IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 3 January 2024, accepted 10 January 2024, date of publication 15 January 2024, date of current version 15 April 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3354075

RESEARCH ARTICLE

Path Planning for Outdoor Mobile Robots Based on IDDQN

JIANG SHUHAI, (Member, IEEE), SUN SHANGJIE, AND LI CUN
School of Mechanical and Electronic Engineering, Nanjing Forestry University, Nanjing, Jiangsu 210037, China
Institute of Intelligent Control and Robotics (IICR), Nanjing Forestry University, Nanjing, Jiangsu 210037, China

Corresponding author: Jiang Shuhai (shuhaijiang@aliyun.com)

This work was supported in part by the National Special Research Fund for Non-Profit Sector under Grant 201404402-03, and in part by the 2023 Jiangsu Province Postgraduate Research and Innovation Program under Grant KYCX23-1140.

ABSTRACT Path planning is one of the research hotspots for outdoor mobile robots. This paper addresses the issues of slow convergence and low accuracy in the Double Deep Q Network (DDQN) method in environments with many obstacles in the context of deep reinforcement learning. A new algorithm, Improve Double Deep Q Network (IDDQN), is proposed, which utilizes second-order temporal difference methods and a binary tree data structure to improve the DDQN method. The improved method evaluates the actions of the current robot using second-order temporal difference methods and employs a binary tree structure to store the results obtained from these methods, replacing the traditional experience pool structure. The environment is constructed using a grid method, programmed in the Python language, with two two-dimensional grid maps created for simple and complex environments. DDQN and four related deep reinforcement learning methods, such as Multi-step updates and Experience Classification Double Deep Q Network (ECMS-DDQN), are compared through simulation experiments with the IDDQN method. Simulation results indicate that the IDDQN method improves various path planning metrics compared to the DDQN method and other relevant reinforcement learning methods. In the simple environment, IDDQN method exhibits a 26.89% improvement in step convergence time, a 22.58% improvement in reward convergence time, and a 10.30% improvement in average reward value after convergence compared to the original DDQN algorithm. It also outperforms other simulated methods in the simple environment, although the difference is not significant. In the complex environment, the IDDQN method avoids falling into local optima compared to other methods, demonstrating the accuracy of its strategy in complex environments. Other methods show artificially high average reward values after converging in local optima, lacking reference value. In the complex environment, IDDQN method exhibits a 33.22% improvement in step convergence time and a 25.47% improvement in reward convergence time compared to the original DDQN algorithm, clearly surpassing other participating simulated methods. The data above indicate that the IDDQN method improves both convergence speed and accuracy compared to the DDQN method and the relevant improvement methods simulated in this paper. Particularly in environments with many obstacles, the performance improvement is evident, allowing for effective path planning in such environments.

INDEX TERMS Outdoor mobile robot, path planning, reinforcement learning, DDQN, IDDQN.

I. INTRODUCTION

Mobile robots integrate environment perception, path planning, control decision-making, and execution of actions [1]. Environment perception and path planning are crucial scientific issues in the localization and navigation of mobile robots. Dynamic path planning for mobile robots is a hot research topic, and overcoming various dynamic obstacles remains a worthy area of study [2]. To enable robots to navigate efficiently and reliably in an environment without any human

The associate editor coordinating the review of this manuscript and approving it for publication was Alba Amato.

assistance, we still face many challenges [3].Path planning refers to the robot autonomously searching for a collision-free path from the starting point to the destination based on the sensors it carries [4]. Robots should be able to extract necessary information from the environment and take necessary actions to plan a feasible collision-free motion path to achieve their goals [5]. To successfully implement path planning, mobile robots need to possess perception, detection, and decision-making capabilities [6].Mobile robots should be capable of completing tasks from the starting point to the destination in the safest, shortest path, or in the shortest time, without human intervention [7]. Path planning and obstacle avoidance methods can be classified based on the environment into static path planning or dynamic path planning, and based on the path planning algorithm, into global path planning and local path planning [8].

Research on the path planning algorithms for mobile robots is extensive, encompassing both global and local path planning. The methods can be broadly categorized into traditional approaches and reinforcement learning-based path planning methods [9]. Representative traditional methods include the A$*$ algorithm [10], simulated annealing algorithm [11], artificial potential field method [12], particle swarm algorithm [13], and ant colony algorithm [14]. Q-learning is a commonly used reinforcement learning-based path planning method [15], and many scholars globally have made improvements and innovations to the Q-learning algorithm to further enhance the efficiency of robot path planning. Soong et al. [16] addressed the dimensionality catastrophe issue during Q-learning training in complex environments by proposing Deep Q-learning Network (DQN). DQN bridges the gap between high-dimensional perceptual inputs and actions using a neural network to store content. While this algorithm successfully addresses the dimensionality catastrophe problem of traditional Q-learning in complex environments, it still faces the issue of overestimation. Hasselt et al. [17] addressed the overestimation problem in the DQN algorithm by proposing the Double Deep Q-network (DDQN) algorithm. In comparison to the DQN algorithm, DDQN introduces a second Q-network layer, with the output of the first layer no longer used for the final output but rather as input for the second layer. Simulations indicate that the training effectiveness of the double-layered DQN network surpasses that of DQN, resolving the overestimation problem and improving accuracy. However, DDQN, using a relatively simple experience pool, exhibits low sampling efficiency in complex environments, impacting the algorithm's learning speed. Additionally, action selection in complex environments after convergence is not highly accurate, and the achieved reward values are not high.

Currently, there is not much research on the improvement of the DDQN algorithm. This paper focuses on addressing the issues of slow convergence speed and low accuracy of the DDQN algorithm in environments with many obstacles. A novel improvement to the Double Deep Q Network (DDQN) algorithm, named Improved Double Deep Q Network (IDDQN), is proposed based on a double-layered network, introducing a second-order temporal difference method [18] and a binary tree structure [19]. The IDDQN algorithm utilizes the second-order temporal difference method to evaluate the effectiveness of the current iteration results. The effect of the actions chosen in the current iteration is inversely proportional to the results calculated by the second-order temporal difference method. Based on the evaluation results, more suitable actions can be selected. The results calculated by the second-order temporal difference method are stored as historical information. To reduce the time of selecting historical information, a binary tree structure is employed to replace the experience pool structure for storing historical information. To demonstrate the effectiveness of the algorithm, three simulation maps with different obstacle positions are established [20], [21]. Simulations indicate that the IDDQN algorithm, compared to the DDQN algorithm and related improvement algorithms, achieves improved convergence speed and accuracy. It can effectively facilitate path planning for mobile robots in complex environments. Fig. 1 illustrates the flowchart of the algorithm.

In summary, the main contributions of this paper are as follows:

1. Improved the existing DDQN algorithm to address environments with numerous obstacles. Simulation results indicate that the training metrics and efficiency of path planning with this algorithm surpass traditional methods.
2. Introduced the use of a second-order temporal difference method to evaluate the iterations of the DDQN algorithm during training. This allows for a more accurate estimation of action values, enhancing the algorithm's precision.
3. Proposed the use of a binary tree structure to replace the experience pool structure for storing iterative actions in the algorithm. This reduces the time required for action selection, thereby improving training efficiency and accelerating convergence.

The paper consists of six parts. The first part is the introduction, presenting some previous research and introducing the innovative aspects of the proposed methods. The second part reviews related work, discussing the development trends of similar studies. The third part focuses on environmental interaction modeling. The fourth part presents the methodology. The fifth part includes experimental analysis. The sixth and final part presents the conclusion.

II. PREVIOUS WORK

This chapter summarizes similar work in recent years into two categories: DQN-based methods and DDQN-based methods, and analyzes the research motivations, original ideas, and limitations of other researchers. The results are summarized in Tab. 1.

A. ALGORITHM BASED ON DQN

Schaul et al. [22] addressed the issue of poor learning performance in DQN by proposing a framework that prioritizes experience and implementing prioritized experience replay in

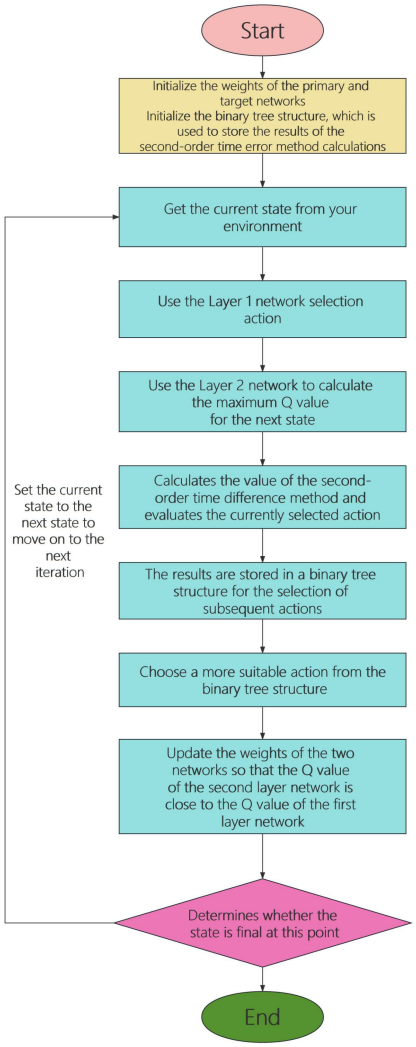

FIGURE 1. Schematic diagram of the IDDQN process.

