DP-A*: For Path Planing of UGV and Contactless Delivery

Xingli Gan, Zhihui Huo, and Wei Li

***Abstract*— The unmanned logistics and distribution urgently require a large number of unmanned ground vehicles(UGVs) under the influence of the potential spread of the Coronavirus Disease 2019 (COVID-19). The path planning of UGV relies excessively on SLAM map, and has no self-optimization and learning ability for the space containing a large number of unknown obstacles. In this paper, a new dynamic parameter-A* (DP-A*) algorithm is proposed, which is based on the A* algorithm and enables the UGV to continuously optimize the path while performing the same task repeatedly. First, the original evaluation functions of the A* algorithm are modified by Q-Learning to memory the coordinates of unknown obstacle. Then, Q-table is adopted as an auxiliary guidance for recording the characteristics of environmental changes and generating heuristic factor to overcome the shortcoming of the A* algorithm. At last, the DP-A* algorithm can realize path planning in the instantaneous changing environment, record the actual situation of obstacles encountered, and gradually optimize the path in the task that needs multiple explorations. By several simulations with different characteristics, it is shown that our algorithm outperforms Q-learning, Sarsa and A* according to the evaluation criteria such as convergence speed, memory systems consume, Optimization ability of path planning and dynamic learning ability.**

***Index Terms*— Path planning, unmanned logistics, contactless delivery, reinforcement learning, A-Star, Q-learning, robotics.**

I. Introduction

IN RECENT years, UGVs has developed rapidly in the disinfection of environments under COVID-19, contactless logistics and delivery, Urban Search and Rescue(USAR), or other areas that have received unprecedented attention in military and commercial applications. With the deepening of robot application technology, improving the ability of online autonomous plan path to rescure robots in complex environments is becoming more and more essential. Route planning based on building plans, lidar point cloud images, plane escape maps, and other pre-disaster maps is one of the most challenging tasks due to the tremendous environmental damage caused by the disaster and the uncertainty of the environment. Path planning technology is the key breakthrough problem to enable efficient autonomous navigation of mobile robots [1]. Only by solving these problems can the rescue robot genuinely develop into an intelligent, mobile, and lightweight robot.

According to the search task requirements in ordinary cases, excellent path planning algorithms can ensure that the intelligent robot completes the specified task at once. Researchers often consider improved performance metrics in path planning algorithms, more common ones being the time used to plan the path, the length of the path, and whether the path meets the kinematic constraints. However, some robots lack adaptability to their surrounding environments in unknown and variable environments. Since traditional algorithms tend to focus on point-to-point tasks, it not only causes a waste of planning time for the same task later but also fails to optimize its path into dangerous areas in variable environments, causing unnecessary loss [2]. Therefore, it is of great relevance and practical value to use big data to improve the path planning ability of unmanned vehicles.

Most of the pieces of literature focus on the improvement of obstacle avoidance ability of robots in known or unknown environments, or the length of path planning and time efficiency of task execution in a task [3]. When a robot performs multiple delivery tasks based on SLAM maps, the robot is unable to optimise its decision making capabilities based on the changing environment. Secondly, it should pay more attention to whether the robot can automatically optimize the path according to the explored space when performing the task many times.

Path planning algorithms in practical applications can be divided into global path planning and local path planning according to their functional characteristics. Most pieces of literature focus on improving the obstacle avoidance ability of robots in known environments, or on the length of path planning and the time efficiency of task execution in a single task [3]. Traditional algorithms tend to focus on point-to-point tasks, it not only causes a waste of planning time for the same task later but also fails to optimize its path into dangerous areas in variable environments, causing unnecessary loss [4].

In recent years, reinforcement learning (RL) techniques have received a lot of attention in the field of autonomous robots, which can continuously gain experience in tasks through reinforcement learning and continuously optimize

Manuscript received 22 August 2022; revised 15 November 2022; accepted 29 December 2022. Date of publication 12 April 2023; date of current version 17 January 2024. This work was supported in part by the National Key Research and Development Program of China under 2020YFC1511700. The Associate Editor for this article was Y. Yuan. *(Corresponding authors: Zhihui Huo; Wei Li.)*

Xingli Gan and Zhihui Huo are with the School of Information and Electronic Engineering, Zhejiang University of Science and Technology, Hangzhou 310023, China (e-mail: 120070@zust.edu.cn; 222008855010@zust.edu.cn).

Wei Li is with the National Time Service Center, Chinese Academy of Sciences, Xi'an 710600, China (e-mail: 610883554@qq.com).

