Training UGV For Optimal Path Planning Using Reinforcement Learning

Phuong Lan Le[1*]
[1]Vietnam Aviation Academy, Vietnam
*Corresponding Author/Email: lanlp@vaa.edu.vn

Manuscript received: October 24, 2025 / Revised: November 10, 2025 / Accepted: November 27, 2025

ABSTRACT

In the current era of artificial intelligence technology, unmanned ground vehicles (UGV) are increasingly being widely used in various fields due to the application of advanced technologies such as computer vision and reinforcement learning (RL). These technologies contribute to enhancing the level of automation, environmental awareness, and real-time flexible decision-making capabilities of UGVs. In this paper, we will apply RL with a decaying exploration rate strategy to train UGVs for path planning, obstacle avoidance, and optimal route discovery. Specifically, in the early stages of training, the exploration rate is set to a maximum value so that the agent can explore and collect more information of the state space; as the training time progresses, the exploration rate is gradually reduced, corresponding to a greater focus on the exploitation to find the optimal. Simulation results show that our proposal achieves faster convergence than RL in benchmark, demonstrating the effectiveness of this algorithm for real-world UGV applications.

KEYWORDS: Reinforcement learning (RL), Unmanned ground vehicle (UGV)

1. Introduction

Unmanned ground vehicles (UGV) have been proving their crucial role in replacing humans in performing dangerous or tedious tasks that are unsuitable for humans or where human labor is unnecessary. With the ability to operate flexibly in various terrains and environments, UGVs are increasingly being widely applied in military, civil, and industrial sectors. In particular, the development of advanced technologies such as computer vision and reinforcement learning has contributed to enhancing the level of automation, environmental awareness, and real-time flexible decision-making capabilities of UGVs.

Reinforcement learning (RL), a significant branch of machine learning, involves an agent interacting with an environment to optimize objectives through trial and error and behavioral adjustments. The RL model is often represented as a Markov Decision Process (MDP), where each state contains sufficient information for the agent to make optimal decisions. A fundamental characteristic of RL is its reward-based learning mechanism, which helps the agent determine optimal action strategies without direct guidance. This learning process requires a balance between exploring new actions to gather more information and exploiting actions that have been identified as optimal to maximize rewards. RL has been applied in numerous important fields. In robotics, RL supports motion and control optimization, enabling robots to perform tasks such as grasping or moving in complex environments. Additionally, RL plays a crucial role in autonomous vehicle control, helping to optimize navigation, cruise control, collision avoidance, and improve operational efficiency.

Among the algorithms of RL, Q-learning stands out as a typical model-free learning method, allowing the agent to learn the optimal policy without a dynamic model of the environment. Q-learning has the advantage of being simple, easy to implement, and requiring low processing power, but has the limitation that if the exploration process is required to collect information about the state space, it will lead to false or slow convergence. To overcome this obstacle, it is necessary to have a strategy to enhance the exploration process but balance it with the exploitation process to achieve fast and accurate convergence. This paper proposes Q-learning combined with a decaying exploration rate strategy to solve this problem.

The research on applying UGVs combined with RL is increasingly attracting attention in the scientific and technical community due to its significant potential in enhancing the efficiency, autonomy, and adaptability of automatic control systems. Recent studies have proposed various approaches, from convex optimization in real-time collision avoidance path planning based on the dynamics model of UGVs, to deep reinforcement learning (DRL) methods for handling navigation situations in partially observed environments.

Specifically, (P. Zhang et al., 2024) trajectory optimization has been approached by successive convexification to effectively address obstacle and destination constraints, while RL methods such as Duel Deep Q-Network, Proximal Policy Optimization, or Imitation-Augmented DRL have been exploited to improve the self-learning and adaptation capabilities of UGVs in dynamic real-world environments (Wang et al., 2024). Study (Hassan et al., 2024) proposed a control framework combining a target network and a critic network to optimize control behavior based on both deviation errors and trajectory distance, thereby enhancing

movement accuracy.

