drones

MDPI

Article

Dynamic Scene Path Planning of UAVs Based on Deep Reinforcement Learning

Jin Tang [1,2], Yangang Liang [1,2] and Kebo Li [1,2,*]

[1] College of Aerospace Science and Engineering, National University of Defense Technology, Changsha 410073, China; tj20@nudt.edu.cn (J.T.); liangyg@nudt.edu.cn (Y.L.)
[2] Hunan Key Laboratory of Intelligent Planning and Simulation for Aerospace Mission, Changsha 410073, China
* Correspondence: likeboreal@nudt.edu.cn

Abstract: Traditional unmanned aerial vehicle path planning methods focus on addressing planning issues in static scenes, struggle to balance optimality and real-time performance, and are prone to local optima. In this paper, we propose an improved deep reinforcement learning approach for UAV path planning in dynamic scenarios. Firstly, we establish a task scenario including an obstacle assessment model and model the UAV's path planning problem using the Markov Decision Process. We translate the MDP model into the framework of reinforcement learning and design the state space, action space, and reward function while incorporating heuristic rules into the action exploration policy. Secondly, we utilize the Q function approximation of an enhanced D3QN with a prioritized experience replay mechanism and design the algorithm's network structure based on the TensorFlow framework. Through extensive training, we obtain reinforcement learning path planning policies for both static and dynamic scenes and innovatively employ a visualized action field to analyze their planning effectiveness. Simulations demonstrate that the proposed algorithm can accomplish UAV dynamic scene path planning tasks and outperforms classical methods such as A*, RRT, and DQN in terms of planning effectiveness.

Keywords: path planning; UAV; deep reinforcement learning; heuristic policy; prioritized experience replay

Citation: Tang, J.; Liang, Y.; Li, K. Dynamic Scene Path Planning of UAVs Based on Deep Reinforcement Learning. *Drones* **2024**, *8*, 60. https://doi.org/10.3390/drones8020060

Academic Editor: Diego González-Aguilera

Received: 11 December 2023
Revised: 4 February 2024
Accepted: 5 February 2024
Published: 9 February 2024

1. Introduction

Unmanned Aerial Vehicles (UAVs) are commonly employed for executing hazardous missions in complex environments and unmanned areas, with path planning being an indispensable and crucial aspect to ensure the safe flight of UAVs [1]. Depending on a UAV's perception of the overall environment, the path planning problem can be divided into local path planning and global path planning [2]. Global path planning involves environmental modeling, and its accuracy depends on the precision of environmental information. Local planning requires real-time perceptual information, making it more responsive and adaptable to dynamic environments; however, relying solely on local information may lead to local optima.

The paramount concern in UAV path planning lies in algorithm design, with planning methods primarily categorized into classical algorithms and intelligent algorithms [2]. Classical algorithms include the A* algorithm [3], artificial potential field method [4], Rapidly Exploring Random Tree (RRT) [5], and cellular decomposition method [6]. However, traditional global path planning algorithms are characterized by fast planning speeds but lack optimality. Planning algorithms that rely on prior environmental information can address general static scene problems but fall short in situations where obstacles are mobile, posing limitations. Conventional local path planning methods, due to the absence of global information, often struggle to generate optimal paths and require high computational capabilities to meet real-time demands.

Therefore, in response to the UAV path planning challenges in complex scenarios, scholars have proposed bio-inspired intelligent algorithms that mimic the foraging behavior of organisms in dynamic environments. These include genetic algorithms [7], ant colony algorithms [8], and particle swarm algorithms [9]. Recognizing the limitations of genetic algorithms, adaptive crossover and mutation methods have been introduced to enhance the algorithm's robustness [10]. Another work [11] combines a graph-search-based distributed ant colony algorithm with grid modeling, analyzing the impact of heuristic factors, population size, and information evaporation coefficient on path length and planning efficiency. Additionally, a dual global optimization genetic-particle swarm optimization algorithm is proposed by integrating genetic algorithms with particle swarm optimization in one study [12], improving algorithm efficiency. In another study [13], this method is employed for the progressive search of a robot's global feasible path, enhancing path quality and convergence speed. However, these intelligent path planning methods struggle with handling high-dimensional information in complex environments, face challenges in balancing optimality and real-time requirements, and are prone to falling into local optima.

With the increasing complexity and uncertainty of the mission environment, UAV path planning faces greater challenges. In recent years, research on UAV path planning based on artificial intelligence has significantly increased, further expanding the application of RL and Deep Reinforcement Learning (DRL) [14] in this field. Ref. [15] using Q-learning employs adaptive random exploration to achieve UAV navigation and obstacle avoidance, but the method is constrained by action space and sample dimensions. Addressing the issue of UAVs' inability to acquire environmental information, ref. [16] defines rewards based on signal strength, proposing a guided reinforcement Q-learning method, and simulation validates the advantages of this approach. However, general RL can only solve simple, discrete state problems and cannot cope with high-dimensional inputs and the issue of dimension explosion. Ref. [17] introduces a Long Short-Term Memory (LSTM) network into a proximal policy optimization algorithm, proposing an intelligent path planning algorithm. However, its environment only considers low walls, limiting its applicability. Ref. [18] suggests an improved Deep Q Network (DQN) path planning algorithm, addressing the overestimation problem of the Q network. A comparison of path length and convergence speed illustrates the algorithm's superiority, yet it does not address the problem of optimizing the path.

Additionally, ref. [19] employs a Dueling Double Deep Q Network (D3QN) algorithm with a dueling architecture to address path planning problems. This algorithm approximates Q-values with the input of situations and validates the method's excellent performance. Ref. [20] proposes an improved path planning approach that combines Q-learning and Boltzmann exploration, allowing for a suitable balance between exploration and exploitation in the policy. In response to the issues present in traditional algorithms and general intelligent algorithms, this article, based on reinforcement learning theory, introduces an enhanced algorithm, the D3QN, incorporating heuristic action policy and prioritized experience replay. The D3QN algorithm successfully resolves path planning problems in dynamic scenarios.

This study conducts research on the path planning problem for UAVs by introducing deep reinforcement learning algorithms to achieve autonomous path planning in both static and dynamic scenarios, enhancing the efficiency and success rate of path planning. This article is organized as follows. Section 2 provides a description of the UAV path planning problem and establishes the environmental model. Section 3 introduces the methods employed in this study, including the construction of a reinforcement learning Markov decision model and the use of the improved D3QN algorithm. Section 4 presents the research results and provides a thorough analysis and discussion. Finally, Section 5 presents the conclusions.

2. Problem Description and Model

2.1. Path Planning Problem

The purpose of UAV path planning is to generate a series of waypoints as targets for autonomous navigation. This involves a sequence decision-making process, and thus, the process exhibits Markovian properties, satisfying the following equation:

$$P_{ss'}^{a} = \mathrm{E}(S_{t+1} = s' | S_t = s, A_t = a) \tag{1}$$

Based on this characteristic, the decision-making process aligns with the fundamental principles of reinforcement learning (RL), allowing for the establishment of a corresponding Markov Decision Process (MDP) model. This can then be integrated into the RL framework, utilizing RL algorithms for solving.

Figure 1 shows how the UAV path planning problem integrates with the principles of reinforcement learning. At a given time step t, the UAV's state is denoted as s, and the UAV agent needs to make decisions on the next action a. Upon executing this action, both the UAV's state and the environment undergo changes, leading to a subsequent decision-making iteration. This process continues iteratively until the completion of the mission. Throughout this journey, an intelligent agent with an effective planning policy is essential for the satisfactory accomplishment of tasks. This policy can be acquired through offline training, which constitutes the overarching approach of this research.

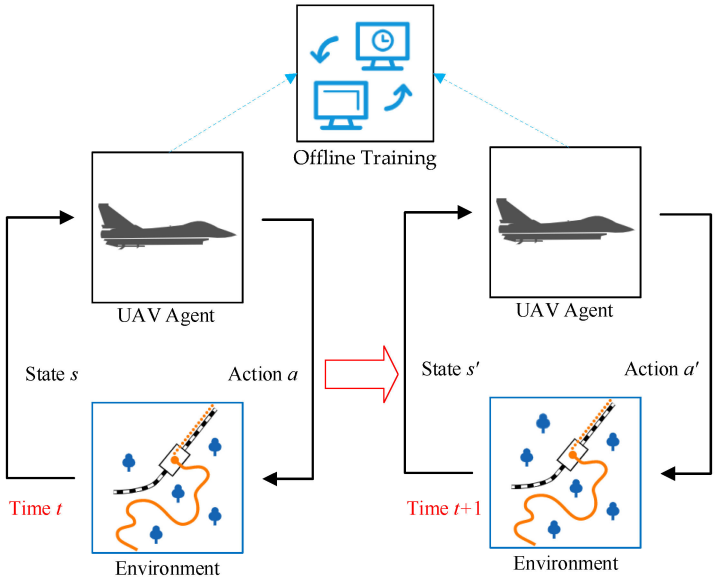


Figure 1. The UAV path planning in RL framework.