Digital Object Identifier 10.1109/TITS.2023.3258186

TABLE I
CLASSIFICATION AND INTRODUCTION OF THE PATH PLANNING ALGORITHM

Algorithm group	Technique	Technique description	Examples
Search-Based	Dijkstra	Known nodes search space with the node weight.The strategy of the greedy algorithm	[11] [12]
	A* family	Eg: D*, Hybrid A*. Heuristic search.The better the evaluation function or the better the algorithm	[13] [14]
Sampling-Based	PRM	Probabilistic Roadmap:For local path planning, the better path is selected by scoring	[15][16]
	RRT family	Eg: RRT*, Informed RRT*. Physical and logical bias are used to generate the random-tree. disadvantage: local optimal solution.	[17][18]
Intelligent algorithm-Based	Genetic algorithm	Adapted to complex dynamic environments;requiring high computational costs and complex parameter tuning, and are suitable for large robotic scenarios	[21][22][23]
	Ant Colony Optimization		
Reinforcement Learning-Based	Policy-Based	agent'sperformance is evaluated by a reward function R. the main challenges:managing the trade-off between exploration and exploitation	[24][25][26]
	Value-Based		

their capabilities and improve efficiency through historical experience [5]. For path planning with unknown environment in real-world applications, the training speed of RL is hardly sufficient for the application requirements. We should pay more attention that the UGV can have heuristic search ability when performing tasks many times.

In this paper, the following contributions are mainly included to improve the A* algorithm, and the main idea is to combine the self-learning mode of RL with A* to realize path planning in an unknown environment, that is, to explore the environment while self-learning knowledge while planning the path. firstly, some traditional algorithms rely excessively on the initial map and cannot be optimized in complex environment with unknown obstacles. Inspired by the Q-Learning algorithm, Q-table is added to A* that records the characteristics of environmental changes based on sensor data. Secondly, in order to utilize the environmental data recorded in the Q-table, the heuristic function in the A* algorithm is modified and added a new decision factor function to consider special cases. The traditional heuristic function relies mainly on the initial map and distance equation cannot better determine the unknown obstacles or new feasible paths. The new heuristic function can continuously optimize the path planning algorithm under multiple tasks based on the reward values recorded in Q-table and solve the problems of slow convergence or deadlock. Finally, because the A* algorithm is based on grid maps and the space environment is complex and changeable, a large number of redundant nodes are recorded in the Q-table, then, a pruning strategy and a choice of node cost-worthy decision method are proposed to improve the efficiency of planning paths.

The remainder of this paper is structured as follows: Section II classifies general path planning algorithms into four categories and analyzes their advantages and disadvantages. Section III presents the definition and assumptions of the multitask-based path planning problem with unknown obstacles, an improved A* algorithm by adding Q-table and modifying its heuristic function is proposed. Section IV provides discussion on the experimental results of the DP-A* algorithm. Section V describes the results and concludes the paper.

II. RELATED WORK

Generally, the global path planner usually calculates the path with fixed obstacles offline in the static environment, while the local path planner just makes up the disadvantage of the global path planner. It typically computes the path of moving obstacles online in a dynamic environment based on real-time information sent by sensors such as lidar.

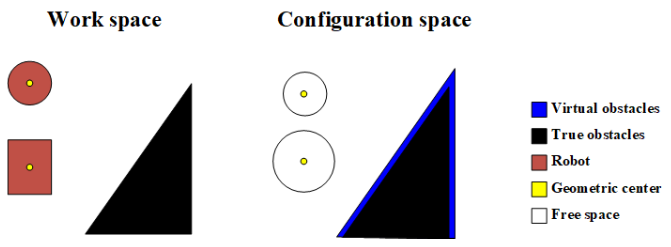


Fig. 1. Conversion from workspace to configuration Space.

Although the algorithm proposed in this paper is based on the improvement of the traditional path planning algorithm A*, to better understand the research status of robot path planning algorithms and the advantages and disadvantages in various application scenarios we early will expand the scope of research all the widely used algorithms, rather than limit our attention to the traditional algorithm. The most common path planning algorithms can be roughly divided into four categories (as shown in Table I): search-based path planning algorithm, sampling-based path planning algorithm, simulated evolutionary optimization algorithm, and reinforcement learning algorithm [6]. The following briefly introduces some classical algorithms in the above classification and their characteristics.

A. Search-Based Path Planning Algorithm

For some path planning tasks, the most basic idea is to traverse the entire data until the end point is found. The range of motion of the robot in the real world can be referred to as the workspace, and the data between the agent and the environment needs the configuration space to be completed, describing the location of the object in the environment and the entire workspace [7], [8]. As shown in Fig. 1, we call the actual exploration area of the robot a workspace. In the workspace, robots are usually irregular geometry, which is detrimental to the path planning and collision detection [9], [10]. Usually we have to convert the workspace into a configuration space, where the robot is path planned as a point and the obstacles will have a certain radius of expansion for collision detection. From the point of view of path exploration, the initial point to the target point exploration is to use some method in the configuration space to gradually search each grid, until the accessed grid is the target point, to give a solution to the path planning problem, which is not necessarily optimal for the path length, and the time complexity and spatial complexity are relatively not to be ignored. However, due to its excellent robustness and stability, many algorithms have been used in the development of self-driving cars and delivery robots based on their characteristics.