Furthermore, hybrid models combining Probabilistic Roadmap and Proximal Policy Optimization have shown effectiveness in leveraging global guidance combined with local optimization, minimizing the local optimal phenomenon common in traditional methods (Han et al., 2024). Beyond single applications, many recent studies have extended control models to collaborative environments between UGVs and UAVs, allowing vehicles to coordinate intelligently to overcome individual limitations such as operating range or accessibility, thereby performing more complex tasks such as cooperative routing, environmental monitoring, or automatic charging support (J. Zhang et al., 2020), (Mondal et al., 2024), (Hulede et al., 2024).

All of these approaches not only affirm the superior potential of RL in enhancing the autonomy and efficiency of UGVs, but also open up prospects for practical UGV applications. However, the above algorithms are all improved algorithms, which have many advantages that can train UGVs in complex spaces, but also require large computational resource consumption. Our hope is to be able to take advantage of the simplicity with low processing power requirements of Q-learning and still be able to apply UGVs in small enterprises had the scope of the factory with a non-complex environment, at low cost. In this paper, we will apply Q-learning combined with a decaying exploration rate strategy (QDR) to train UGVs for path planning, obstacle avoidance, and optimal route discovery. Our simulation results have demonstrated the effectiveness of this algorithm for real-world UGV applications.

The rest of this paper is organized as follows: Section 2 introduces RL modeled as MDP. Section 3 describes the system model and the implementation of training with a specific algorithm. Section 4 will perform simulations and evaluate the results obtained, and conclusions will be presented in Section 5.

2. Reinforcement Learning Modeled as Markov Decision Process

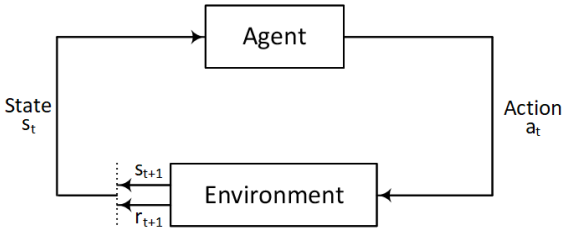


Figure 1: The agent–environment interaction in a MDP

To model RL, we use MDP as shown in Figure 1, so we need to consider some concepts:

- Agent: In RL, the agent is not told what actions to take but interacts with the environment on its own, aiming in the short term to make optimal action choices for the greatest possible reward, and in the long term to achieve the target.
- Environment: The scope in which the agent operates, encompassing everything surrounding the agent.
- State space: The set of all possible states of an agent in the environment, is expressed as

$$\mathcal{S} = \{s_1, s_2, \dots, s_N\}, \quad (1)$$

where N is a finite sum of states.

- Action space: The set of all actions an agent can perform in the environment, is expressed as

$$\mathcal{A} = \{a_1, a_2, \dots, a_M\}, \quad (2)$$

where M is a finite sum of actions.

- Reward function: Defined for each state. The reward can be positive or negative depending on the state. A positive reward encourages the agent to move to that state, while a negative reward penalizes the agent for entering undesirable states, teaching the agent to avoid them. Usually defined as a function

$$R: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R} \quad (3)$$

Meaning the agent in current state s_t, performs action a_t, and moves to the next state s_{t+1}, receives reward $R(s_t, a_t, s_{t+1})$.

- Policy π: The strategy the agent uses to select actions at each state in the environment. When the agent is in the state s_t and takes action at a_t, then the policy $\pi(a_t \mid s_t)$ is the probability of taking action when in state s_t, expressed as

$$\pi(a_t|s_t) = P(A_t = a_t|S_t = s_t) \quad (4)$$

and

$$\sum_{a_t \in \mathcal{A}} \pi(a_t|s_t) = 1 \quad (5)$$

In general, the model in Figure 1 shows that at time t, the agent in state s_t with reward r_t interacts with the environment through action a_t to transition to a new state s_{t+1} and receive reward r_{t+1}. Thus, the agent in MDP generates sequences (s_t, r_t, a_t, s_{t+1}, r_{t+1}, a_{t+1}, s_{t+2}, r_{t+2}...) from the starting point to the ending point, with the goal that this sequence is executed with the maximum reward.

