will be different.
Experiments show that the DQN proposed in the article combines heuristic rewards and adaptive exploration with high learning efficiency and fast convergence, and the planned path is also optimal.

4. Conclusion
The ordinary DQN algorithm can find a collision-free path to the target point in an unknown simple environment lacking prior information, but the learning efficiency is low and the convergence is slow. To solve this problem, we propose a DQN path planning method that combines heuristic rewards and adaptive exploration strategy. A continuous heuristic reward function is designed to replace the sparse reward function, and the heuristic reward function is designed based on the artificial potential field method. The gravitation and repulsion reward function in the formula function improves the learning efficiency and convergence, and proposes an adaptive $\varepsilon - greedy$ exploration strategy, which adaptively adjusts the balance between exploration and utilization in reinforcement learning, and accelerates learning efficiency. After experimental verification, the improved DQN algorithm proposed in this paper has fast convergence, fast path finding, and can find the optimal path, which solves the problem of low learning efficiency and slow convergence caused by sparse reward and blind exploration.

References

[1] Hart, P. E., Nilsson, N. J., Raphael, B. (1972). A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science & Cybernetics, 4(2): 28-29.

[2] Khatib, O. (2003). Real-time obstacle avoidance for manipulators and mobile robots. Proceedings. 1985 IEEE International Conference on Robotics and Automation. IEEE, 2:500-505.

[3] Ben J. A. Kröse. (1995). Learning from delayed rewards. Robotics & Autonomous Systems, 15(4): 233-235.

[4] Zhao, Y., Chen, Q., Hu, W. (2010). Reinforcement learning algorithm based on information entropy. Xi Tong Gong Cheng Yu Dian Zi Ji Shu/Systems Engineering and Electronics, 32(5): 1043-1046.

[5] Liu, X., Zhou, Q., Ren, H., Sun, C. (2018) Reinforcement Learning for Robot Navigation in Nondeterministic Environments[C]. CCIS,IEEE. pp. 615-619.

[6] Dong P, Zhang Z, Mei X, Zhu S. (2018) Reinforcement learning path planning algorithm introducing potential field and trap search[J]. Computer Engineering and Applications, 1: 129-134.

[7] Volodymyr M, et al. (2013) Playing Atari with Deep Reinforcement Learning. NIPS

[8] Sichkar V N. (2019) Reinforcement Learning Algorithms in Global Path Planning for Mobile Robot[C]. ICIEAM. pp. 1-5.

[9] Wang, C., Wang, J., Shen, Y., Zhang, X. (2019). Autonomous navigation of uavs in large-scale complex environments: a deep reinforcement learning approach. IEEE Transactions on Vehicular Technology, 68(3): 2124-2136.

[10] Marashi, M., Khalilian, A., Shiri, M. E. (2012). Automatic reward shaping in Reinforcement Learning using graph analysis. Computer and Knowledge Engineering (ICCKE), 2012 2nd International eConference on. IEEE.pp. 111-116.

[11] Choi, J., Park, K., Kim, M., Seok, S. (2019). Deep Reinforcement Learning of Navigation in a Complex and Crowded Environment with a Limited Field of View. 2019 International Conference on Robotics and Automation (ICRA),pp. 20-24.