DQN. This method, compared to uniform replay in DQN, exhibits superior performance with significantly increased reward values, but there is still room for improvement in the algorithm's convergence speed.

Cheng and Meng [23] tackled the problem of non-optimal paths in DQN by linearly combining the action-value function output by the neural network and the environment's intrinsic value to improve the original DQN network (LCV-DQN). This refinement results in more accurate and efficient path planning, but the overestimation problem of DQN persists.

Gu et al. [24]addressed the slow convergence issue in DQN by decomposing the network structure, decoupling action selection and action evaluation, thereby accelerating DQN convergence. However, this approach may not adapt well to complex environments, leading to suboptimal paths in the algorithm's planning.

Saito et al. [25] proposed a DQN algorithm based on a taboo list strategy (TLS-DQN) for indoor single-path environments. While this algorithm can find a path to the target point in simple indoor environments, it lacks the ability to guarantee effective path planning in complex environments due to simulations being limited to environments with fewer obstacles.

Gao et al. [26] addressed the slow convergence issue in the DQN algorithm combined with prioritized experience replay by introducing the k-means algorithm to handle the agent's state space. This improvement enhanced the convergence speed of the DQN algorithm. However, the simulation environment was limited to a simple setting in Tianjin Port, Bohai Bay, and the algorithm exhibited unrealistic path planning in complex environments at other ports.

B. ALGORITHM BASED ON DDQN

Zhang et al. [27]addressed issues in the DQN algorithm's path planning process, such as overestimation and poor algorithm stability. They combined the Double Deep Q Network (DDQN) with the Average Deep Q Network (ADQN), using prioritized experience replay instead of average sampling to enhance the utilization of value samples, reduce overestimation issues, and improve algorithm stability. However, this approach is not suitable for continuous state spaces.

Wang et al. [28] tackled the slow convergence issue in DDQN networks by establishing a DDQN reinforcement learning algorithm based on a greedy strategy. This method demonstrated faster network convergence speed and improved path planning capabilities compared to the DQN algorithm. However, it still employed an experience pool structure, reducing sampling efficiency.

Yang et al. [29] addressed the issue of low accuracy in DDQN paths by combining prior knowledge and integrating the action mask method to handle ineffective actions generated by the agent, improving the DDQN algorithm. This method, compared to traditional methods such as DQN, A∗, and RRT, generated paths with better performance. However, it exhibited an overreliance on prior knowledge, requiring manual intervention for specific tasks.

Chu et al. [30] addressed the non-optimality of DDQN paths in complex environments by proposing a dynamic composite reward function to enhance DDQN, improving the algorithm's path planning capabilities in complex environments. However, it faced issues of poor convergence during the training phase.

Peng et al. [31] combined multi-step update methods with Deep Double Q Networks, proposing the MS-DDQN algorithm, which exhibited better learning efficiency and generalization capabilities. However, it might become trapped in local optima when the state space is extensive or contains highly complex structures.

Zhang et al. [32], building upon MS-DDQN, combined the advantages of multi-step guidance DDQN (MS-DDQN) and experience categorization DDQN (EC-DDQN) algorithms, developing a novel Experience Categorization Multi-Step DDQN (ECMS-DDQN) algorithm. This improved the algorithm's generalization capabilities in total returns and path planning. However, the reliance on prior knowledge

in experience categorization made the algorithm sensitive to environmental changes, limiting its universality. Moreover, the experimental setup featured overly simple random environments with few obstacles, potentially leading to local optima, and the algorithm parameters need further optimization.

III. ENVIRONMENTAL INTERACTION MODELING

Robots are learners or decision-makers, and the environment is everything with which the robot interacts, excluding the robot itself. The process of predicting the robot's intentions can be seen as a continuous interaction between the robot and the environment. Each interaction corresponds to a reward value, and the best action can be chosen based on the iterative results of these reward values. Fig. 2 illustrates the interaction between the robot and the environment. In this figure, $\boldsymbol{S_t}$ represents the robot's position in the $\boldsymbol{t}$ -th iteration, $\boldsymbol{r_t}$ is the reward function value in the $\boldsymbol{t}$ -th iteration, and $\boldsymbol{A_t}$ is the action taken by the robot in the $\boldsymbol{t}$ -th iteration.

From Fig. 2, it can be observed that when the robot performs action A_t, it interacts with the current environment, resulting in a current position S_t and a reward value r_t. This reward value reflects the score of the current action. Based on the current position S_t and the reward value r_t obtained from the interaction with the environment, the robot can choose the next action A_{t+1}, which will be executed in the next iteration. In the continuous iterative cycle, the robot continuously updates its position and reward values based on environmental information. After iterating a sufficient number of times, the robot will obtain an optimal strategy.

This paper defines four actions for the grid simulation environment, namely up, down, left, and right. In reinforcement learning, the robot receives a special reward signal during its interaction with the environment. For each step, the reward is a numerical value, and the robot's goal is to maximize its reward. The environment provides positive rewards when the robot performs well and negative rewards (punishments) when the robot performs poorly.

The following reward rules are defined:

(1) Rewards include ''reward'' and ''penalty,'' where rewards are defined as positive and penalties as negative.

(2) The reward values range from -1.0 to 1.0.

(3) Moving from one cell to an empty cell earns $+0.25$ points.

(4) The robot incurs a penalty of -0.05 points for each movement to ensure finding the shortest path.

(5) To avoid collisions with obstacles, moving to an obstacle cell results in a penalty of -0.75 points.

(6) If the robot moves beyond the boundaries of the grid map, a penalty of -0.75 points is imposed.

(7) The robot receives a penalty of -0.25 points for any action taken in cells it has previously visited.

(8) Arriving at the destination earns a reward of $+0.75$ points.

TABLE 1. Previous work.

Algorithm	Number of neural networks	Characteristics	Limitations
Prioritized Experience Replay-based DQN[22] (Tom et al.,2015)	1	significant increase in rewards	slow convergence speed
LCV-DQN[23] (Cheng et al.,2022)	1	path is optimized	overestimation issue
Decoupled action selection and action evaluation with improved DQN[24] (Gu et al.,2022)	1	accelerated convergence	unable to adapt to complex environments
TLS-DQN[25] (Nobuki et al.,2022)	1	optimized performance in indoor environments	cannot guarantee practicality in dynamic and complex outdoor environments
Improved DQN using k-means algorithm[26] (Gao et al.,2023)	1	applicable to simple harbor environments	not tested in complex harbor environments
Combine DDQN with ADQN[27] (Zhang et al.,2022)	2	improves the utilization rate of value samples and the stability, reduces the problem of overestimation	not suitable for contiguous spaces
DDQN based on greedy strategy[28] (Wang et al.,2022)	2	improved the convergence speed of the DDQN algorithm	still using an experience replay buffer structure, which reduces sampling efficiency
DDQN based on prior knowledge, integrated action mask method[29] (Yang et al.,2022)	2	increased path accuracy	excessive reliance on prior knowledge, some tasks require manual intervention
DDQN based on Dynamic Compound Reward Function[30] (Chu et al.,2022)	2	improved the path planning capability of the algorithm in complex environments	the experience replay buffer structure reduces sampling efficiency
MS-DDQN[31] (Peng et al.,2021)	2	improved learning efficiency and generalization capability	cannot adaptable to complex environments
ECMS-DDQN[32] (Zhang et al.,2022)	2	improved returns and generalization capability	reliance on prior knowledge, sensitivity to environmental changes.

(9) To prevent infinite loops, set a minimum total reward ($-0.5 * $ environment size). When the total reward falls below this minimum, stop the episode and move to the next one.

Fig. 3 is an example of an action and reward function. The green grid represents the robot, the blue grid represents the end point that the robot wants to reach, the red grid represents the untouchable grid that is the obstacle, and the gray grid represents the grid that the agent can enter. As can be seen in Fig. 3, the reward for each lawful action is $+0.25$ points, and -0.05 points are required for each step, e.g. from step 1 to step 2, from step 2 to step 3, or from step 4 to step 5. When a robot moves to a visited unit, it will receive a penalty of -0.25 points. For the final action, the robot reaches its destination, so the reward is 0.75 points.

IV. METHODOLOGY

Deep reinforcement learning algorithms have prominent advantages in solving intent prediction problems compared to other algorithms. This paper proposes an Improved Double

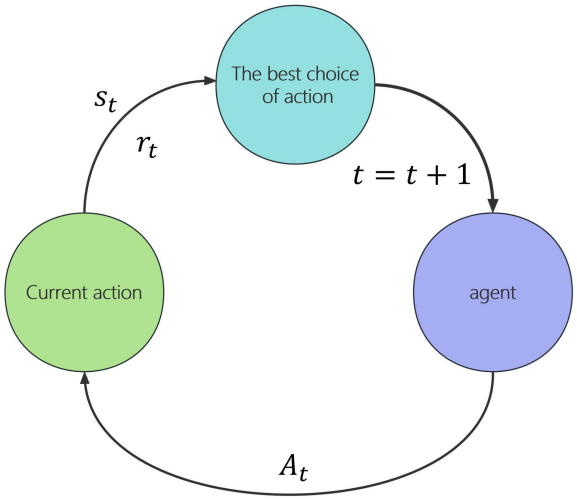

FIGURE 2. Diagram of the interaction between the robot and the environment.