3. System Model And Implementation

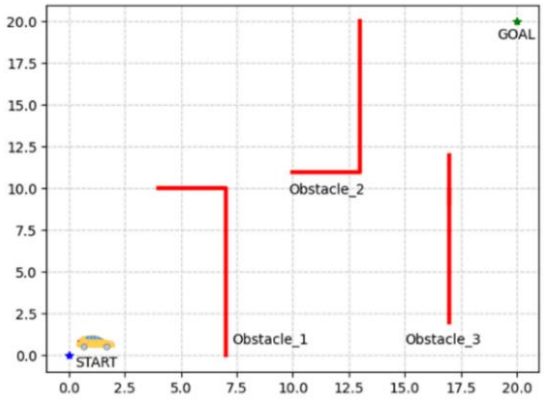


Figure 2: System model

We consider the UGV's movement floor model as a $x_N \times y_M$ grid with several obstacles represented as red lines, as shown in Figure 2. The UGV's task is to find a path from the bottom left corner (START point) to the top right corner (GOAL point) with the fewest steps, i.e. traveling the shortest distance and avoiding collisions with obstacles.

To solve this problem using MDP above, we define some objects corresponding to the concepts in MDP:

- Agent: In this paper, the agent is the UGV. The agent's task is to find the optimal path from the START point to the GOAL point.
- Environment: being the $x_N \times y_M$ grid containing states with different characteristics and obstacles. In our research, the characteristics of the environment and obstacles are fixed. This information can be modified but must be provided to the agent before training. The agent will navigate between these states to reach GOAL without colliding with obstacles.
- State: being the agent's position in the environment, typically represented as s_t, s_{t+1}, corresponding to the current state and the state the agent will move to. The set of states $\mathcal{S}$ is the set of grid points, expressed as $[x_i, y_j]^T \in \mathcal{S}$, where i = {1, 2, …, N}, j = {1, 2, …, M}
- Action a_t: The action the agent takes to move from one state to another. At each state, the agent can move to one of its eight adjacent positions through corresponding actions defined by the action set $\mathcal{A} = [-3, -2, -1, 0, 1, 2, 3, 4]$. This action set corresponds to the agent moving to the nearest grid point at an angle of $\left[-\frac{3\pi}{4}, -\frac{\pi}{2}, -\frac{\pi}{4}, 0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}, 2\pi\right]$. After each action to move to the nearest grid point, the agent is considered to perform a step.
- Reward r_t: Defined as 1 at GOAL point, -1 at obstacles and 0 in other states.
- Policy π: The strategy the agent uses to select actions in $\mathcal{A}$ at each state st in the environment.

The interaction process between the agent and the environment is carried out according to the MDP with the goal that this sequence is executed with the fewest steps, i.e., the getting path having the shortest distance with the average total reward, where the average total reward is determined by total reward divided by number of steps.

The key point of RL is to statistically determine the value of the state-action pair $Q(s_t, a_t)$. We use the Monte Carlo method to solve the problem. Our main objective is to estimate these values, storing them in a buffer, and updating them frequently during the agent's learning process, namely Q-learning. This helps the agent gain more information about the environment, enabling it to choose the optimal action, i.e., the action with the highest probability at each state where it needs to select an action. A policy is used for action selection, namely ε- greedy policy which involves choosing the action with the highest state-action value, as follows

$$\pi(a_t|s_t) = \underset{a}{\operatorname{argmax}}\, Q(s_t, a_t). \quad (6)$$

The state-action value of the agent is then updated as follows (Sutton & Barto, 2020)

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left(R + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right), \quad (7)$$

where $Q(s_t, a_t)$ is the current state-action value, $Q(s_{t+1}, a)$ is the next state-action value when execute action a, γ is the discount factor, representing the importance of the distant future to the current state-action value, and α is the learning rate (ln_rate), which is a parameter affecting the learning speed and accuracy of the training process.

4. Simulation And Evaluation

We use Python to simulate the model described in Figure 1 with $x_N \times y_M = 20 \times 20$ and utilize RL with Q-learning in benchmark (QLB), detailed in Algorithm 1, to train the UGV with crucial RL parameters that require careful consideration through experimentation.