2.2. Environmental Model

The path planning environment comprises the UAV, the target, and the obstacle zone. In this study, reference [21] is consulted to establish a model for assessing obstacle zone. Typically, the UAV conducts lots of missions at a constant altitude. Therefore, the designed scenario involves a two-dimensional space, without considering variations in altitude.

The Situation Assessment (SA) model serves as the foundation for path planning problems [22]. In aerial scenarios, various obstacle assessment models are utilized, typically constructing the SA model based on the distance between the UAV and the no-fly zone center. In this paper, an obstacle assessment model is established based on the distance between the UAV and the no-fly zone center, composed of detection radar and SAM [22]. When the detector detects the UAV, it employs the SAM to intercept the UAV, with the destruction probability determined by the relative distance between the UAV and the no-fly zone.

Building upon the original model described in the reference, this paper simplifies and integrates the model, aiming to further increase the survivability difficulty of UAVs. It modifies the region between the radar detection radius and the SAM kill radius in the original model to be the UAV's no-escape zone, effectively expanding the radius of the UAV's no-escape zone. This increases the risk of UAVs entering the threat area, posing a greater challenge to the D3QN deep reinforcement learning algorithm enhanced in this study and better reflecting the method's advantages in addressing this problem. The dangerous level of the no-fly zone center towards the UAV is defined by the following equation.

$$T_p = \begin{cases} 0 & D \geq R_{\max} \\ 2^{-\frac{D-R_{M\max}}{R_{\max}-R_{M\max}}} & R_{M\max} \leq D < R_{\max} \\ 1 & D < R_{M\max} \end{cases} \tag{2}$$

where D represents the distance between the UAV and the obstacle center. $R_{\max}$ signifies the maximum radius of the no-fly zone. $R_{M\max}$ denotes the maximum radius of the no-escape zone, assuming the maximum detection range of the detector is equal to the maximum radius of the no-fly zone, i.e., $R_{\max}$. The dangerous level T_p indicates the probability of being destroyed. Assuming the destruction probabilities of different no-fly zones are mutually independent, the overall danger from the no-fly zone center to the UAV is calculated using the following equation [23].

$$T_s = 1 - \prod_{i=1}^{k} (1 - T_p^i) \tag{3}$$

where k represents the number of obstacle zones in the no-fly zone. T_P^i is the probability of the UAV being destroyed when passing through the i-th obstacle zone.

3. Methods

For the problem of path planning, this study adopts the approach of modeling the problem using MDP. The MDP model elements are designed, and an improved DRL algorithm is used for solving. Firstly, we establish the MDP model for this problem.

3.1. MDP Element Design

The three main elements of the Markov Decision Process (MDP) model include the state space, action space, and reward function, which are described as follows.

3.1.1. State

Based on the previous section's modeling and analysis, as shown in Figure 1, the environment considered in this study is two-dimensional. Therefore, we describe the position of the UAV in the environment using coordinate points and take the position as the state variable S of the MDP model.

$$S = (x, y) \tag{4}$$

3.1.2. Action

The deep reinforcement learning algorithm used in this paper has a discrete action space [14]. It requires designing discrete UAV action instructions accordingly. To enhance the efficiency of path planning, the task area is divided into grids. The size of the grids can be designed based on the specific environment, assuming the UAV can freely fly to any of the eight surrounding grids, as shown in Figure 2.

As shown in Figure 2, the discrete action space consists of eight actions. The actions A are represented as Equation (5). The numerical coding (0, 1, 2, 3, 4, 5, 6, 7) in this study serves solely to denote corresponding movement directions, i.e., maneuvering from the current grid position to an adjacent grid position. For instance, 0 signifies moving one grid

north, one south, and seven southeast. The numerical sequence does not strictly correlate; it only pertains to our programming conventions.

$$A = (0, 1, 2, 3, 4, 5, 6, 7) \tag{5}$$

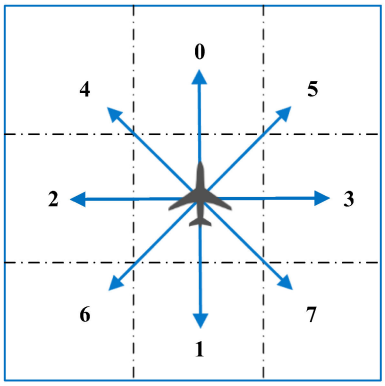


Figure 2. Eight discrete actions.

3.1.3. Reward

The design of the reward function is a critical factor in solving tasks in reinforcement learning. Many reinforcement learning environments suffer from the problem of sparse rewards, where there are no reward values. This can result in slow or even non-convergence of the algorithm. A viable solution is to introduce a reward-shaping technique to guide the agent in learning the optimal policy. In this study, we employ the reward-shaping method to design the reward function, thereby mitigating the issue of sparse rewards.

To address this problem, a well-designed reward function is proposed, which consists of five main components, as shown in Table 1. The reward function R is represented by the following equation:

$$R = r_1 + r_2 + r_3 + r_4 + r_5 \tag{6}$$

where the reward terms r_1, r_{12}, r_3, r_4, and r_5 are defined in Table 1, representing the reward components for the UAV in different states.

Table 1. The specific rewards.

Reward	State of UAV	Reward Value
r_1	In the obstacle area	r_t
r_2	In the target area	+200
r_3	Outside the environmental boundary	−50
r_4	Beyond the maximum step	−50
r_5	Other state	−0.5

The reward term r_t in the table is correlated with the total dangerous level T_s. When T_s exceeds the threshold T_σ, the probability of the UAV being destroyed is 100%, resulting in a penalty of −50. When T_s is below the threshold, the UAV has a certain probability of being destroyed, which is represented as follows:

$$r_t = \begin{cases} -10T_s - 0.5 & 0 \le T_s \le T_\sigma \\ -50 & T_\sigma \le T_s \le 1 \end{cases} \tag{7}$$

When the UAV state satisfies any of the four conditions mentioned in Table 1, the episode terminates; otherwise, the UAV remains in another state. The four conditions mentioned here pertain to the first four states of the UAV. The environment includes target area, obstacle area, and boundary. The first two states refer to the UAV flying to the target area and the obstacle area. The UAV can only move at the boundary of the environment. If

it exceeds the boundary, the round will be terminated. In addition, the simulation also sets the maximum flight step of UAV to describe the maximum flight duration of UAV, which is limited by the fuel carried. The agent receives a relatively small negative penalty (−0.5) to encourage the UAV to reach the target area as quickly as possible.

3.2. Action Selection Policy

During the path planning process and at each time step of the RL algorithm, the agent needs to output an action to make a decision, which is crucial for the policy. Commonly used action exploration policies include ε-greedy policy and Boltzmann exploration policy [20], where ε represents the exploration rate. A higher value of ε indicates that the agent is more inclined to explore the random action space rather than selecting the action with the highest *Q*-value. In this paper, heuristic rules are introduced into the ε-greedy policy [21] to optimize the action selection and improve learning efficiency during training.

The specific process for selecting actions using heuristic search rules is described in Algorithm 1, as shown below. When the UAV agent randomly selects an action from the action space, the action space is further divided based on the current position and target position of the UAV. Subsequently, the UAV agent can randomly select an action within the smaller action space. For example, let us assume that (x_1, y_1) represents the current position of the UAV, and (x_2, y_2) represents the position of its target. In this scenario, if the target position is located to the southeast of the UAV, satisfying $x_1 < x_2$ and $y_1 > y_2$, the UAV agent will randomly select an action from the action space $A_1 = (1, 3, 5, 6, 7)$, rather than attempting all possible actions across the entire action space. In this way, the UAV avoids choosing invalid actions, preventing numerous meaningless training episodes, enhancing training efficiency, and reducing the training cycle.

Algorithm 1. Action selection policy with heuristic search rules.

Input: *Q*-value, UAV location (x_1, y_1), Target location (x_2, y_2).
Output: Action *a*.

```
Generate p randomly, p∈(0, 1)
if x1 < x2:
    if y1 > y2:
        a = random (1,3, 5, 6, 7)
    else:
        a = random (0,3, 4, 5, 7)
    end if
else:
    if y1 > y2:
        a = random (1,2, 4, 6, 7)
    else:
        a = random (0,2, 4, 5, 6)
    end if
end if
```

Moreover, to prevent falling into local optima issues, a single action subspace not only consists of three randomly selected actions in the target direction; it also includes two actions at the boundaries of the target direction. For instance, $A_1 = (1, 3, 5, 6, 7)$ is not simply composed of (1, 3, 7); it also includes actions 5 and 6.