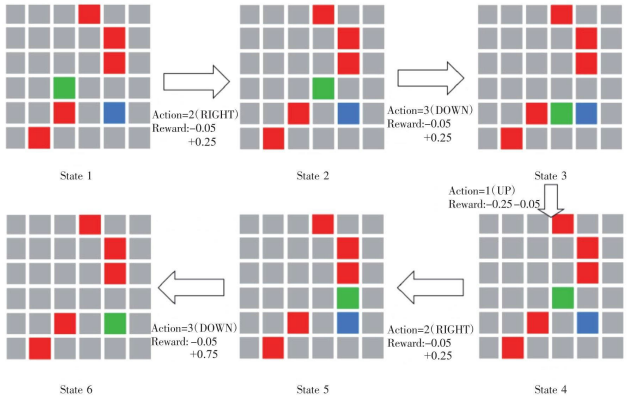

FIGURE 3. Schematic diagram of robot rewards and actions.

Deep Q Network (IDDQN) algorithm based on the Double Deep Q Network (DDQN) with a dual-layer network. The IDDQN algorithm introduces a second-order temporal difference method and a binary tree structure to enhance performance. The second-order temporal difference method evaluates the effectiveness of the current iteration's results, where the impact of the chosen action in the current iteration is inversely proportional to the results calculated by the second-order temporal difference method. The results of the second-order temporal difference method are stored as historical information. To reduce the time required for selecting historical information, a binary tree structure is employed to replace the experience pool structure for storing historical information. Integrated with the robot-environment interaction model discussed in Section Two, this improved algorithm can be applied to path planning for robots in environments with numerous obstacles. The algorithmic model is illustrated in Fig. 4.

With this improvement, the algorithm can perform calculations more quickly and efficiently handle the interaction between the robot and the environment. As a result, it can more accurately predict the next actions and strategies the robot will take in different environments. This contributes to the robot making more precise decisions in complex environments, enhancing both the practicality of the robot in real-world scenarios and the training efficiency of the algorithm. The specific description of the model structure in Fig. 4 is provided below.

A. SECOND-ORDER TEMPORAL DIFFERENCE

In this paper, the traditional DDQN algorithm is enhanced using the second-order temporal difference method and a binary tree structure, addressing issues of low accuracy and slow convergence in the traditional DDQN. The second-order temporal difference method is introduced to evaluate the current actions of the robot, and this evaluation is compared with the next iteration. When the second-order temporal difference is significant, a reselection of actions is warranted. In any m-th iteration calculation, at any time t, given the current state $(s_t, a_t,)$, DDQN uses the estimation discrepancy from the previous round for $(s_t, a_t,)$ as the temporal difference, as shown in formula (1):

$$\mathbf{TDE}_m(s_t, a_t) = \beta_t[r_{t+1}(s_t, a_t) + \gamma max\, Q_{m-1}(s_{t+1}, a_{t+1}) - Q_{m-1}(s_t, a_t)] \quad (1)$$

In the equation, $\mathbf{TDE}_m(s_t, a_t)$ represents the temporal difference for the state-action pair at time t in the m-th round, β_t is the learning rate, $r_{t+1}(s_t, a_t)$ is the immediate reward value for that state-action pair, γ is the discount factor, and $Q_{m-1}(s_{t+1}, a_{t+1})$ denotes the estimated Q-value for taking action a_t in state s_t in the $(m-1)$-th round. In equation (2), the estimation of the current state-action value is based on the maximum value of the estimations for subsequent state-action pairs, specifically utilizing the extreme values of Q for subsequent state-action pairs instead of the true extremes for the state-action pair, thus calculating the temporal difference.

Definition of Second-Order Temporal Difference:

In the m-th round, at any time t the immediate reward value is denoted as r_{t+1}, and the Q-value is represented as:

$$Q_m(s_t, a_t) = r_{t+1} + \gamma max Q_m(s_{t+1}, a_{t+1}) \quad (2)$$

where $Q_m(s_{t+1}, a_{t+1})$ represents the estimation for the state-action pair at time $t+1$ in the m-th round. The definition of the second-order temporal difference is illustrated in Formula (3), denoted as $\mathbf{TDE}^2$.

$$\mathbf{TDE}^2_m(s_t, a_t) = \alpha_m [Q_{m-2}(s_t, a_t) - Q_{m-1}(s_t, a_t)] + (1-\alpha_m)[Q_{m-1}(s_t, a_t) - Q_{m-2}(s_t, a_t)] \quad (3)$$

Here, $\mathbf{TDE}^2_m(s_t, a_t)$ represents the second-order temporal difference for the action pair at time t in the m-th round. $Q_{m-2}(s_t, a_t)$ denotes the estimated Q-value for taking action a_t in state s_t in the $(m-2)$-th round. $Q_{m-1}(s_t, a_t)$ represents the estimated Q-value for taking action a_t in state s_t in

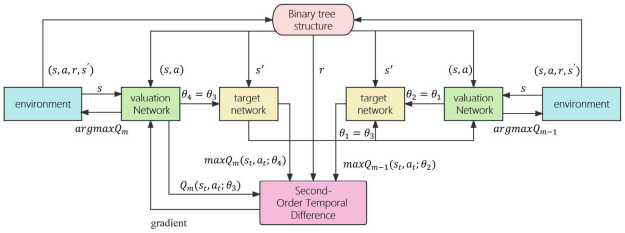

FIGURE 4. The DDQN model structure based on second-order temporal difference and binary tree structure.

the $(m-1)$-th round. α_m is a parameter between 0 and 1 used to differentiate the differences between the weighted first two estimation functions. $(1-\alpha_m)$ is the complement of α_m, used to distinguish the differences between the weighted last two estimation functions. The meaning of the formula is to obtain a comprehensive second-order temporal difference error by taking a weighted average of the differences between the first two estimation functions and then taking a weighted average of the differences between the last two estimation functions.

B. LOSS FUNCTION CONSTRUCTION

The larger the difference between the actual output and the expected output of the algorithm, i.e., the larger the value of the loss function, the worse the convergence effect of the algorithm. In response to the stability issue of algorithm convergence, this paper introduces the concept of second-order temporal difference to reconstruct the loss function, aiming to improve the convergence of the algorithm.

In the learning process of the DDQN model, sample data is randomly selected from the experience pool, and parameter updates are performed through gradient descent. In the DDQN structure with second-order temporal difference, random sampling is conducted through experience replay, learning is performed through the second-order temporal difference function, and the weights θ are updated. The formula for calculating the second-order temporal difference is as shown in Formula (4):

$$\begin{aligned}\mathbf{TDE}_m^2 = {} & \alpha\left[Q_{m-2}(s_t,a_t;\theta_2)-Q_{m-1}(s_t,a_t;\theta_3)\right] \\ & +(1-\alpha)\left[Q_{m-1}(s_t,a_t;\theta_4)-Q_{m-2}(s_t,a_t;\theta_2)\right]\end{aligned} \tag{4}$$

where $\mathbf{TDE}_m^2$ represents the second-order temporal difference of the action pair at time t in the m-th round. $Q_{m-2}(s_t,a_t;\theta_2)$ represents the estimated Q value of the value function at round $(m-2)$ when taking action a_t in state s_t, using the weight parameters θ_2. $Q_{m-1}(s_t,a_t;\theta_3)$ represents the estimated Q value of the value function at round $(m-1)$ when taking action a_t in state s_t, using the weight parameters θ_3. $Q_{m-1}(s_t,a_t;\theta_4)$ represents the estimated Q value of the value function at round $(m-1)$ when taking action a_t in state s_t, using the weight parameters θ_4. α is a parameter between 0 and 1, used to distinguish the difference between the two estimated value functions before weighting. $(1-\alpha)$ is the complement of α, used to distinguish the difference between the two estimated value functions after weighting. The meaning of the formula is to perform a weighted average of the differences between the first two estimated value functions, and then perform a weighted average of the differences between the latter two estimated value functions, obtaining a comprehensive second-order temporal difference error. The α weight parameter can be adjusted to choose the relative importance of the two estimated value functions. θ_2 is the parameter of DDQN_1 (DDQN first-layer target network), θ_3 and θ_4 are the weight parameters of the value network and target network of DDQN_2 (DDQN second-layer network), respectively. In each step of model training, the parameter update process is as follows: pass the parameters of the value network in DDQN_2 to the value network in DDQN_1, while keeping itself updated, i.e., $\theta_3 = \theta_1$, $\theta_3 = \theta_3$, where θ_3 represents the weight parameters of the value network in DDQN_2 for the next state, and every N steps, pass the parameters of the value network in DDQN_1 to the target network, i.e., $\theta_2 = \theta_1$, and at the same time pass the parameters of the value network in DDQN_2 to the target network, i.e., $\theta_4 = \theta_3$. Thus, the improved loss function is given by Formula (5):

$$\begin{aligned}L(\theta) = {} & E_{s,a,r,s'}[(1-\alpha)(TQ_2 - TQ_1) \\ & -\alpha(T\,Q_1 - BQ_2)]^2\end{aligned} \tag{5}$$