We divide this process into two stages: 1) examining the impact of the ln_rate on the training results by conducting experiments with different ln_rates while keeping other parameters constant, such as the exp_rate, discount factor, and number of iterations; 2) examining the impact of the exp_rate on the training results, performed similarly by experimenting with different exp_rates while keeping other parameters constant, such as the ln_rate, discount factor, and number of iterations.

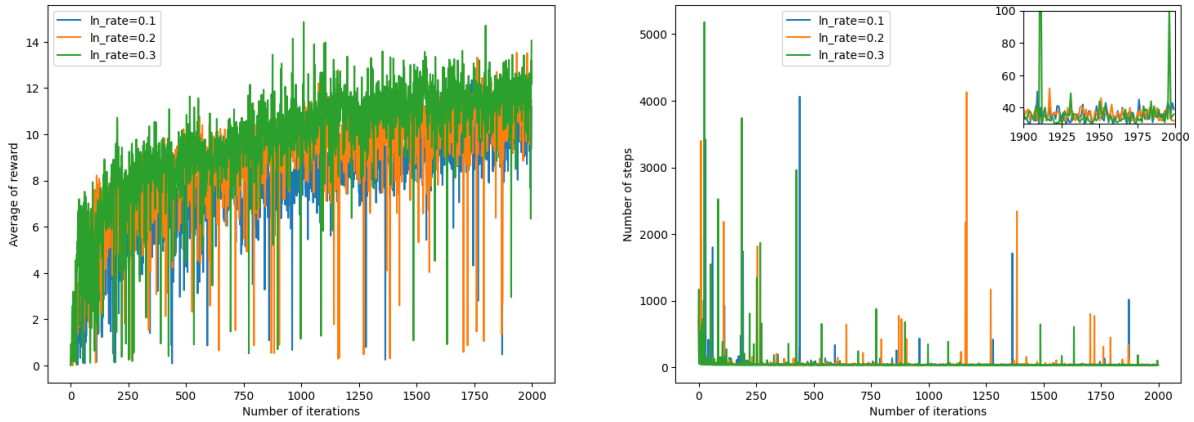
The experimental results for different ln_rates are shown in Figure 3, indicating that with a very small ln_rate, the learning speed will be slow and convergence will be slow. However, if the ln_rate is higher, the learning speed will be faster, but increasing the ln_rate too much can affect the accuracy of the results and may even prevent convergence. Therefore, it is necessary to strike a balance between the speed and accuracy of the training process.

Algorithm 1 QLB algorithm

```
Initialise Q(s, a) = [] ∀s ∈ S, ∀a ∈ A
Initialise tracking_path = []
Initialise START point
for episode = 1; … ; M do
    while state != GOAL
        Choose a_t according to (6)
        Execute the action a_t to move to new state s_{t+1}
        and receive reward r_{t+1}
        Store in tracking_path
    if state == GOAL
        Update the value of the state-action pair
Q(s_t, a_t) according to (7)
episode += 1
```

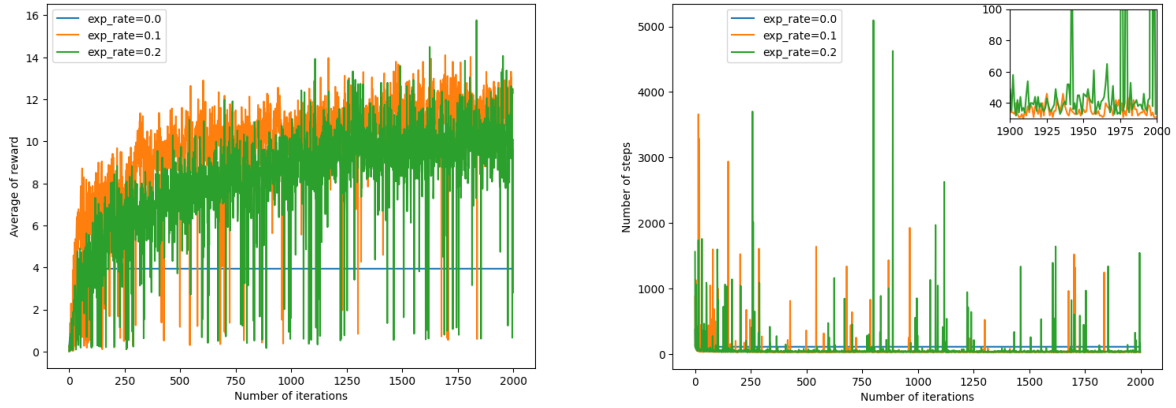
The exp_rate is the coefficient in the ε-greedy strategy aimed at balancing exploration and exploitation during training. Here, exploitation is the process where the agent selects the action with the best state-action value function, while exploration is an action performed randomly to gather new information about the environment. The experimental results for the exp_rate are illustrated in Figure 4. When the exp_rate is small, the agent performs less exploration, resulting in faster convergence; however, this increases the likelihood of being stuck in a local optimum, since the environment has not been sufficiently explored. Conversely, with a higher exp_rate, the agent explores the environment more extensively, gaining richer information and thus improving the chances of finding the global optimum.