3.3. Improved D3QN Algorithm

Based on the established reinforcement learning model, this study adopts the DQN algorithm for optimization. The key technologies employed by the DQN algorithm include Deep Neural Networks (DNNs) and the Q-learning algorithm. DL models generally refer to Deep Neural Networks designed to simulate the interactive responses of biological neural systems to external stimuli. The fundamental unit of a neural network is the neuron, and the prevalent "M-P neuron model" [24] is currently adopted. Neurons receive input

signals transmitted by other neurons through weighted connections, and these signals are processed and outputted through an activation function. Connecting multiple neurons in a structured hierarchy yields a neural network. Commonly used types of neural networks include Backpropagation (BP) neural networks, Radial Basis Function (RBF) networks, and Convolutional Neural Networks (CNNs) [25]. A basic DNN is employed, with internal layers categorized into input, hidden, and output layers, connected through fully connected links between layers. Among neural networks integrated with reinforcement learning methods, DNNs are widely utilized.

The Q-Learning algorithm [26] is one of the foundational reinforcement learning algorithms, representing a temporal difference offline control algorithm based on value functions. The fundamental principle of Q-learning involves utilizing an extensive Q-table to characterize the value function of states and actions. The algorithm's process relies on employing a greedy strategy to select actions based on states, continually learning and updating its value function by choosing actions that maximize the Q-value, thereby obtaining an optimal strategy. In existing research, the Q-learning algorithm is frequently employed to enhance traditional path planning methods, finding extensive applications in UAV path planning algorithms [27].

However, the traditional DQN algorithm has certain limitations. Firstly, its target Q-values are obtained directly through the greedy method. Using the "max" operation allows the Q-values to quickly approximate the potential optimal targets, but the algorithm is prone to overestimation, leading to significant bias in the learned model. Secondly, the network structure does not sufficiently consider the influence of actions and states, resulting in poor learning performance.

In response to the issues inherent in traditional DQN [28], this paper proposes an improvement and introduces a Prioritized Experience Replay D3QN algorithm that incorporates a heuristic action selection policy. The proposed enhanced algorithm is employed to construct a simulation environment for training and learning purposes. The following section provides an overview of the proposed improvement method.

The DQN algorithm primarily employs an experience replay buffer and a dual-network mechanism. Experience replay involves storing the rewards obtained from each interaction along with state updates, used for updating the target Q-values. During training, small batches of data are randomly sampled from the experience pool to train the neural network. The dual-network mechanism utilizes two Q-networks with identical structures: the current Q-network selects actions and updates model parameters, while the target network Q' computes the target Q-values. All these measures aim to address the issue of data correlation. However, they also introduce challenges in accurately computing the target Q-values. For non-terminal states, the target Q-value calculation formula for DQN is as follows:

$$y_j = R_j + \gamma \max_{a'} Q'(\phi(S'_j), A'_j, w') \tag{8}$$

where w' represents the parameters of the target network. γ represents the discount factor, which is employed to denote the UAV agent's emphasis on future rewards, taking values within the range [0, 1]. The subscript j in the formula denotes the current round, i.e., the j-th round, where R_j signifies the reward obtained in the current round state. $\phi(S)$ denotes the feature vector of state S, used for approximating the state value. Within the designed neural network architecture, there are two components: the evaluation network (the current Q-network) and the target network. To distinguish and train the parameters and weights of these networks, different parameters are used for description and representation. Parameter w represents the evaluation network, while parameter w' represents the target network, corresponding to the network structure depicted in Figure 3. Following the principles of DDQN [29], we decouple the action selection for target Q-values from the computation of target Q-values. Instead of directly identifying the maximum Q-value in the target Q-network, we first determine the action corresponding to the maximum Q-value in the

current Q-network and subsequently calculate the Q-value based on that action. Therefore, the above expression can be updated as follows:

$$y_j = R_j + \gamma Q'(\phi(S'_j), \text{argmax}_{a'} Q(\phi(S'_j), A, w), w') \tag{9}$$

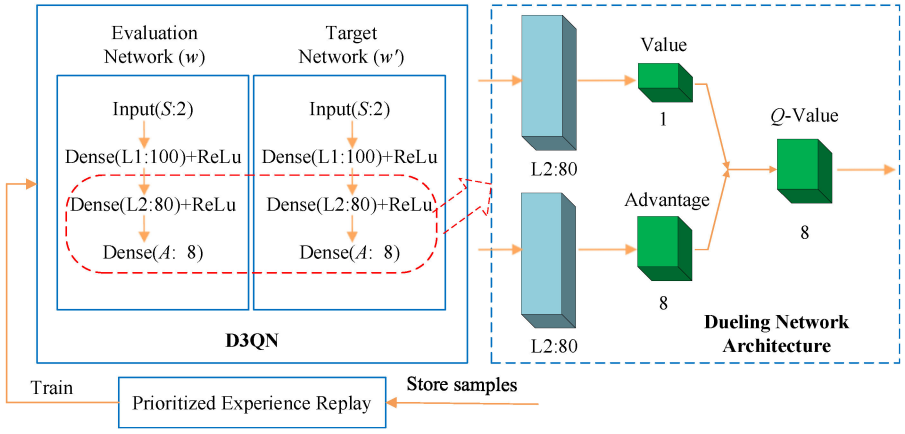


Figure 3. The network structure of the improved algorithm.

Furthermore, this study extends the traditional DQN algorithm by introducing certain modifications to its neural network structure, incorporating a competitive architecture [30]. While the DQN algorithm employs a neural network structure with a direct connection from the hidden layer to the output layer, the enhanced network introduces an additional layer between the hidden layer and the output layer. This intermediate layer divides the Q-network output into two parts (Figure 3): one part is solely dependent on the state S and independent of the action A, termed the value function $V(S, w, \alpha)$), and the other part is related to both S and A, referred to as the advantage function $A(S, A, w, \beta)$. The final value function can be redefined as follows:

$$Q(S, A, w) = V(S, w, \alpha) + (A(S, A, w, \beta) - \frac{1}{|\mathrm{A}|} \sum_{a' \in A} A(S, a', w, \beta)) \tag{10}$$

where w represents the shared parameters, α denotes the parameters of the value function V, and β represents the parameters of the advantage function. The cardinality of A, denoted as $|\mathrm{A}|$, equals the size of the action space.

In the data sampling process of DQN, all samples have equal sampling probabilities, which is inefficient. To address this, we enhance the algorithm's training efficiency by introducing a prioritized experience replay mechanism [31]. Building upon the mentioned improvements, we further optimize the logic of the algorithm's experience replay component. The fundamental concept of prioritized experience replay is to assign larger sampling weights to states with higher learning efficiency. In the experience replay pool, samples with larger TD errors exert a greater influence on the Backpropagation process, while samples with smaller TD errors have a relatively minor impact on the computation of backward gradients. This prioritization ensures that samples with larger absolute TD errors are more likely to be sampled. Due to the variation in sample weights, the loss function of the network has also been optimized. The improved equation is as follows:

$$Loss = \frac{1}{n} \sum_{j=1}^{n} \omega_j (y_j - Q(S_j, A_j, w))^2 \tag{11}$$

where ω_j represents the priority weight of the j-th sample. After updating the gradients of the network parameters, it is necessary to recalculate the TD error and update it in the tree structure, followed by storage.

According to the mentioned improvements, the structure of the neural network is as illustrated in Figure 3. Both the evaluation network and the target network share the same architecture, consisting of an input layer, a hidden layer composed of two fully connected layers (L1, L2), and an output layer. The number of neurons in the input layer corresponds to the size of the state space, while the two fully connected layers, L1 and L2, consist of 100 and 80 neurons, respectively. To expedite the learning process during the training of network parameters with samples selected from the experience pool, a prioritized experience replay mechanism is employed. On the right side of the figure, the competitive network architecture is depicted, consisting of two fully connected layers (L1, L2) representing the structural flow of two competitive systems. The final fully connected layer of the value function has one output, while the advantage function part has eight outputs, aligning with the action space. To accelerate training convergence, rectified linear units (ReLU) [32] are chosen as the activation functions for the fully connected layers in the hidden layer.