This formula calculates the loss function $L(\theta)$, where θ represents the parameters of the model, $E_{s,a,r,s'}$ represents the expectation over all possible states s, actions a, rewards r, and next states s'. TQ_2 represents the Q value of the second estimated function calculated using the target network. TQ_1 represents the Q value of the first estimated function calculated using the target network. BQ_2 represents the Q value of the second estimated function calculated using the current network. α is a parameter between 0 and 1, used to weight the two differences. The loss function $L(\theta)$ describes the difference between the two estimated value functions, and it computes the variance of the parameter values for the two networks. Solving the reciprocal of the loss function in Equation (5), the weight gradient is obtained as shown in Equation (6):

$$\begin{aligned}\nabla_\theta L(\theta) = {} & E_{s,a,r,s},\{[(1-\alpha)(TQ_2 - TQ_1) \\ & -\alpha(TQ_1 - BQ_2)]\nabla_\theta Q(s,a;\theta_3)\}\end{aligned} \tag{6}$$

The weight gradient is used to update the parameters of the value network with the aim of keeping the current network's weight parameters fixed. This helps the value network better approximate the true values and prevents convergence instability in the algorithm.

C. BINARY TREE STRUCTURE

In addition, this paper replaces the traditional experience pool structure with a binary tree structure to store the results obtained from the aforementioned second-order time difference method. This reduces the time required to retrieve

historical information. To simplify the binary tree formula, let the second-order time difference $\mathbf{TDE}_{\boldsymbol{m}}^{\mathbf{2}}$, calculated using the m -th round of the estimation function and the $(m-1)$-th round of the estimation function at time step t, be denoted as $\boldsymbol{d_t}$, i.e.,:

$$d_t = \mathrm{TDE}_m^2 \tag{7}$$

where the nodes in the binary tree structure are proportional to the results of the time difference $\boldsymbol{d_t}$. The schematic diagram of the binary tree storing the time difference is shown in Fig. 5.

Combining the temporal difference method and the binary tree structure with the traditional DDQN method efficiently selects historical information with larger errors, thereby improving the accuracy and convergence speed of the traditional DDQN algorithm. In this context, $\boldsymbol{p_t}$ represents the value of each binary tree node, and this value is determined based on the magnitude of $\boldsymbol{d_t}$. Its calculation is as shown in the following formula:

$$p_t = |d_t| + \mu \tag{8}$$

In the above equation, μ is a small constant used to avoid $\boldsymbol{p_t}$ being an integer due to $\boldsymbol{d_t}$ being an integer. $t \in 1, 2 \ldots, j$, where j represents the capacity of the binary tree.

In the previously mentioned method IDDQN, which combines the temporal difference and binary tree methods with the traditional DDQN approach, the principle of prioritized sampling is that the larger the value of leaf nodes in the binary tree, the higher their priority and therefore the higher the probability of being selected. Through the IDDQN method, time difference results d_t can be selected quickly and efficiently, reducing the algorithm's runtime and improving its efficiency. The prioritized sampling $\boldsymbol{P_t(t)}$ can be calculated using the following expression:

$$\mathrm{P}_t(t) = \frac{p_t}{\sum_{t=1}^{j} p_t} \tag{9}$$

Prioritized sampling can lead to an uneven distribution of algorithm results, potentially introducing bias. To reduce the risk of premature convergence and improve the stability of the sampling process, an importance sampling weight method is introduced. The goal of this method is to estimate certain distribution properties and reduce variance by adjusting the probability distribution. The calculation formula is as follows:

$$\omega_t(t) = \frac{1}{j \cdot \mathrm{P}_t(t)} \tag{10}$$

where $\boldsymbol{\omega_t}$ represents the importance sampling weight, t is the current time, and j is the priority at the time of sampling. This method helps improve the stability of sampling and reduces bias.

Next, the neural network parameters $\hat{\theta}_t$ are updated through backpropagation with the following update expression:

$$\hat{\theta}_t = \theta_t^{l2} + \omega_t(t) \cdot (d_t)^2 \tag{11}$$

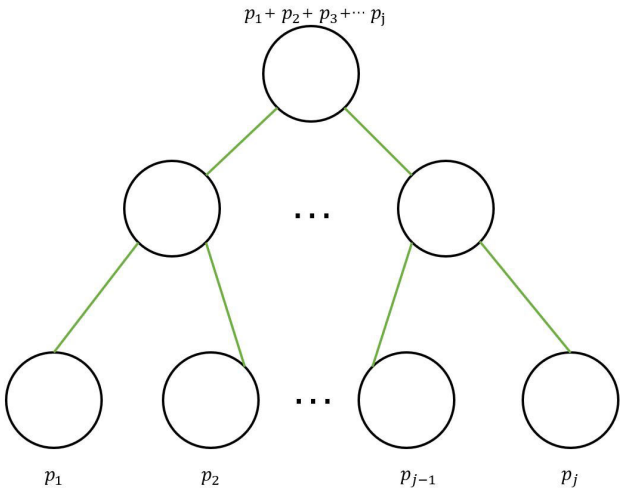

FIGURE 5. Binary tree structure diagram.

In the above equation, $\hat{\theta}_t$ represents the updated Q-network parameters after t iterations, $\boldsymbol{\theta_t^{l2}}$ is the current parameters, $\boldsymbol{\omega_t}$ is the importance sampling weight, and $\boldsymbol{d_t}$ is the temporal difference error result. This step is used to adjust the neural network parameters to improve the algorithm's performance.

After obtaining the neural network parameters, you can use the approximation of the action-value function to calculate the policy. If the next iteration is not in the terminal state, you can determine the action for the next iteration based on Equation (12). The algorithm will continue to execute in a loop until it reaches the final state.

The flowchart of the algorithm is shown in Fig. 5.

$$\pi_{t+1} = ((1-\varepsilon) \cdot \pi_{t+1,best} \wedge \varepsilon \cdot \pi_{t+1,other}) \tag{12}$$

V. EXPERIMENTS AND ANALYSIS

A simple and effective grid map can be used for path planning simulation research for robots other than drones, enabling simulation of complex environments and reducing modeling complexity. At the same time, the robot's starting point, endpoint, obstacles, and path can all be displayed in a 2D grid map, facilitating the visualization of path planning information. It is convenient to use 2D grid maps to simulate information about real environments, as shown in the first map in Fig. 6, which represents a complex 25∗25 grid environment, and the second map represents an extremely complex 30∗30 grid environment. Blue squares represent obstacles, red dots represent the robot, the endpoint is set as a yellow square, and gray positions represent safe areas. Therefore, in the research on path planning for mobile robots in this paper, the grid method is used for environmental simulation.

In the first map of Fig. 6, the obstacle ratio is 8%, indicating a lower number of obstacles, defined as a simple environment. In the second map, the obstacle ratio is 25%, indicating a higher number of obstacles, defined as a complex environment. These two maps will be used

for the simulation of DQN algorithm, DDQN algorithm, TLS-DQN algorithm, MS-DDQN algorithm, ECMS-DDQN, and IDDQN algorithm. They will be used to test the convergence speed, total average reward values, and path conditions of these algorithms. This paper proposes an improvement based on the DDQN method. The DDQN algorithm is currently a mature deep reinforcement learning algorithm, and thus, the simulation results of the DDQN algorithm are used as the baseline for performance evaluation. Metrics such as convergence speed in steps, convergence speed in reward values, average reward values after convergence, and whether it leads to the optimal path are used to measure the algorithm's performance. This is done to demonstrate the advantages of the improved IDDQN algorithm over the DDQN algorithm, as well as its strengths and weaknesses compared to other algorithms in the same category.

A. SIMULATION OF SIMPLE ENVIRONMENT

The order in the Fig. 7, Fig. 8, Fig. 9 is as follows: (a) DQN, (b) DDQN, (c) TLS-DQN, (d) MS-DDQN, (e) ECMS-DDQN, (f) IDDQN.

Fig. 7 shows the optimal paths simulated by the six algorithms, all with a step count of 42. Therefore, these paths can all be considered as optimal paths for the given task in a simple environment.

Fig. 8 illustrates the change in step length with the number of iterations, and Tab. 2 provides a performance comparison of the step length convergence for the six algorithms based on Fig. 8 in a simple environment. The data in the table are the averages of 10 experiments. Using the convergence performance of the DDQN algorithm as a baseline, the comparison of each algorithm with DDQN in terms of step length convergence speed is shown. DQN exhibits less convergence performance than DDQN in a simple environment. MS-DDQN improves the convergence performance by 16.01% compared to the original DDQN. The ECMS-DDQN, based on the improvement of MS-DDQN, outperforms the original DDQN with a convergence performance improvement of 26.41%. IDDQN shows a 26.89% improvement in convergence performance compared to the original DDQN. It can be observed that in a simple environment, the proposed IDDQN algorithm and the ECMS-DDQN algorithm have similar performance, indicating that there is no significant advantage of IDDQN over ECMS-DDQN in a simple environment. However, they both demonstrate significant advantages over the original DDQN algorithm, DQN algorithm, TLS-DQN algorithm, and MS-DDQN algorithm.