Nevertheless, if the exp_rate becomes excessively large, the agent may overemphasize exploration, focusing on gathering information about the environment rather than exploiting the optimal strategies it has already learned.



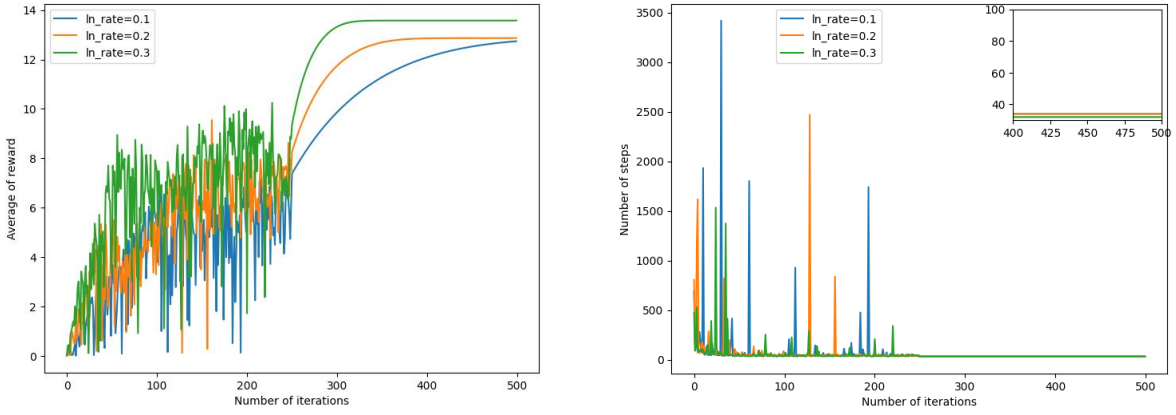
a) Average of reward b) Number of steps

Figure 3: Training results of QLB in different ln_rate cases (keeping exp_rate = 0.1)



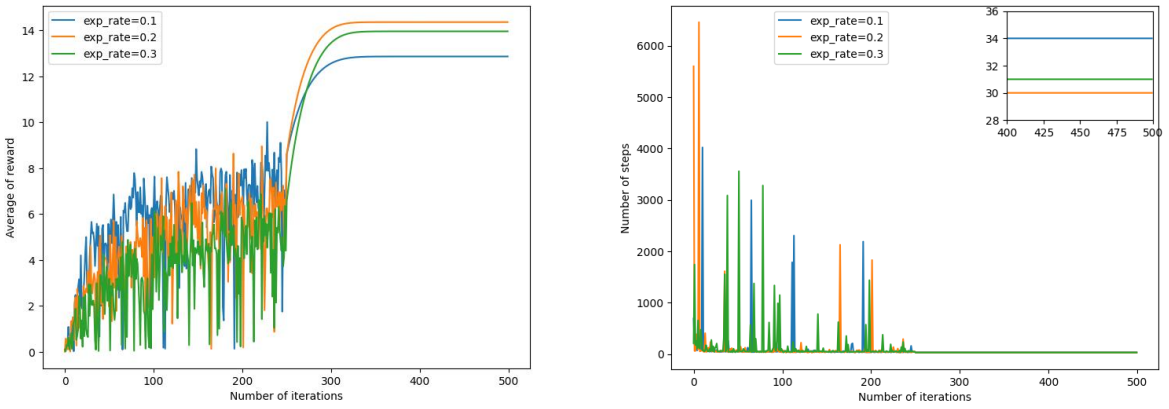
a) Average of reward b) Number of steps

Figure 4: Training results of QLB in different exp_rate cases (keeping ln_rate = 0.3)