The algorithm we utilized for training also incorporated the heuristic action selection policy outlined in Algorithm 1, which similarly contributed to enhancing training efficiency. Throughout the actual training process, the exploration rate ε diminishes progressively. ε_0 denotes the initial exploration rate of the UAV agent in Equation (12). ε_1 denotes the final exploration rate. Initially, ε is relatively high, enabling the UAV to explore different actions with a higher probability. As the number of training iterations increases, ε decreases gradually until it stabilizes. At this stage, the agent tends to favor actions with higher Q-values. The evolution of ε is defined by the following equation, where i denotes the current training round, i_{pre} represents the pre-training iterations designed to generate a certain amount of data, $i_{descend}$ indicates the number of exploration rate decay steps, and $i_{\max}$ signifies the maximum training iterations.

$$\varepsilon = \begin{cases} \varepsilon_0 & 0 \le i < i_{\text{pre}} \\ \varepsilon_0 - \frac{(i - i_{\text{pre}})(\varepsilon_0 - \varepsilon_1)}{i_{\text{descend}}} & i_{\text{pre}} \le i < i_{\text{descend}} \\ \varepsilon_1 & i_{\text{descend}} \le i < i_{\max} \end{cases} \tag{12}$$

The procedural steps for the proposed enhanced algorithm are detailed in Algorithm 2. In this algorithm, s, a, r, and d represent the environment state, action, reward, and terminal state of the reinforcement learning MDP, respectively. The parameter T signifies the number of learning iterations, and the termination state of each iteration is determined based on the reward function.

The implementation workflow of the training algorithm is illustrated in Figure 4. The process is divided into two parts: the environment and the UAV agent. Firstly, based on the designed environmental scenario, we conducted modeling. Subsequently, an intelligent agent was designed based on the Markov Decision Process (MDP) model. At each moment, the environment updates state information for the intelligent agent. The agent, following the steps outlined in Algorithms 1 and 2, accomplishes data updates and policy learning. Moreover, at each time step, the agent provides chosen actions to the environment, closing the loop of the process. Through extensive training, the objective of optimizing the policy is achieved.

Algorithm 2. Improved D3QN considering action selection policy with heuristic rules.

1. **Initialize** Q network parameter w, target Q network parameter $w = w'$.
2. **Initialize** replay memory D with capacity N, the priority of all Sum Tree leaf nodes $p_j = 1$.
3. **for** $i = 1$ **to** T **perform**
4. **Initialize** s as the first state of UAV.
5. **while** s is not Termination:
6. Choose an action according to **Algorithm 1**.
7. Execute action a. Transfer to the next state s', and obtain the immediate reward r. Judge whether it is in the termination state d.
8. Store transition $\{s, a, s', r, d\}$ in D. Replace the oldest tuple if $\|D\| > N$.
9. Sample N_b tuples from D, $\{s_j, a_j, s'_j, r_j, d_j\}, j = 1, 2, 3, \ldots, n$. The probability is $P_j = p_j / \sum_i p_i$. Compute the weight of loss function: $\omega_j = \left(N * P_j\right)^{-\beta} / \max_i \omega_i$.
10. Compute the current target Q-value y_i. $y_i = r_j + (1 - d)\gamma Q'\left(s'_j, argmax_{a'} Q\left(s'_j, a, w\right), w'\right)$
11. Compute the loss as Equation (11). Update Q network parameter w.
12. Compute TD error for sample data: $\delta_j = y_j - Q\left(s_j, a_j, w\right)$. Update the priority for all Sum Tree nodes: $p_j = \left|\delta_j\right|$.
13. For every N' step, update the target Q network parameter $w' = w$.
14. $s = s'$.
15. **end for**

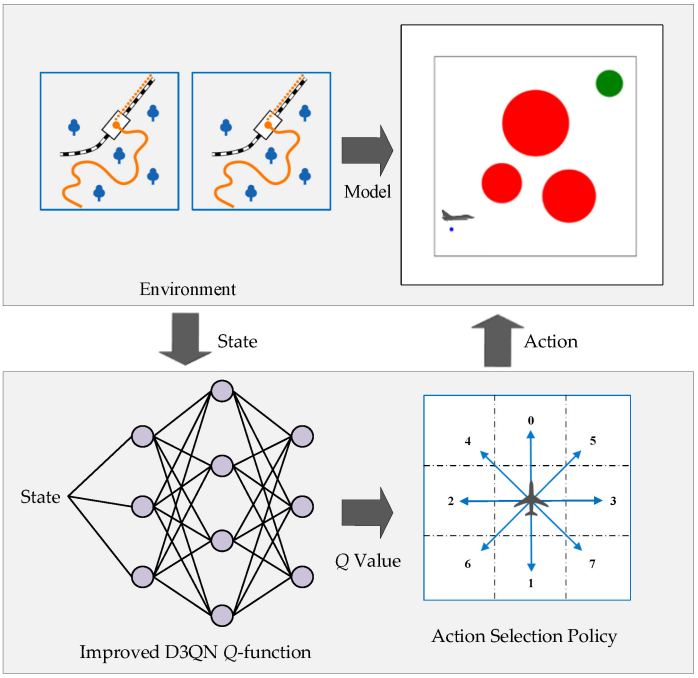


Figure 4. The workflow of the training algorithm.

4. Results

This section primarily focuses on the training and simulation results testing. The mission area for the UAV is a rectangular region measuring 60 × 60 km, as depicted in Figure 4. Within this mission area, there is one reconnaissance target and three adversarial obstacles represented as circles. The boundaries of the obstacle areas have a radius of $R_{\max}$. The planned location for the reconnaissance target is (52, 52), and the parameters for the obstacle zones are detailed in Table 2.

Table 2. The obstacle area parameters.

Obstacle Areas	Location of Center (km)	$R_{\max}$ (km)	$R_{M\max}$ (km)
Obstacle 1	(20, 22)	6	5
Obstacle 2	(30, 40)	10	9
Obstacle 3	(40, 18)	8	7

The improved algorithm presented in this paper incorporates several hyperparameters, with some parameters exhibiting sensitivity to the environment. Inappropriately set parameter values may hinder the algorithm's convergence. Therefore, based on extensive numerical simulations, the hyperparameters for the algorithm in this section are specified in Table 3.

Table 3. Setting of simulation parameters.

Parameter	Value	Parameter	Value
The radius of the target area	4 km	Target network update rate	0.008
Max episodes i	10,000	Discount factor	0.96
Max steps of UAV	500	Q network learning rate	0.0005
Replay memory capacity N	10,000	Initial exploration	1
Minibatch size N_b	32	Final exploration	0.1
Number of pre-training rounds	200	Number of annealing	2000

The maximum flight distance for the UAV is set at 500, with the step length representing the UAV's flight distance. If the UAV fails to reach the target area within 500 steps, the round will terminate. In the initial simulation phase, a pre-training of 200 rounds is conducted to generate data in the experience replay pool. The weights of the target network are adjusted to the initial network every eight steps. The exploration–exploitation parameter ε is linearly annealed from 1 to 0.1 over 2000 rounds and then held constant at 0.1.

4.1. Training Results

The algorithmic environment in this study is implemented using Python programming, employing the TensorFlow learning library and TensorBoard visualization tool for analysis. The optimization algorithm for network parameters is Adam. To introduce training randomness, at the beginning of each round, the initial position of the UAV is randomly selected within a designated mission area. The distance along the X-axis is within (0, 30) km, and along the Y-axis, it is within (0, 15) km. The training reward smoothing curve is illustrated in Figure 5, where the horizontal axis represents the number of training rounds, and the vertical axis represents the cumulative rewards per round. In this section, comparative simulations are conducted, contrasting the improved D3QN algorithm with the DDQN, and DQN algorithms with essentially similar parameters and network structures in the same scenario.

As depicted in the figure, the reward curves for DQN and DDQN eventually converge to a similar pattern, with a slight difference in convergence speed. However, DDQN achieves a more stable cumulative reward, and its convergence is faster. DDQN reaches its maximum after 4000 rounds of training, whereas DQN requires 6000 rounds and exhibits greater fluctuations. This indicates that DDQN effectively addresses overestimation issues, enhancing algorithm stability. The reward convergence speed of the proposed improved D3QN method is slower than that of DQN and DDQN, gradually converging over 6000 rounds. However, its final reward value surpasses that of DQN and DDQN under the same conditions. This is attributed to the fact that the Q network in D3QN, with a separated data stream structure, can more effectively and accurately approximate the Q function, demonstrating the advantages of the proposed method.