Fig. 9 reflects the change in reward values over iterations, and Tab. 3 compares the convergence performance and average reward values after convergence for the six algorithms in a simple environment based on the data from Fig. 9. The data in the table is averaged over 10 experiments, with the convergence performance of the DDQN algorithm taken as the baseline. Except for the original DQN, other algorithms show improvements in both the speed of reward convergence and the average reward values after convergence compared to DDQN.The proposed algorithm shows the most significant improvement, with a 22.58% increase in reward convergence speed compared to DDQN. Moreover, after over 1500 iterations, the reward values avoid negative values, demonstrating more accurate policy choices at each step, minimizing backtracking, and collisions with obstacles. However, the difference with the ECMS-DDQN algorithm is not substantial.

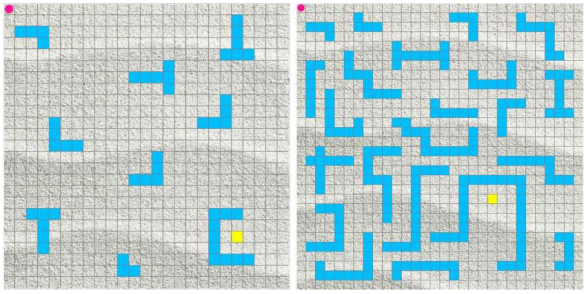

FIGURE 6. Simulation environment.

In the simple environment, none of the six algorithms fall into local optima, all finding an optimal path. Therefore, the average reward values after convergence are valuable as a reference. Except for the original DQN algorithm, the average reward values of the other algorithms are higher than the DDQN algorithm. The proposed IDDQN algorithm achieves the highest average reward value, with a 10.30% improvement over the original DDQN algorithm, demonstrating the accuracy of the path strategy proposed in this study.

In summary, the DQN, DDQN, TLS-DDQN, MS-DDQN, ECMS-DDQN, and IDDQN algorithms all successfully complete path planning in a simple environment, finding an optimal path. IDDQN exhibits better performance in terms of convergence speed, with a 26.89% improvement over DDQN. The proposed algorithm shows no significant advantage over ECMS-DDQN in a simple environment but has a clear advantage over other algorithms. IDDQN excels in both reward convergence speed and average reward values after convergence, with a 22.58% and 10.30% improvement, respectively, compared to the original DDQN algorithm. In a simple environment, IDDQN demonstrates a clear advantage over the original DDQN algorithm and some previous algorithms. However, it does not show a significant difference compared to ECMS-DDQN, warranting further comparison in a complex environment.

B. SIMULATION OF COMPLEX ENVIRONMENT

The order in the Fig. 10, Fig. 11, Fig. 12 is as follows: (a) DQN, (b) DDQN, (c) TLS-DQN, (d) MS-DDQN, (e) ECMS-DDQN, (f) IDDQN.

Fig. 10 depicts the optimal paths in a simulation of six algorithms in a complex environment. The optimal path for the DQN algorithm is 60 steps, while the optimal paths for the DDQN and TLS-DQN algorithms are 58 steps. The optimal paths for the MS-DDQN and ECMS-DDQN algorithms are

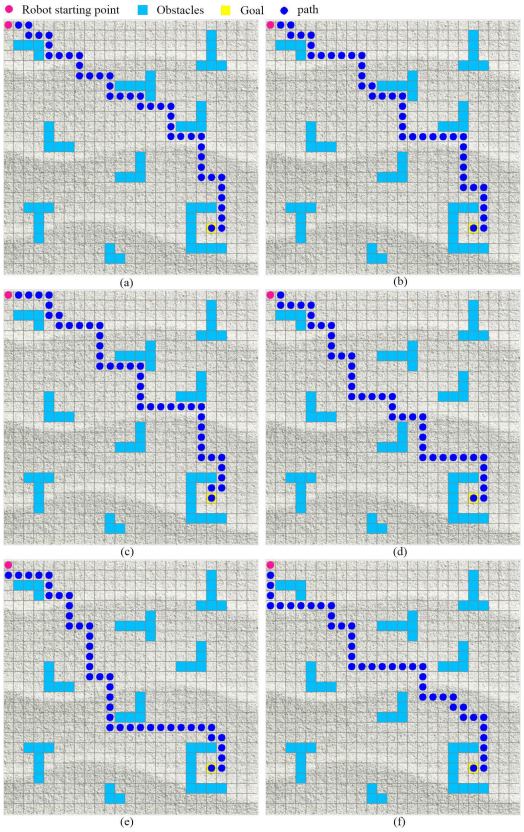

FIGURE 7. Optimal trajectories of algorithms in a simple environment.

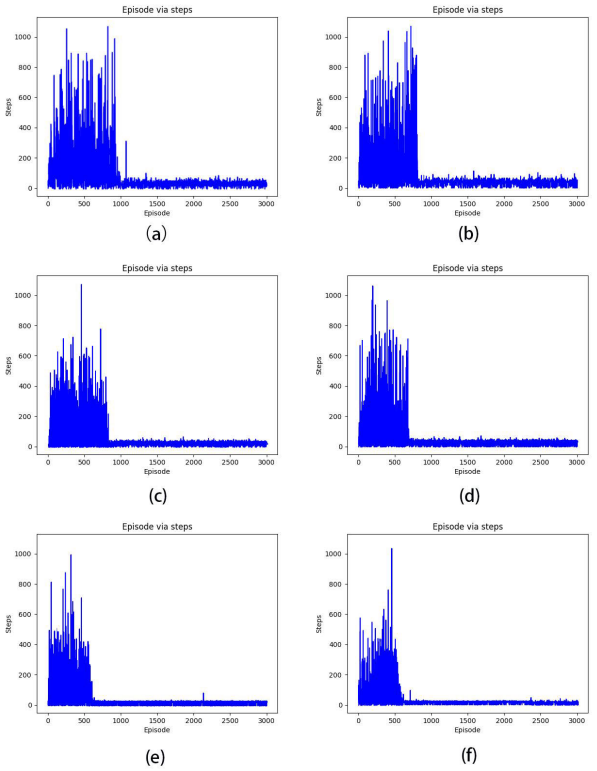

FIGURE 8. The graph depicts the variation of step length with the number of iterations for algorithms in a simple environment.

TABLE 2. The statistics of required episodes for step length convergence of algorithms in a simple environment are as follows.

Algorithm	Required episode for step convergence	Percentage (baseline:DDQN)
DQN	983	-20.17%
DDQN	818	0%
TLS-DQN	821	0.37%
MS-DDQN	687	16.01%
ECMS-DDQN	602	26.41%
IDDQN	598	26.89%

56 steps, and the optimal path for the IDDQN algorithm is 52 steps. Upon observing the paths, it is evident that, except for the IDDQN algorithm, the other algorithms exhibit instances of backtracking to avoid obstacles. In contrast, the IDDQN algorithm does not encounter this situation and finds the shortest path. After multiple experiments, it has been demonstrated that in environments with numerous obstacles, algorithms like DQN and DDQN may experience local optima without fully exploring the entire map. The IDDQN algorithm, which stems from the optimization of action exploration strategies, avoids getting stuck in local optima.

Fig. 11 illustrates the variation of step size with the number of iterations. Tab. 4 presents a performance comparison of the convergence characteristics of six algorithms in a complex environment based on Fig. 11. The data in the table represent the average of 10 experiments, with the convergence performance of the DDQN algorithm used as the baseline. The comparison highlights the convergence speed of each algorithm relative to DDQN. It is evident that DQN and TLS-DQN algorithms exhibit significantly poorer convergence performance in complex environments compared to DDQN. MS-DDQN shows a 9.29% improvement in convergence performance over the original DDQN, while ECMS-DDQN, an improvement based on MS-DDQN, demonstrates a 19.16% improvement in convergence performance over the original DDQN. IDDQN, compared to the original DDQN, exhibits a remarkable 33.22% improvement in convergence performance. It can be observed that in complex environments, the step convergence performance of the IDDQN algorithm proposed in this paper surpasses the other five algorithms. This demonstrates a clear advantage of the IDDQN algorithm in complex environments.

Fig. 12 illustrates the variation of reward values with the number of iterations. Tab. 5 presents a performance comparison of the convergence characteristics and average reward values after convergence for six algorithms in a complex environment based on Fig. 12. The data in the table represent the average of 10 experiments, with the convergence performance of the DDQN algorithm used as the baseline. Except for the original DQN, all other algorithms show improvements in reward value convergence speed relative to DDQN.