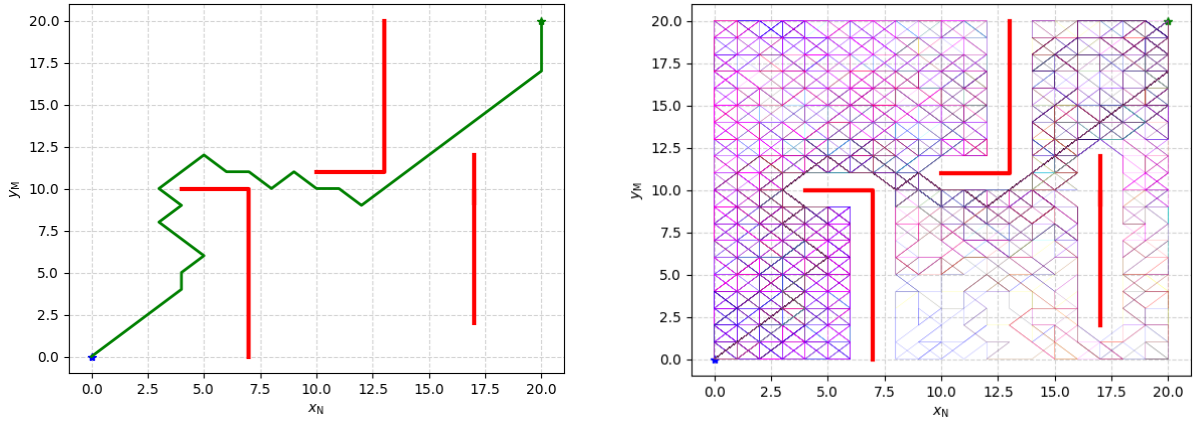
a) Average of reward b) Number of steps

Figure 5: Training results of QDR in different ln_rate cases (with maximum exp_rate = 0.1)



a) Average of reward b) Number of steps

Figure 6: Training results of QDR in different maximum exp_rate cases (keeping ln_rate = 0.3)



a) Optimal path of trained agent b) All paths of agent when training

Figure 7: Path of agent

Considering the parameters with constant values throughout the QLB training process merely provides insights into the nature of these parameters, but does not yield the desired optimal results. This is because the training process has not yet achieved convergence, as the agent places excessive emphasis on exploring information about the environment. An exception occurs when exp_rate = 0.0, as shown in Figure 4. However, in the case of exp_rate=0.0, although the training process converges, the results fail to meet the desired requirements. This is because the agent lacks sufficient information about the environment and becomes trapped in a local optimum. This is evident in Figure 4a, where the convergence line for exp_rate=0.0 reaches an optimal value significantly smaller than that of the other cases. Another one in Figure 4b, which zooms in on the range between 30 and 100 steps, the optimal result for exp_rate=0.0 still does not appear within this interval. To overcome this problem, it is necessary to use QDR during the training process. Specifically, the training process is divided into two stages: 1) the first stage involves training with a large exp_rate so that the agent explores more and obtains more information about the environment; 2) the second stage involves gradually reducing the exp_rate to focus more on exploitation to converge to a stable optimal value. The details of the QDR algorithm are presented in Algorithm 2.