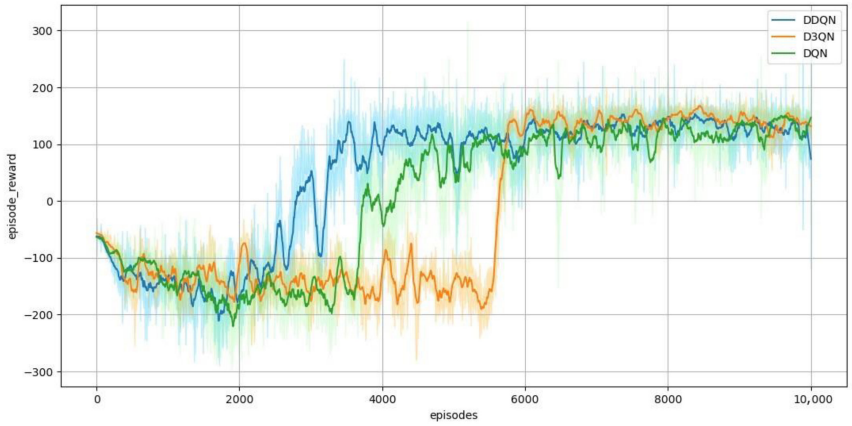
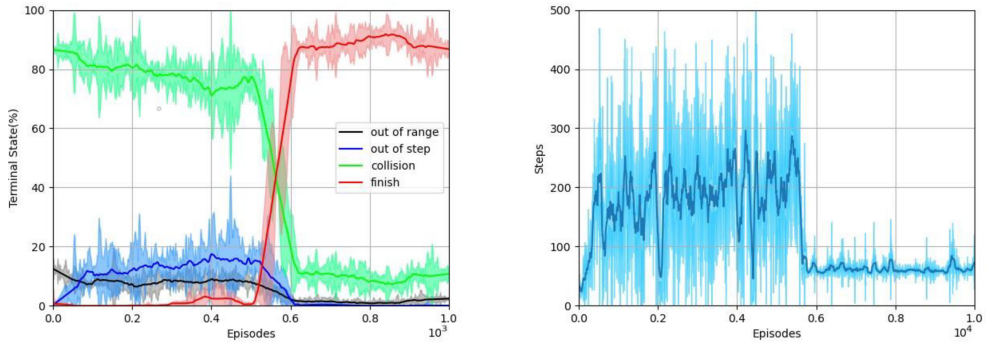


Figure 5. The smoothing curve for cumulative rewards of three algorithms.

Figure 6 displays the terminal states of the UAV along with changes in flight distances, with terminal states calculated every 10 rounds. The UAV's terminal states include task completion (finish) and non-completion, which encompasses scenarios such as exceeding the boundary range (out of range), surpassing the maximum flight distance (out of step), or encountering destruction upon entering obstacle zones (collision).



(**a**) The terminal Status (**b**) The steps of UAV

Figure 6. The terminal state and UAV step.

From the figure, within the initial 5000 rounds, the UAV faces challenges in completing the planned tasks, with a high probability of entering obstacle zones and being destroyed, and a lower probability of flying out of the task area or exceeding the flight distance. As the learning process progresses and the agent's policy optimizes, the task completion rate surpasses 90%. The rates of flying out of the task area or exceeding the flight distance tend towards 0%, while the probability of destruction in obstacle zones remains below 10%. According to Figure 6, the required flight distances for the UAV continue to decrease, indicating a gradual convergence of the UAV's planned path length towards optimality.

4.2. Results in Static Scenario

To assess the effectiveness of the proposed algorithm, this section conducts multiple simulations and compares them with other path planning algorithms. The initial position of the UAV is set at (5, 8). Figure 7 illustrates trajectories generated by various planning algorithms. In the figure, the red color represents obstacle zones, while the green color indicates the target area. When plotting the boundaries of obstacle zones in this paper, only the region corresponding to the $R_{\max}$ radius is displayed. The red area's radius in the figure is the detection radius of the corresponding obstacle zone. During simulation, the parameters for DQN and DDQN are the same as those for D3QN. The A* algorithm uses a neighborhood search number of 8, the RRT-GoalBias algorithm has an expansion step size of 1, a target sampling rate of 0.1, and a maximum iteration of 250 rounds.

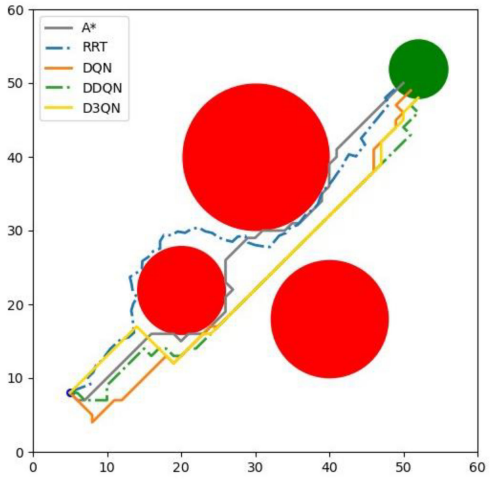
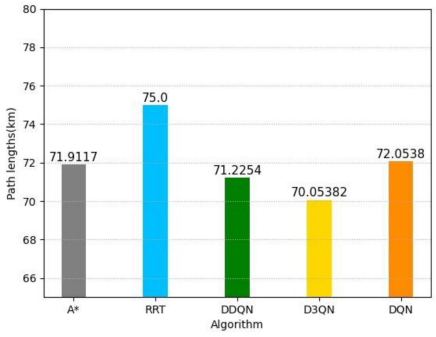


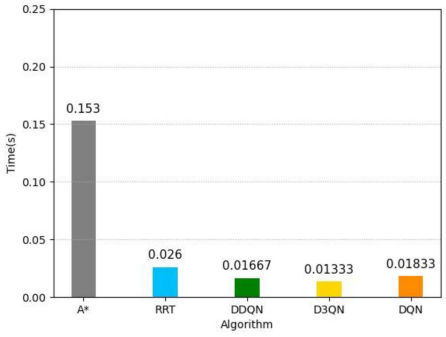
Figure 7. The trajectory planning results of multiple algorithms.

All planning methods successfully found safe paths. Compared to other algorithms, the improved D3QN method in this study obtained paths that are further away from obstacle zones, thereby better ensuring the UAV's safety and reducing the probability of being destroyed.

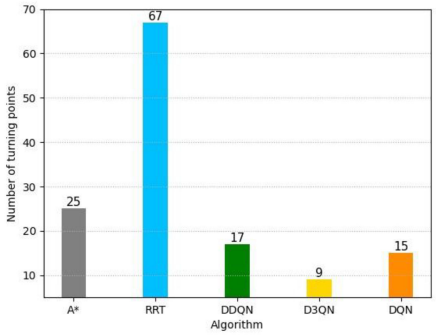
Figure 8 provides quantitative results from three aspects: path length, planning time, and the number of turning points. Simulations were conducted using the same computer under identical conditions to compute the time required for planning UAV paths. The prioritized experience replay D3QN path planning algorithm resulted in the shortest path length. Compared to A* and RRT-GoalBias, the paths generated by DDQN and DQN were shorter in length. Clearly, these results illustrate the superiority of the algorithm proposed in this article, showcasing an improvement in both the efficiency and quality of path planning compared to existing research.



(**a**) Path length.



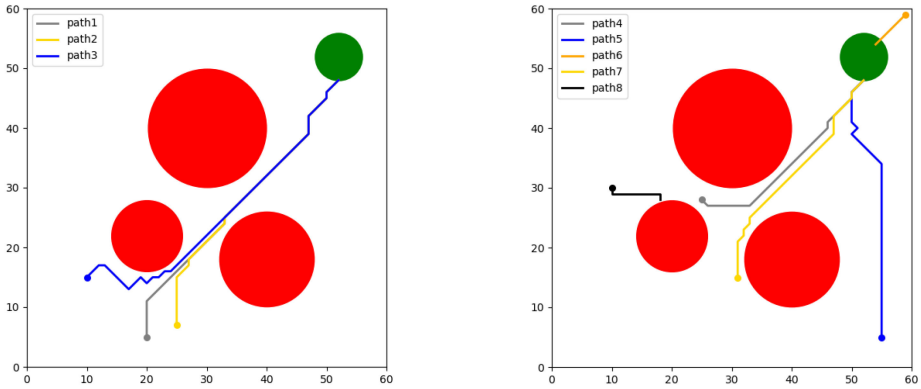
(**b**) Plan time.



(**c**) Number of turning points.

Figure 8. The qualitative comparison results of multiple algorithms.