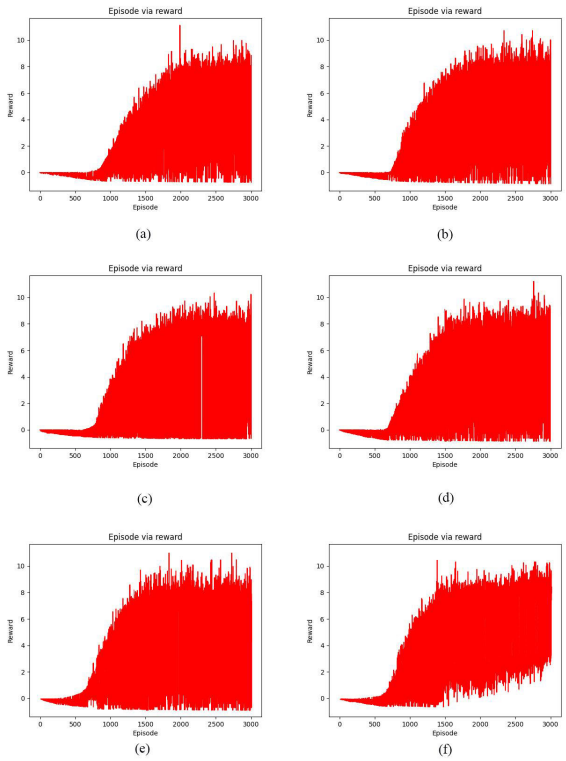

FIGURE 9. The graph illustrates the variation of reward values with the number of iterations in a simple environment.

TABLE 3. Statistics on the number of episodes required for reward convergence and the average reward value after convergence for algorithms in a simple environment are shown below.

Algorithm	Required episode for reward convergence	Percentage (baseline: DDQN)	The average reward value after convergence	Percentage (baseline: DDQN)
DQN	2036	-9.99%	8.05	-2.42%
DDQN	1851	0%	8.25	0%
TLS-DQN	1814	2.00%	8.35	1.21%
MS-DDQN	1631	11.89%	8.65	4.85%
ECMS-DDQN	1498	19.07%	8.90	7.88%
IDDQN	1433	22.58%	9.10	10.30%

The algorithm proposed in this paper exhibits the most significant improvement, with a 25.47% increase in reward value convergence speed compared to DDQN. Moreover, after over 2200 iterations, the reward values avoid negative values, demonstrating more accurate policy selection with fewer instances of backtracking and collision with obstacles.

In the complex environment constructed in this study, all algorithms, except IDDQN, fall into local optima, resulting in artificially high average reward values after convergence, rendering them of little reference value. Only the IDDQN algorithm avoids local optima, providing further evidence of the accuracy of its strategy in a complex environment.

In summary, DQN, DDQN, TLS-DDQN, MS-DDQN, and ECMS-DDQN algorithms all fall into local optima in a complex environment. IDDQN demonstrates superior performance in step convergence, with a 33.22% improvement over DDQN, establishing a clear advantage for the proposed algorithm in complex environments. IDDQN also outperforms other algorithms by showing a 25.47% improvement in reward value convergence speed compared to the original DDQN algorithm. In a complex environment, IDDQN stands out among the algorithms, as it avoids local optima, highlighting the accuracy of its strategy.

C. ABLATION EXPERIMENTS

This paper introduces two main improvements to DDQN:

1. The proposal to use a binary tree structure instead of the experience pool structure for storing algorithm iterations. This modification, referred to as Module A, reduces the time taken for action selection, enhancing training efficiency, and expediting convergence.

2.The introduction of a second-order temporal difference method to evaluate the results of DDQN algorithm iterations. This approach, defined as Module B, provides a more accurate estimation of action values, thereby improving algorithm accuracy.

In this section, we conduct ablation experiments for these two modules in a complex environment. Initially, we simulate DDQN using only a binary tree to replace the experience pool structure. Subsequently, we compare this simulation with the IDDQN algorithm. Next, we evaluate the performance of using only the second-order temporal difference method in DDQN algorithm training iterations. Finally, we compare the results of this simulation with the IDDQN algorithm, studying the performance of the algorithm when only the second-order temporal difference module is employed.

Tab. 6 presents a comparison of path conditions, convergence of step sizes, convergence of reward values, and average reward values after convergence based on Fig. 13, Fig.14, and Fig.15. The data represent the average values after 10 experiments. It can be observed that using only Module A cannot find an optimal path, while using only Module B and the IDDQN method can both find an optimal path. The convergence performance of step sizes using only Module A is 17.79% lower than the IDDQN method, and the convergence performance of reward values is 62.59% lower. Due to falling into a local optimum, the reward values after convergence have little reference value. Using only Module B shows a 14.04% lower convergence performance in step sizes and a 16.79% lower convergence performance in reward values compared to the IDDQN method. The average reward values after convergence are 0.89% lower than the IDDQN method.

It is evident that the performance of using a single module is not as good as the IDDQN method. Each module plays its role, with the second-order temporal difference module (Module B) mainly contributing to selecting better strategies. The fact that the average reward values after convergence

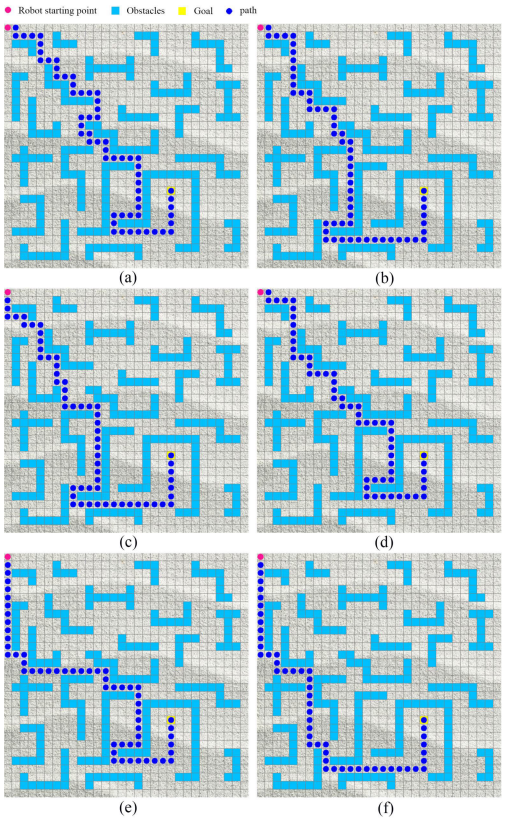

FIGURE 10. Optimal trajectories of algorithms in a complex environment.

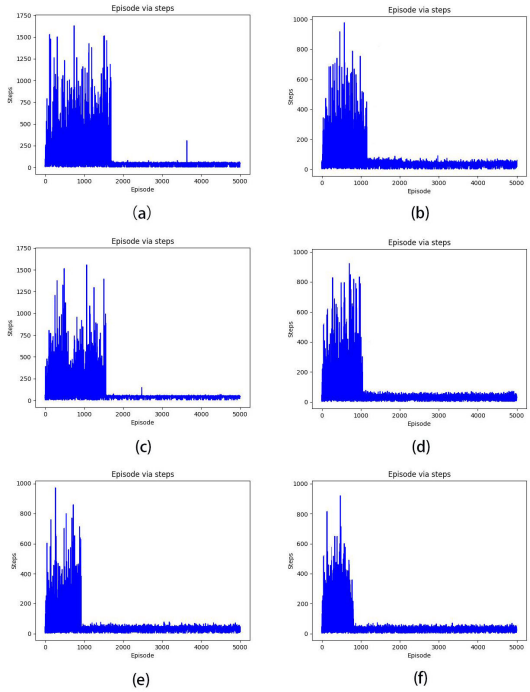

FIGURE 11. The graph depicts the variation of step length with the number of iterations for algorithms in a complex environment.

TABLE 4. The statistics of required episodes for step length convergence of algorithms in a complex environment are as follows.

Algorithm	Required episode for step convergence	Percentage (baseline:DDQN)
DQN	1551	-29.79%
DDQN	1195	0%
TLS-DQN	1423	-19.08%
MS-DDQN	1084	9.29%
ECMS-DDQN	966	19.16%
IDDQN	798	33.22%

are not much lower than the IDDQN method indicates that this module helps improve the accuracy of path planning. Since this module aids in finding better strategies, it also accelerates the convergence of the algorithm. On the other hand, the binary tree module (Module A) primarily helps reduce the retrieval time for historical information, further speeding up the convergence rate based on the second-order temporal difference module.

D. SUMMARY

In summary, proposing the use of a second-order temporal difference method to evaluate the results of DDQN algorithm training iterations and employing a binary tree structure instead of an experience pool structure for storing algorithm iterations to improve DDQN algorithm have shown promising results. These modifications enable more accurate estimation of action values, reduce the time required for action selection, enhance training efficiency, and expedite convergence.

Simulations were conducted for DQN, DDQN, TLS-DQN, MS-DDQN, ECMS-DDQN, and IDDQN algorithms in both simple and complex environments. In a simple environment, all six algorithms found the optimal path. Regarding performance metrics such as step convergence speed, reward convergence speed, and average reward values after convergence, the IDDQN method outperformed the DDQN method by 26.89%, 22.58%, and 10.30%, respectively. It also showed superiority over DQN, TLS-DQN, and MS-DDQN methods but exhibited a slight performance difference (0.48%) in step convergence speed compared to ECMS-DDQN. In terms of reward convergence speed and average reward values after convergence, IDDQN performed slightly better than ECMS-DDQN.