Algorithm 2 QDR algorithm

01: Initialise $Q(s,a) = []\forall s \in \mathcal{S}, \forall a \in \mathcal{A}$
02: Initialise tracking_path = []
03: Initialise START point
04: for episode = 1; … ; M do
05: Determine exp_rate according to episode
06: while state != GOAL
07: Choose a_t according to (6)
08: Execute the action a_t to move to new state s_{t+1} and receive reward r_{t+1}
09: Store in tracking_path
10: if state == GOAL
11: Update the value of the state-action pair$Q(s_t, a_t)$ according to (7)
12: episode += 1

The experimentation to determine the parameters in the

QDR algorithm is also performed similarly to the experimentation process for the QLB algorithm. The experimental results for different ln_rates are shown in Figure 5, indicating that with a small ln_rate, i.e., when ln_rate = 0.1, the agent learns slowly and converges slowly (Figure 5a), and even the enlarged portion in Figure 5b shows the value exceeding the limit of portion, i.e., over 100 steps, compared to other results. In Figure 5, ln_rate = 0.3 appears to be the most effective, and we will keep this value for testing different maximum exp_rates, where is the exp_rate value set from the beginning and used in the first stage of QDR and is the basis for implementing the exp_rate decay strategy in the later stages.

The experimental results in Figure 6 for different maximum exp_rates once again confirm that a small maximum exp_rate will not provide enough information for the agent to find the optimal solution, while an excessively large maximum exp_rate will cause the exploration information to be spread too widely, also leading to a failure to converge to the optimal solution. The most important thing is to choose a reasonable value to achieve the most optimal value. In this paper, a maximum exp_rate = 0.2 is the most suitable. Furthermore, the experimental results in Figures 5 and 6 also show that the QDR algorithm significantly outperforms QLB, requiring only 500 iterations to achieve the global optimal value, while QLB cannot find the optimal value even after 2000 iterations.

To compare QLB and QDR quantitatively, we tried to increase the number of iterations of QLB to achieve convergence and compared the convergence results of the two methods. As a result, we had to increase the number of iterations of QLB to 10000 to achieve convergence. The comparison results are given in Table 1.

Table 1: The convergence results between QLB and QDR

Parameter	QLB	QDR
Number of iterations to train	10000	500
Number of iterations getting convergence	3500	300
Average reward	13.62	14.36
Number of steps	30	30

Figure 7a shows the optimal path of the agent after the training process with 500 iterations, applying a ln_rate = 0.3 and an maximum exp_rate = 0.2. Figure 7b shows the path of the agent in all 500 iterations, indicating that all states have been visited by the agent to some extent to gather information, thereby selecting the path with the largest average total reward, i.e., the smallest number of steps.

To examine the stability and reliability of the algorithm, we ran the algorithm 50 times for statistics. The results after 50 times are shown in Figure 8, we obtained a mean of steps of 30.88 with the results of each run ranging from 30 to 32, and a mean of reward of 14.01 with the results of each run ranging from 13.57 to 14.36. The statistical results have fluctuations within the acceptable threshold, showing that the algorithm has the accuracy and stability to achieve the global optimum.

Applying the found optimal parameters of the algorithm to a larger state space model to approach real-world applications, i.e., the state space with $x_N \times y_M = 200 \times 150$ grid, we get the path of the agent after training as shown in Figure 9. This result once again confirms that our proposal can be applied to real-world applications, even with larger spaces if processing power allows.