From the figures, it is evident that these methods require relatively short amounts of time, with minor differences, yet the improved D3QN requires the least amount of time. As post-planning path smoothing is necessary for these methods, fewer turning points imply less time required for smoothing, indicating that this method is more conducive to efficiency enhancement. Figure 9 illustrates that the improved D3QN has fewer turning points in its path, demonstrating the advantages of the proposed UAV path planning algorithm.



(**a**) Result one (**b**) Result two

Figure 9. The results for initial position change of UAV.

Table 4 presents a comparative analysis of three RL algorithms. The paths generated by improved D3QN require 51 steps to reach the target area, resulting in a cumulative reward of 175.0. In contrast, both DDQN and DQN need 53 steps to achieve rewards of 174.5 and 174.0, respectively. This indicates that the paths obtained by improved D3QN are superior, with higher rewards. In summary, in a static scenario, the path planning capability of the algorithm proposed in this study surpasses that of the planning algorithms employed by DDQN and DQN, as well as A* and RRT-GoalBias algorithms.

Table 4. The comparison of RL algorithms.

Algorithm	DQN	DDQN	Improved D3QN
Steps of UAV	53	53	51
Cumulative reward	174.0	174.5	175.0

Due to the random initialization of UAVs in a small area during training scenarios, this section not only conducts planning verification with a fixed starting position but also includes multiple validation sets. To assess the generalization performance of the obtained model, tests are conducted in non-training scenarios, as shown in Figure 9 and Table 5.

Table 5. The results for non-training scenarios.

Path Number	Initial Position	UAV Step	Cumulative Reward
1	(20, 5)	43	179.0
2	(25, 7)	41	188.0
3	(10, 15)	46	177.5
4	(25, 28)	29	188.0
5	(55, 5)	43	179.0
6	(59, 59)	5	198.0
7	(31, 15)	33	184.0
8	(10, 30)	11	−55.0

The first three sets have UAV starting points within the training area, and the trained model already possesses corresponding training policy. For the subsequent five sets

simulating non-training scenarios, as depicted in Figure 9, UAVs in training scenarios successfully plan paths. When the starting position is outside the training area, in most cases, paths are successfully planned (e.g., path4–path7). However, there is still one set of results where the UAV is guided into the obstacle zone and destroyed, as seen in path8, indicating a failure to plan a path.

4.3. Visualized Action Field

To clearly illustrate the effectiveness of the intelligent policy proposed in this study, a visualized action space is established in this section following the methods outlined in references [33,34]. This visualized action space is designed to showcase the UAV path planning policy within the task area. It provides a more intuitive representation of the control policies of the path planning controller designed in this section, as depicted in Figures 10 and 11.

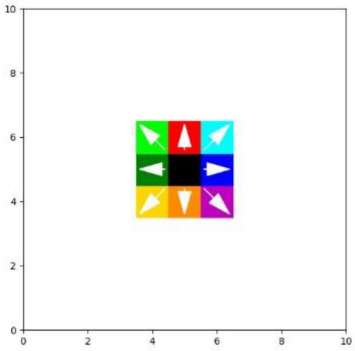
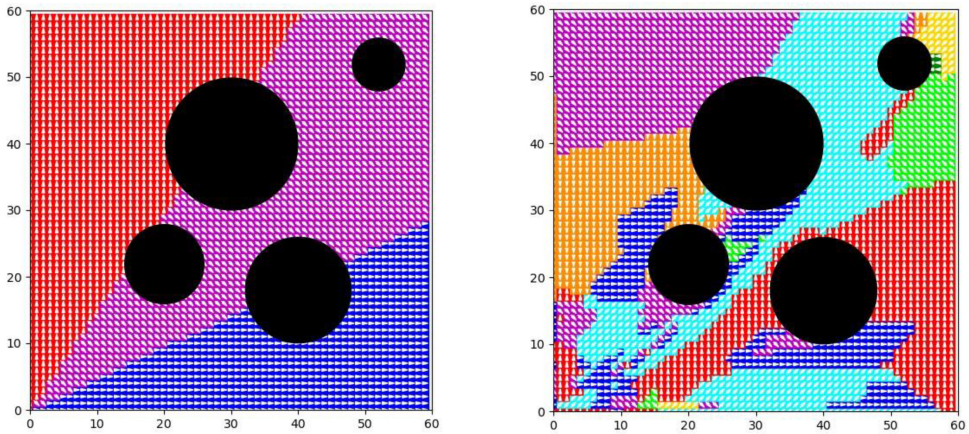


Figure 10. The diagram of visual action field rules.



(**a**) Before training (**b**) After training

Figure 11. The visual action field is based on improved D3QN.

As illustrated in Figure 10, eight discrete actions are represented by distinct colors. The white arrow at the center signifies the velocity direction the UAV should adopt at the current state, while the black portion indicates the absence of a flight command.

The action spaces before and after training were obtained according to the rules. The left side of Figure 7 depicts the visualized action space before training, following the random initialization of network parameters. The right side displays the results after 10,000 training iterations. To emphasize the training policy, four black circular regions represent three obstacle zones and one target area within the scene. The target area is situated in the upper right corner, while the remaining circles denote obstacle zones. It can be observed that the action space before training exhibits only three colors, indicating

a random and irregular distribution that lacks meaningful implications for early-stage UAV path planning. After training optimization, different positions display distinct action commands, and the overall trend is oriented towards the target area. This suggests that the intelligent agent's policy has been optimized, enabling the UAV to fly towards the target based on the control instructions provided by the policy.

However, there are still small areas, such as the surrounding boundaries, where the action instructions are not entirely rational. This explains the occurrence of path planning failures in the later stages of training. The visualized results of the policy in this section correspond to the static scene planning results from the previous section.

Additionally, Appendix A provides the visualized action field for DDQN and DQN. It can be observed that both the DDQN and DQN algorithms only generated feasible paths when addressing this path planning problem. In terms of policy applicability, the improved D3QN method proposed in this paper yields a superior D3QN path planning policy compared to DDQN and DQN algorithms.

4.4. Results in Dynamic Scene

This section conducts algorithm simulations based on dynamic scenes. The dynamic environment implies that the obstacle zones in the mission area are no longer fixed; the obstacle center is mobile. Consequently, the detection range of obstacle zones varies with the movement of the obstacle center. This paper assumes that the no-fly zone center follows a pre-defined trajectory. In this section, building upon the static scenes described earlier, all three obstacle zones are in motion, each with distinct movement directions and velocities. The planning objective remains unchanged, and the initial position of the UAV is randomly selected within a small mission area.

The model trained in the static scene is further fine-tuned in the dynamic scenario. The initial parameters of the model are based on the best weight allocation obtained during training in the static scene, rather than being randomly assigned. This approach helps avoid many rounds of ineffective exploration. The algorithm hyperparameters remain fundamentally the same as those in Table 3, with an increase in the maximum training rounds to 20,000. Pre-training rounds are no longer set, and adjustments are made to the learning rate and target network update frequency, as specified in Table 6. The initial parameters of the obstacle center, detection radius, and strike radius remain unchanged. Figure 12 presents the smoothed curve of cumulative rewards and the success rate of UAV path planning obtained from the dynamic scene.

Table 6. The hyperparameters in dynamic scenes.

Parameter	Value	Parameter	Value
Max episodes i	20,000	Number of pre-training	0
Replay memory capacity N	50,000	Q network learning rate	0.00005
Target network update rate	0.0004		

During the training process, cumulative rewards are recorded and averaged every 10 rounds. Similarly, the success rate of path planning is calculated every 10 rounds. This yields curves depicting cumulative rewards, smoothed average rewards, and the success rate over time. The reward values (blue) and success rate (red) are plotted using different y-axes, displayed on the left and right sides of the graph in distinct colors.

It can be observed that in the initial 8000 rounds of training in the dynamic environment, the average cumulative reward obtained by the UAV is -55, with an average reward of -180. This is significantly lower than the average cumulative reward and average reward obtained in the static scene. During this period, the UAV agent has not yet learned the correct path planning policy. This is primarily attributed to the increased difficulty in UAV path planning due to the variability and complexity introduced by the dynamic environment, making it challenging for the agent to achieve higher rewards.