In a complex environment, DQN, DDQN, TLS-DQN, MS-DDQN, and ECMS-DDQN methods all encountered local optima and failed to find the optimal path. Consequently, the average reward values after convergence in this scenario were deemed not reliable. However, IDDQN demonstrated the most significant improvement over DDQN in terms of step convergence speed (33.22%) and reward convergence speed (25.47%). It exhibited a clear advantage over the other four methods, including the most effective

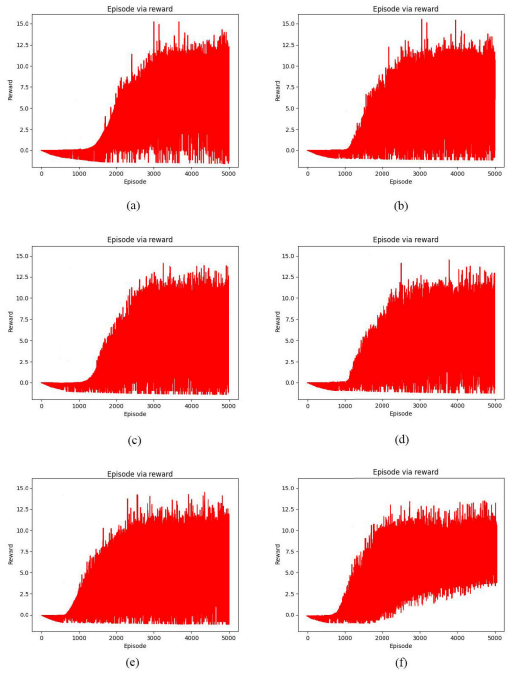

FIGURE 12. The graph illustrates the variation of reward values with the number of iterations in a complex environment.

TABLE 5. Statistics on the number of episodes required for reward convergence and the average reward value after convergence for algorithms in a simple environment are shown below.

Algorithm	Required episode for reward convergence	Percentage（baseline：DDQN）	The average reward value after convergence	Percentage（baseline：DDQN）
DQN	3016	-	12.50	2.88%
DDQN	2733	10.35% 0%	12.15	0%
TLS-DQN	2751	0.66%	12.25	0.82%
MS-DDQN	2442	10.65%	11.75	-3.29%
ECMS-DDQN	2221	18.73%	11.95	-1.65%
IDDQN	2037	25.47%	11.25	-7.41%

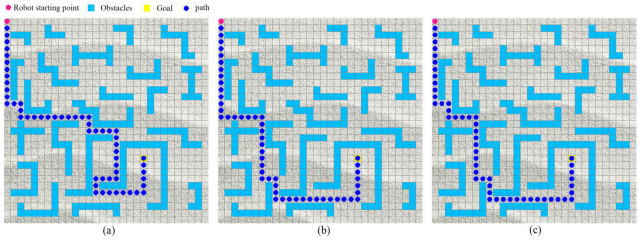

FIGURE 13. (a) Only using Module A, (b) Only using Module B, and (c) Comparing with IDDQN paths.

ECMS-DDQN. This underscores the effectiveness of IDDQN in complex environments.

In ablation experiments, comparing IDDQN with algorithms utilizing only the second-order temporal difference module or only the binary tree module highlighted that the second-order temporal difference module primarily improves action selection strategies, aiding the robot in choosing better actions and accelerating algorithm convergence to some extent. The binary tree module, building upon the use of the second-order temporal difference module, further expedited algorithm convergence.

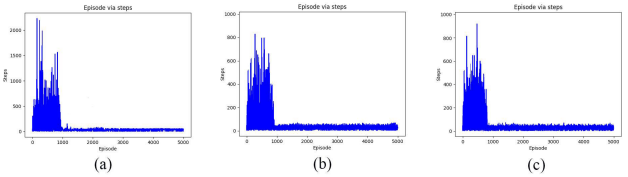

FIGURE 14. (a) Only using Module A, (b) Only using Module B, and (c) IDDQN Changes in step size with the number of iterations.

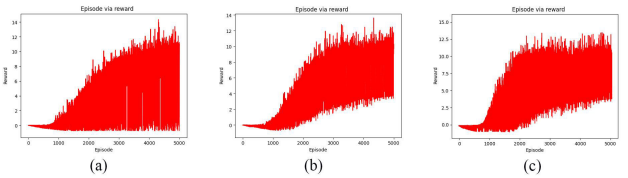

FIGURE 15. (a) Using only Module A, (b) Using only Module B, and (c) IDDQN changes in reward values with the number of iterations.

TABLE 6. Comparison of performance in ablation experiments.

Algorithm（baseline：IDDQN）	Only Module a is used	Only Module b is used	IDDQN
Whether it is the optimal path	NO	YES	YES
Required episode for step Convergence	941	908	798
Percentage	-17.79%	-14.04%	0%
Required episode for reward convergence	3312	2379	2037
Percentage	-62.59%	-16.79%	0%
The average reward value after convergence	11.95	11.15	11.25
Percentage	-6.22%	-0.89%	0%

VI. CONCLUSION

In this paper, an improved DDQN algorithm, named IDDQN, is proposed to address the issues of slow convergence speed and low accuracy in the presence of numerous obstacles in the DDQN algorithm of deep reinforcement learning. The proposed IDDQN algorithm utilizes the second-order temporal difference method and a binary tree data structure. Firstly, the second-order temporal difference method is introduced to evaluate the actions of the current robot, and this evaluation is then compared with the actions in the next iteration. When

the second-order temporal difference is significant, a new action should be chosen. The algorithm replaces the traditional experience pool structure with a binary tree structure to store the results obtained using the second-order temporal difference method, reducing the time required for retrieving historical information.

Subsequent simulation analyses were conducted to analyze the convergence of step lengths, reward values, average reward values after convergence, and the identification of optimal paths for DQN, DDQN, TLS-DQN, MS-DDQN, ECMS-DDQN, and IDDQN algorithms after multiple iterations in both simple and complex environments. In a simple environment, all six algorithms found an optimal path. Compared to the original DDQN algorithm, IDDQN showed improvements of 26.89%, 22.58%, and 10.30% in step convergence time, reward convergence time, and average reward values after convergence, respectively. IDDQN also outperformed DQN, TLS-DQN, and MS-DDQN, but the difference in step convergence speed with ECMS-DDQN was only 0.48%. In terms of reward convergence speed and average reward values after convergence, IDDQN slightly outperformed ECMS-DDQN.

In a complex environment, all methods except IDDQN encountered local optima, failing to find the optimal path, making the average reward values after convergence unreliable. IDDQN demonstrated the largest improvements over DDQN in terms of step convergence speed (33.22%) and reward convergence speed (25.47%). It also showed a clear advantage over the other four methods, including the most effective ECMS-DDQN, confirming the effectiveness of IDDQN in complex environments.

The study focused on researching the IDDQN algorithm in a static environment, proving its superiority over DQN and DDQN algorithms in environments with many obstacles. However, in real-world scenarios, dynamic obstacles may exist, and the study may have limitations in dynamic environments. Therefore, future research will investigate the performance of the IDDQN algorithm in dynamic environments.

REFERENCES

[1] Y. Chen and X. Zhou, "Research and implementation of robot path planning based on computer image recognition technology," *J. Phys., Conf. Ser.*, vol. 1744, no. 2, Feb. 2021, Art. no. 022097, doi: 10.1088/1742-6596/1744/2/022097.

[2] Y. Quan, H. Ouyang, C. Zhang, S. Li, and L.-Q. Gao, "Mobile robot dynamic path planning based on self-adaptive harmony search algorithm and morphin algorithm," *IEEE Access*, vol. 9, pp. 102758–102769, 2021, doi: 10.1109/ACCESS.2021.3098706.

[3] B. B. K. Ayawli, R. Chellali, A. Y. Appiah, and F. Kyeremeh, "An overview of nature-inspired, conventional, and hybrid methods of autonomous vehicle path planning," *J. Adv. Transp.*, vol. 2018, pp. 1–27, Jul. 2018, doi: 10.1155/2018/8269698.

[4] S. Campbell, N. O'Mahony, A. Carvalho, L. Krpalkova, D. Riordan, and J. Walsh, "Path planning techniques for mobile robots a review," in *Proc. 6th Int. Conf. Mechatronics Robot. Eng. (ICMRE)*, Feb. 2020, pp. 12–16, doi: 10.1109/ICMRE49073.2020.9065187.

[5] Y. Zhu, K. Chu, X. Chen, X. Wang, and H. Su, "Research and application of a multi-degree-of-freedom soft actuator," *Sens. Actuators A, Phys.*, vol. 338, May 2022, Art. no. 113492, doi: 10.1016/j.sna.2022.113492.

[6] P. Phueakthong, J. Varagul, and N. Pinrath, "Deep reinforcement learning based mobile robot navigation in unknown environment with continuous action space," in *Proc. 5th Int. Conf. Intell. Auto. Syst. (ICoIAS)*, Sep. 2022, pp. 154–158.