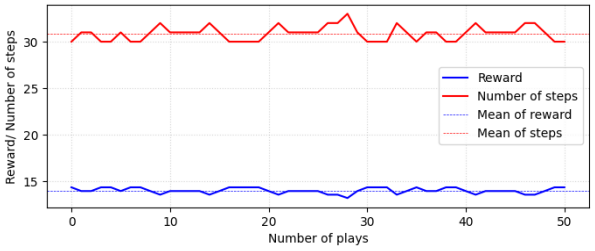


Figure 8: The stability and reliability of the algorithm

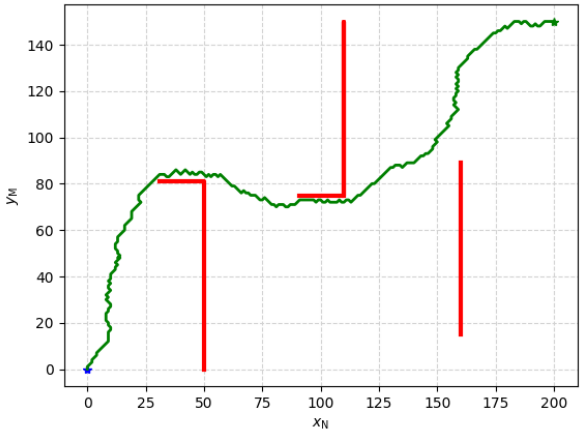


Figure 9: The path of trained agent in larger state space

5. Conclusions

Due to the limited processing power of personal computers, this paper only simulates an environment model with a size of xN × yM = 20 × 20. However, simulation results using QDR algorithm to train UGV with selected hyperparameters, e.g. ln_rate = 0.3 and exp_rate = 0.2, show that the trained UGV finds the optimal path with low computational power consumption in a fixed environment. It can be concluded that the QDR algorithm can be applied in practice in factories with fixed terrain distribution and large areas with the investment cost made by Q-learning solution is acceptable. The application of training UGV to move within the factory premises will help reduce labor costs for tedious tasks that can be automated. Within the scope of this paper, we have only focused on optimizing the number of steps from the START point to the GOAL point and have not yet optimized the shape of the path, so the visual appearance of the path still seems unsatisfactory. We will continue to improve in the future to achieve a more optimal path and develop additional stopping points to better suit the needs of actual application.

References

Han, Z., Chen, P., Zhou, B., & Yu, G. (2024). Dual-Layer Path

Planning for Unmanned Ground Vehicles Based on Probabilistic Roadmap and Proximal Policy Optimization. 2024 IEEE 22nd International Conference on Industrial Informatics (INDIN), 1–6. https://doi.org/10.1109/INDIN58382.2024.10774269

Hassan, I. A., Ragheb, H., Sharaf, A. M., & Attia, T. (2024). Reinforcement Learning for Precision Navigation: DDQN-Based Trajectory Tracking in Unmanned Ground Vehicles. 2024 14th International Conference on Electrical Engineering (ICEENG), 54–59. https://doi.org/10.1109/ICEENG58856.2024.10566281

Hulede, I. E. L., Mukherjee, A., & Ashdown, J. (2024). Hierarchical Adaptation of Multiagent Deep Reinforcement Learning for Multi-Domain Uncrewed Aerial and Ground Vehicle Coordination. MILCOM 2024 - 2024 IEEE Military Communications Conference (MILCOM), 487–492. https://doi.org/10.1109/MILCOM61039.2024.10773714

Mondal, M. S., Ramasamy, S., Humann, J. D., Dotterweich, J. M., Reddinger, J.-P. F., Childers, M. A., & Bhounsule, P. (2024). An Attention-aware Deep Reinforcement Learning Framework for UAV-UGV Collaborative Route Planning. 2024 IEEE/RSJ International Conference on ntelligent Robots and Systems (IROS), 13687–13694. https://doi.org/10.1109/IROS58592.2024.10801704

Sutton, R. S., & Barto, A. (2020). Reinforcement learning: An introduction (Second edition). The MIT Press.

Wang, W., Kong, C., & Diao, H. (2024). Path Planning of UGV via Deep Reinforcement Learning with Observation Enhancement. 2024 IEEE 2nd International Conference on Control, Electronics and Computer Technology (ICCECT), 1552–1558. https://doi.org/10.1109/ICCECT60629.2024.10545909

Zhang, J., Yu, Z., Mao, S., Periaswamy, S. C. G., Patton, J., & Xia, X. (2020). IADRL: Imitation Augmented Deep Reinforcement Learning Enabled UGV-UAV Coalition for Tasking in Complex Environments. IEEE Access, 8, 102335–102347. https://doi.org/10.1109/ACCESS.2020.2997304

Zhang, P., Chen, H., Ma, Z., & Wu, C. (2024). Convex Optimization Based Collision Avoidance Path Planning Method for Unmanned Ground Vehicles. 2024 36th Chinese Control and Decision Conference (CCDC), 844–849. https://doi.org/10.1109/CCDC62350.2024.10588101