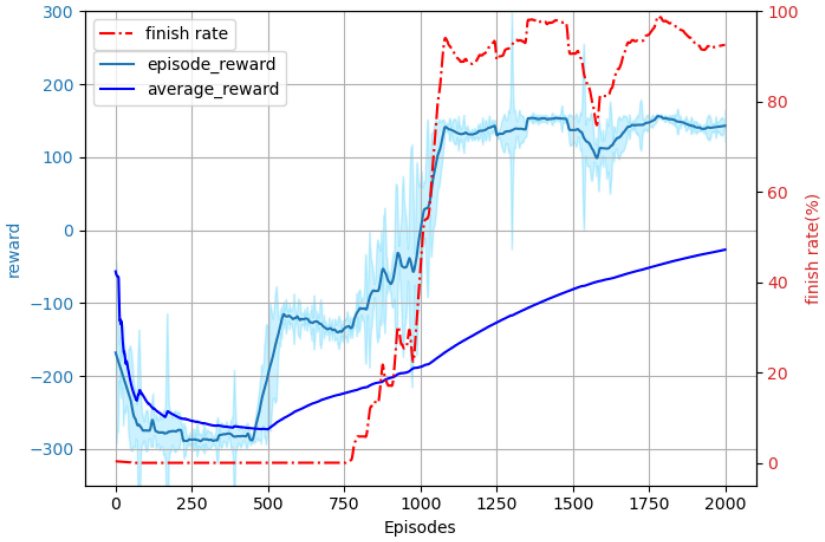


Figure 12. The smoothing curve of cumulative rewards and success rate in dynamic scenarios.

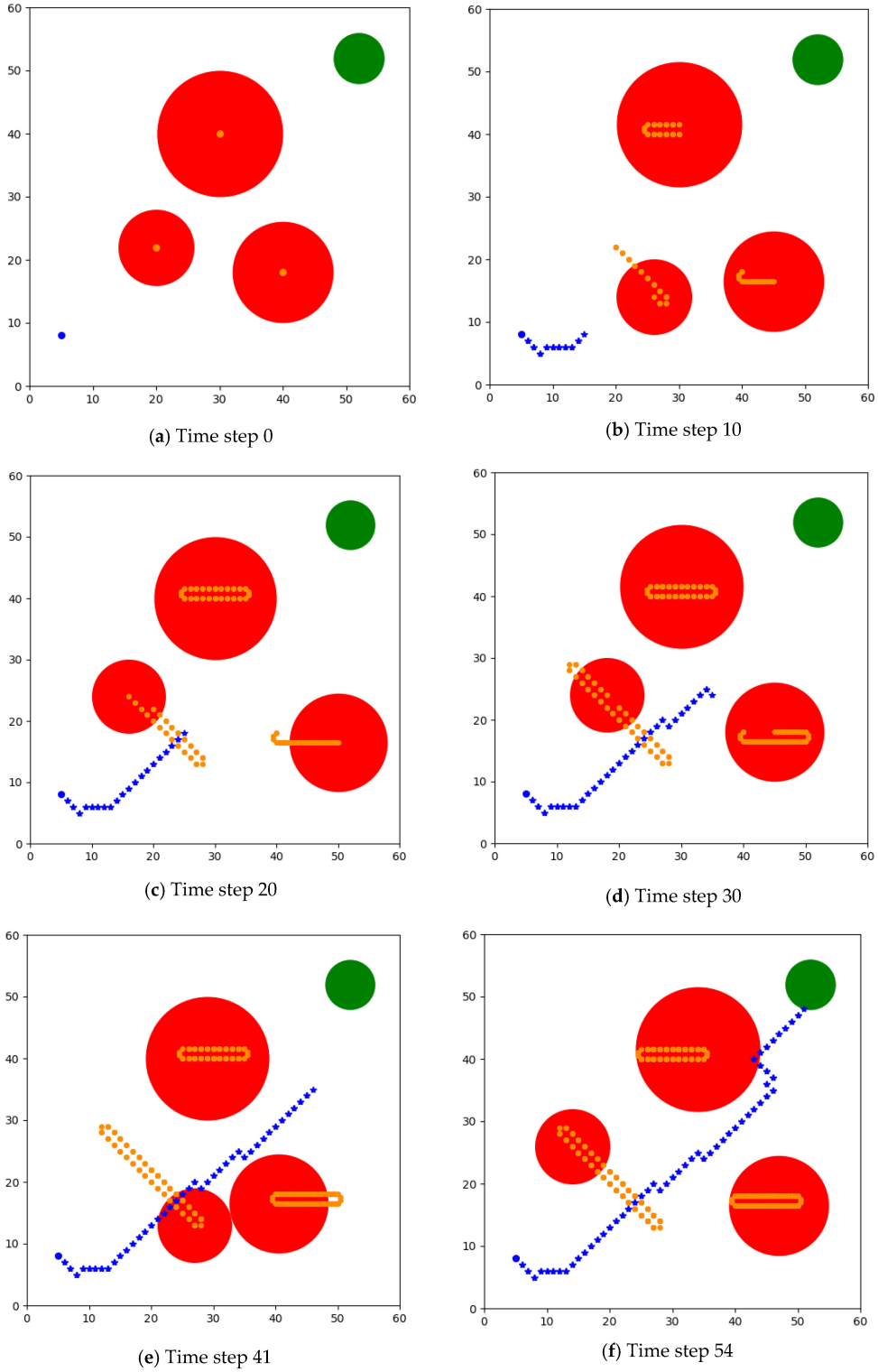
According to the reward curve, after 10,000 rounds of training, there is a noticeable increase in the reward values, gradually reaching a maximum. The reward values converge to the maximum in the final 2500 rounds. The trend in the success rate curve closely mirrors that of the reward values. In the initial 10,000 rounds, the success rate is essentially 0, but as the training progresses, the path planning policy is effectively optimized, and the success rate approaches 95%.

As shown in Figure 13, the simulated results of a test in a dynamic scenario are presented in the form of a sequence of motion trajectories. This is performed to validate the generalization capability of the improved D3QN path planning model obtained through training. In the simulation test, the exploration rate ε is set to 0, indicating that the UAV fully utilizes the predicted Q-values from the trained model to select its flight action commands. The initial position of the UAV is set at (5, 8), and the cumulative reward obtained by the UAV intelligent agent in this simulation is 175.0.

In the figure, the red and green areas represent the coverage zones of obstacles and targets, respectively. The blue dot indicates the initial position of the UAV, while the blue asterisk represents the current position of the UAV along with its historical motion trajectory. The orange solid dots correspond to the current positions and historical motion trajectories of each obstacle zone's center. Among these, two move back and forth within a certain distance in the X-direction, one moves back and forth along an inclined direction, and the obstacle zone in the bottom-right corner moves at a slower speed compared to the other two obstacle zones.

The UAV successfully reached the target area, as depicted in the figure, while effectively avoiding the search of obstacle zones. This indicates that the trained policy model is capable of planning a safe and feasible path. Additionally, multiple sets of simulations were conducted in this section. Regardless of the UAV's initial position being arbitrarily chosen within the small area defined during training, the DRL path planning model consistently avoided obstacle zones and generated viable UAV paths.

In conclusion, we have trained a reinforcement learning path planning policy for the static scenarios described in Sections 4.1 and 4.2. The simulation results indicate that compared to A*, RRT, DQN, and DDQN, this policy performs better in terms of path length and planning time, significantly improving planning efficiency. For the dynamic scenarios described in Section 4.3, the reinforcement learning planning policy can control the UAV to navigate around threat zones effectively, demonstrating good path planning performance.



(**a**) Time step 0 (**b**) Time step 10

(**c**) Time step 20 (**d**) Time step 30

(**e**) Time step 41 (**f**) Time step 54

Figure 13. Test results of successful path planning for UAV in dynamic scenarios.

5. Conclusions

In addressing the UAV path planning problem in dynamic scenarios, this study proposes an enhanced deep reinforcement learning algorithm as a solution. Through extensive simulation training in the TensorFlow framework, distinct reinforcement learning path planning policy for static and dynamic scenarios are obtained. Validations across various scenarios indicate that the proposed path planning strategy is capable of effectively

devising secure paths while avoiding adversary obstacle zones. The primary contributions are outlined as follows:

(1) The MDP model for UAV path planning is translated into the framework of reinforcement learning. The design encompasses the definition of state space, action space, and reward functions, while heuristic rules are incorporated into the action exploration strategy.
(2) An improved deep reinforcement learning approach is proposed, employing the Q-function approximation of the prioritized experience replay D3QN to estimate the action Q-values of the agent. The algorithm's network structure is designed based on the TensorFlow framework, combining double Q-learning, a competitive architecture of neural networks, and prioritized experience replay. The resulting algorithmic flow is based on the improved D3QN for path planning.
(3) Reinforcement learning path planning policy for static scenarios is trained and subjected to simulation analysis. The results indicate that, compared to A*, RRT, DQN, and DDQN, this strategy demonstrates superior performance in terms of path length and planning time. A visualized action space is utilized to analyze the learned path planning policy of the intelligent agent.
(4) Building upon the static scenario, further training is conducted to derive reinforcement learning planning policy for dynamic scenarios, with simulation testing in typical scenarios. The results demonstrate the successful navigation of the UAV to the target area, showcasing the algorithm's robust performance by effectively avoiding search in obstacle zones.