[7] B. Pradhan, D. Roy, and N. Hui, "Multi-agent navigation and coordination using GA-fuzzy approach: SocProS 2017," *Adv. Intell. Syst. Comput.*, vol. 2, pp. 793–805, Jan. 2019, doi: 10.1007/978-981-13-1595-4_63.

[8] S. Aggarwal and N. Kumar, "Path planning techniques for unmanned aerial vehicles: A review, solutions, and challenges," *Comput. Commun.*, vol. 149, pp. 270–299, Jan. 2020, doi: 10.1016/j.comcom.2019.10.014.

[9] H. Qin, S. Shao, T. Wang, X. Yu, Y. Jiang, and Z. Cao, "Review of autonomous path planning algorithms for mobile robots," *Drones*, vol. 7, no. 3, p. 211, Mar. 2023, doi: 10.3390/drones7030211.

[10] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. SSC-4, no. 2, pp. 100–107, Jul. 1968, doi: 10.1109/TSSC.1968.300136.

[11] M. Steinbrunn, G. Moerkotte, and A. Kemper, "Heuristic and randomized optimization for the join ordering problem," *VLDB J. Int. J. Very Large Data Bases*, vol. 6, no. 3, pp. 191–208, Aug. 1997, doi: 10.1007/s007780050040.

[12] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots in cluttered environments," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 1990, pp. 572–577, doi: 10.1109/ROBOT.1990.126042.

[13] J. Kennedy, "The particle swarm: Social adaptation of knowledge," in *Proc. IEEE Int. Conf. Evol. Comput. (ICEC)*, Apr. 1997, pp. 303–308, doi: 10.1109/ICEC.1997.592326.

[14] X. Wu, G. Wei, Y. Song, and X. Huang, "Improved ACO-based path planning with rollback and death strategies," *Syst. Sci. Control Eng.*, vol. 6, no. 1, pp. 102–107, Jan. 2018, doi: 10.1080/21642583.2018.1471426.

[15] D. Zhao, H. Wang, K. Shao, and Y. Zhu, "Deep reinforcement learning with experience replay based on SARSA," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Dec. 2016, pp. 1–6, doi: 10.1109/SSCI.2016.7849837.

[16] E. S. Low, P. Ong, and K. C. Cheah, "Solving the optimal path planning of a mobile robot using improved Q-learning," *Robot. Auton. Syst.*, vol. 115, pp. 143–161, May 2019, doi: 10.1016/j.robot.2019.02.013.

[17] H. V. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI Conf. Artif. Intell.*, vol. 30, Sep. 2015, pp. 2094–2100, doi: 10.1609/aaai.v30i1.10295.

[18] P. Heredia and S. Mou, "Finite-sample analysis of multi-agent policy evaluation with kernelized gradient temporal difference," in *Proc. 59th IEEE Conf. Decis. Control (CDC)*, Dec. 2020, pp. 5647–5652, doi: 10.1109/CDC42340.2020.9303966.

[19] N. Gottlieb and M. Werman, "DecisioNet: A binary-tree structured neural network," Jul. 2022, *arXiv:2207.01127*, doi: 10.48550/arXiv.2207.01127.

[20] N. A. Khaleq and A. Ai-Araji, "Intelligent hybrid path planning algorithms for autonomous mobile robots," *Int. J. Intell. Eng. Syst.*, vol. 15, pp. 309–325, Jul. 2022, doi: 10.22266/ijies2022.1031.28.

[21] A. A. Rashed, A. Ai-Araji, and M. N. Abdullah, "Static and dynamic path planning algorithms design for a wheeled mobile robot based on a hybrid technique," *Int. J. Intell. Eng. Syst.*, vol. 15, pp. 167–181, May 2022, doi: 10.22266/ijies2022.0831.16.

[22] T. Schaul, J. Quan, L. Antonoglou, and D. Silver, "Prioritized experience replay," Nov. 2015, *arXiv:1511.05952*.

[23] C. Yi and M. Qi, "Research on virtual path planning based on improved DQN," in *Proc. IEEE Int. Conf. Real-time Comput. Robot. (RCAR)*, Sep. 2020, pp. 387–392, doi: 10.1109/RCAR49640.2020.9303290.

[24] Y. Gu, Z. Zhu, J. Lv, L. Shi, Z. Hou, and S. Xu, "DM-DQN: Dueling munchausen deep Q network for robot path planning," *Complex Intell. Syst.*, vol. 9, no. 4, pp. 4287–4300, Dec. 2022, doi: 10.1007/s40747-022-00948-7.

[25] N. Saito, T. Oda, A. Hirata, K. Toyoshima, M. Hirota, and L. Barolli, "A movement adjustment method for DQN-based autonomous aerial vehicle," in *Proc. Int. Workshop Intell. Netw. Collaborative Syst.*, Jan. 2022, pp. 136–148, doi: 10.1007/978-3-030-84910-8_15.

[26] X. Gao, Y. Dong, and Y. Han, "An optimized path planning method for container ships in Bohai bay based on improved deep Q-learning," *IEEE Access*, vol. 11, pp. 91275–91292, 2023, doi: 10.1109/access.2023.3307480.

[27] Y. Long and H. He, "Robot path planning based on deep reinforcement learning," in *Proc. IEEE Conf. Telecommun., Opt. Comput. Sci. (TOCS)*, Dec. 2020, pp. 151–154, doi: 10.1109/TOCS50858.2020.9339752.

[28] Y. Wang, C. Jiang, and T. Ren, "UAV path planning based on DDQN for mountain rescue," in *Proc. Int. Conf. Intell. Robot. Appl.*, Aug. 2022, pp. 509–516, doi: 10.1007/978-3-031-13841-6_46.
[29] Y. Xiaofei, S. Yilun, L. Wei, Y. Hui, Z. Weibo, and X. Zhengrong, "Global path planning algorithm based on double DQN for multi-tasks amphibious unmanned surface vehicle," *Ocean Eng.*, vol. 266, Dec. 2022, Art. no. 112809, doi: 10.1016/j.oceaneng.2022.112809.
[30] Z. Chu, F. Wang, T. Lei, and C. Luo, "Path planning based on deep reinforcement learning for autonomous underwater vehicles under ocean current disturbance," *IEEE Trans. Intell. Vehicles*, vol. 8, no. 1, pp. 108–120, Jan. 2023, doi: 10.1109/TIV.2022.3153352.
[31] X. Peng, R. Chen, J. Zhang, B. Chen, H.-W. Tseng, T.-L. Wu, and T.-H. Meen, "Enhanced autonomous navigation of robots by deep reinforcement learning algorithm with multistep method," *Sensors Mater.*, vol. 33, no. 2, p. 825, Feb. 2021, doi: 10.18494/sam.2021.3050.
[32] X. Zhang, X. Shi, Z. Zhang, Z. Wang, and L. Zhang, "A DDQN path planning algorithm based on experience classification and multi steps for mobile robots," *Electronics*, vol. 11, no. 14, p. 2120, Jul. 2022, doi: 10.3390/electronics11142120.

JIANG SHUHAI (Member, IEEE) received the B.E. degree in mechanical design and manufacturing and the Ph.D. degree in forest engineering from Northeast Forestry University, Harbin, China, in 1986 and 2000, respectively. He was a Researcher with the Postdoctoral Research Station of Forestry Engineering, Nanjing Forestry University. He was a Lecturer and an Associate Professor with Nanjing Forestry University, in 2002 and 2006, respectively. In 2007, he was appointed as the Director of the Institute of Intelligent Control and Robot Technology, Nanjing Forestry University. For many years, he was engaged in the research of robots and automation. He has successively participated in the research and development of rotary cutting intelligent centering wood climbing robot, new intelligent stump cleaning robot, and led the team to carry out research and development of multifunctional mobile intelligent robot system, motion control and system integration of palletizing and handling robot, hexapod bionic disaster reduction and rescue robot, forest ember detection and cleaning robot, and outdoor food delivery robot. His current research interests include robot and automation, mechanism modeling and control, pattern recognition and intelligent systems, and deep learning. He is a member of the Chinese Society of Automation.

SUN SHANGJIE was born in Zhenjiang, Jiangsu, China, in 1998. They received the bachelor's degree in engineering from Nanjing Forestry University, in 2021, where they currently pursuing the master's degree in robotics engineering.

In 2020, they published a article titled "Forest Fire Robot Path Planning Based on Deep Learning" in the "*Forest Engineering*" journal. In 2021, they were granted a patent for an "Intelligent Flower Pot Based on STM32." In 2022, their article titled "Stability Analysis of the Food Delivery Robot with Suspension Damping Structure" was published in the "*Heliyon*" journal. Their research interests include robot vision perception and path planning.

LI CUN was born in Baoding, Hebei, China, in 2001. They received the bachelor's degree in engineering from Nanjing Forestry University, in 2022, where they currently pursuing the master's degree in control science and engineering.

In 2023, they applied for the Jiangsu Provincial Graduate Student Research and Innovation Program Project, and the name of the applied project was Research on Path Planning Methods for Outdoor Mobile Robots Based on Visual Perception. Their research interests include robot vision perception and path planning.