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

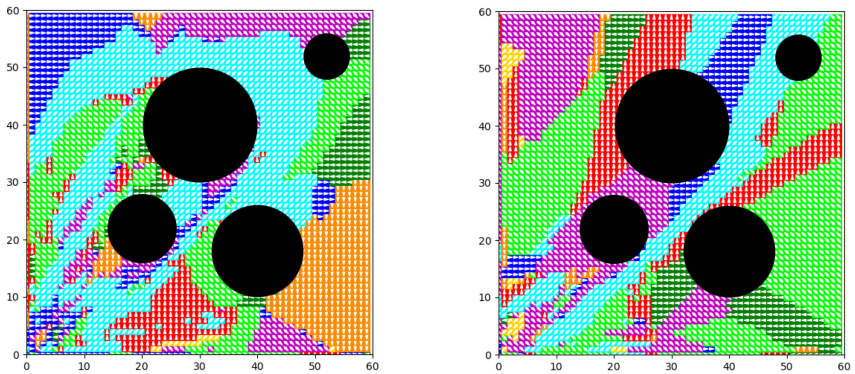
Funding: This study received funding from the National Natural Science Foundation of China (No. 12002370).

Data Availability Statement: All data are included in the article.

Acknowledgments: The authors thank the editor and anonymous reviewers for their helpful comments on improving the quality of this article. They would also like to thank their laboratory team members for the technical support.

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

Appendix A



(**a**) DQN (**b**) DDQN

Figure A1. The visualized action field for DDQN and DQN.

References

1. Bulka, E.; Nahon, M. Automatic control for aerobatic maneuvering of agile fixed-wing UAVs. *J. Intell. Robot. Syst.* **2019**, *93*, 85–100. [CrossRef]
2. Chen, Q.; Jin, Y.; Han, L. A review of research on unmanned aerial vehicle path planning algorithms. *Aerodyn. Missile J.* **2020**, *5*, 54–58. [CrossRef]
3. Chen, G. Application of improved A* algorithm in robot path planning. *Electron. Des. Eng.* **2014**, *19*, 96–98.
4. Liu, Y.; Dai, T.; Song, J. Research of path planning algorithm based on improved artificial potential field. *J. Shenyang Ligong Univ.* **2017**, *1*, 61–65.
5. LaValle, S. Rapidly-exploring random trees: A new tool for path planning. *Res. Rep. 9811* **1998**, 293–308.
6. Li, Y.; Zhang, Q. Overview of indoor unknown environment traversal path planning algorithms. *Comput. Sci.* **2012**, *39*, 334–338.
7. Xu, X.; Liang, R.; Yang, H. Path planning for agent based on improved genetic algorithm. *Comput. Simul.* **2014**, *31*, 357–361.
8. Kang, B.; Wang, X.; Liu, F. Path planning of searching robot based on improved ant colony algorithm. *J. Jilin Univ.* **2014**, *44*, 1062–1068.
9. Li, Q.; Zhang, C.; Chen, P. Improved ant colony optimization algorithm based on particle swarm optimization. *Control. Decis.* **2013**, *28*, 873–879.
10. Wang, L.; Li, M. Application of Improved Adaptive Genetic Algorithm in Mobile Robot Path Planning. *J. Nanjing Univ. Technol. Nat. Sci. Ed.* **2017**, *41*, 627–633.
11. Shi, E.; Chen, M.; Li, J. Research on global path planning method for mobile robots based on ant colony algorithm. *Trans. Chin. Soc. Agric. Mach.* **2014**, *45*, 53–57.
12. Wang, X.; Shi, Y.; Ding, D. Double global optimum genetic algorithm-particle swarm optimization-based welding robot path planning. *Eng. Optim.* **2016**, *48*, 299–316. [CrossRef]
13. Contreras, M.; Ayala, V.; Hernandez, U. Mobile robot path planning using artificial bee colony and evolutionary programming. *Appl. Soft Comput. J.* **2015**, *30*, 319–328. [CrossRef]
14. Mnih, V.; Kavukcuoglu, K.; Silver, D. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef]
15. Zhao, Y.; Zheng, Z.; Zhang, X.; Liu, Y. Q learning algorithm-based UAV path learning and obstacle avoidance approach. In Proceedings of the 36th Chinese Control Conference, Dalian, China, 26–28 July 2017; pp. 95–100.
16. Zhou, B.; Guo, Y.; Li, N.; Zhong, X. Drone path planning based on directed reinforcement Q-learning. *Acta Aeronaut. ET Astronaut. Sin.* **2021**, *42*, 506–513.
17. Huang, D.; Jiang, C.; Han, K. A 3D path planning algorithm based on deep reinforcement learning. *Comput. Eng. Appl.* **2020**, *56*, 30–36. [CrossRef]
18. Feng, S.; Shu, H.; Xie, B. 3D environment path planning based on improved deep reinforcement learning. *Comput. Appl. Softw.* **2021**, *38*, 250–255. [CrossRef]
19. Cao, J.; Wang, X.; Wang, Y.; Tian, Y. An improved Dueling Deep Q-network with optimizing reward functions for driving decision method. *Proc. Inst. Mech. Eng. Part D J. Automob. Eng.* **2023**, *237*, 2295–2309. [CrossRef]
20. Li, S.; Xin, X.; Lei, Z. Dynamic path planning of a mobile robot with improved Q-learning algorithm. In Proceedings of the 2015 IEEE International Conference on Information and Automation, Lijiang, China, 8–10 August 2015; pp. 409–414.
21. Yan, C.; Xiang, X.; Wang, C. Towards Real-Time Path Planning through Deep Reinforcement Learning for a UAV in Dynamic Environments. *J. Intell. Robot. Syst.* **2020**, *98*, 297–309. [CrossRef]
22. Gao, Y.; Xiang, J. New threat assessment non-parameter model in beyond-visual-range air combat. *J. Syst. Simul.* **2006**, *18*, 2570–2572.
23. Wen, N.; Su, X.; Ma, P.; Zhao, L.; Zhang, Y. Online UAV path planning in uncertain and hostile environments. *Int. J. Mach. Learn. Cybern.* **2017**, *8*, 469–487. [CrossRef]
24. Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. A general reinforcement learning algorithm that master chess, shogi, and go through self-play. *Science* **2018**, *362*, 1140–1144. [CrossRef]
25. Silver, D.; Huang, A.; Maddison, C.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. Mastering the game of Go with deep neural networks and tree search. *Nature* **2016**, *529*, 484–489. [CrossRef] [PubMed]
26. Sutton, S.; Barto, G. Reinforcement learning: An introduction. *IEEE Trans. Neural Netw.* **1998**, *9*, 1054–1065. [CrossRef]
27. Liu, Z.; Lan, F.; Yang, H. Partition Heuristic RRT Algorithm of Path Planning Based on Q-learning. In Proceedings of the 2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), Chengdu, China, 20–22 December 2019.
28. Tai, L.; Liu, M. Towards cognitive exploration through deep reinforcement learning for mobile robots. *arXiv* **2016**, arXiv:1610.01733.
29. Van-Hasselt, H.; Guez, A.; Silver, A. Deep reinforcement learning with double Q-learning. *Proc. AAAI Conf. Artif. Intell.* **2015**, *30*, 2094–2100. [CrossRef]
30. Wang, Z.; Schaul, T.; Hessel, M.; Hasselt, H.; Lanctot, M.; Freitas, N. Dueling network architectures for deep reinforcement learning. In Proceeding of 33rd International Conference on Machine Learning (ICML), New York, NY, USA, 20–22 June 2016; pp. 1995–2003.

31. Schaul, T.; Quan, J.; Antonoglou, I.; Silver, D. Prioritized experience replay. *Comput. Sci.* **2015**, 1–17. [CrossRef]
32. Maniatopoulos, A.; Mitianoudis, N. Learnable Leaky ReLU (LeLeLU): An Alternative Accuracy-Optimized Activation Function. *Information* **2021**, *12*, 513. [CrossRef]
33. Sui, Z.; Pu, Z.; Yi, J.; Xiong, T. Formation control with collision avoidance through deep reinforcement learning. In Proceedings of the International Joint Conference on Neural Networks, Budapest, Hungary, 14–19 July 2019; pp. 1–8.
34. Xie, N.; Hu, Y.; Chen, L. A distributed multi-agent formation control method based on deep Q learning. *Front. Neurorobotics* **2022**, *16*, 817168. [CrossRef]