Dijkstra algorithm: It is suitable for solving single point shortest path problems in entitled graphs and requires that the node weights are non-negative. A single complete traversal search yields the shortest path from a vertex to each of the remaining nodes. A description of the basic flow of the algorithm and a more detailed scenario of the application can be found in [11]. The Dijkstra algorithm is being extensively studied in the field of unmanned driving and vehicle-road collaboration [12].

A* algorithm: It is a graph-based search algorithm where the node selection strategy is influenced by a heuristic function that can have efficient search efficiency (it is an extension of Dijkstra's graph search algorithm) [13]. One of the most important parts is the definition of the decision function, which will directly affect the search efficiency of the algorithm. The proposed algorithm is relatively applicable to the prophetic space and is extremely widely used in indoor robots, but the time complexity and space complexity consumed in the environment of large regions are large [14].

B. Sampling-Based Path Planning Algorithm

Algorithms based on global random sampling strategy do not require parameters of grid maps and node weights, and this class of algorithms is used randomly in the configuration space or with some policy constraints on the sampling range. This type of algorithm will expand outwards from the starting point into an irregular search tree according to the policy until the distance between the leaf nodes and the end point equals the set threshold. The algorithm always searches for the final path as long as enough nodes are generated. Although it guarantees the probabilistic completeness of the result, the optimality of the result is difficult to guarantee. The most typical algorithms used in path planning are the Probabilistic Road Map Method (PRM) [15] and the Rapid Exploration Random Tree (RRT) [16].

Rapid exploration random tree (RRT): this is the most primitive path planning algorithm based on spatial sampling, which is suitable for environments with sparse spatial obstacles [17]. It allows fast planning in a semi-structured space [18] and often also incorporates robot kinematic constraints (e.g., maximum steering angle of the robot) in practical applications. In Karaman et al. [19] a new approach of this algorithm was developed, named RRT*. This new search strategy can improve the probability of obtaining a locally optimal solution [20]. The biggest drawback of this class of algorithms is that it is inefficient and never guarantees that the path is optimal. For very narrow regional sampling points, only a very small probability falls inside, which is very unsuitable for application in complex environments. Relevant algorithms derived from this method are also constantly solving the above shortcomings.

C. Simulated Evolutionary Optimization Algorithm

Inspired by biological group behavior, evolutionary advantage, and community structure, intelligent bionic algorithms are highly robust and self-organizing [21]. Intelligent bionic algorithms have better effects than traditional ones, so for the more complex path planning scenarios, intelligent bionic algorithms stand out. The Intelligent biomimetic algorithm is an intelligent computational method that mimics biological groups' intelligent behavior or organism structure and function or ecological mechanism. The main feature of these algorithms is that 1. The working mechanism of intelligent bionic algorithms is very close to the nature or ecological mechanism of natural organisms. 2. Intelligent bionic algorithms are simple and easy to understand, but the results are amazing, reflecting the emergency. 3. Intelligent bionic algorithm can improve the adaptive ability through self-learning or self-organization [22].

The most classical Deed Particle Swarm Algorithm (PSO) [23]was initially been a population intelligence-based evolutionary computing technology proposed to simulate migration and cluster behavior in 1995 by Dr. Kennedy and Eberhart during bird foraging.

D. RL-Based Path Planning Algorithm

Reinforcement learning algorithms based on repeated exploration mechanisms have attracted wide attention in recent years; the main feature is no need to know environmental data in advance [24]. The core of the algorithm is to obtain the maximum cumulative reward by continuously exploring the environment. If an intelligence performs a specific action in a state and then receives a reward, the probability of selecting the rewarded action in that state in subsequent explorations increases [25], [26].

Due to the property that reinforcement learning does not require too many hyperparameters, it has been widely used in intelligent robot environment exploration and path planning [27], [28]. However, this property also causes fatal flaws in reinforcement learning. First, when the robot is placed in a complex environment, the complexity of the problem and the computational requirements increase exponentially with the increase of variables (the "curse of dimensionality"). In addition, slow convergence remains a problem for reinforcement learning [29]. Finally, since the training of an intelligence is often in a specific environment, this leads to poor portability and generality of reinforcement learning for the path planning problem [30]. In addition, reinforcement learning algorithms used in robot path planning algorithms are addressing environment-specific cognitive abilities to better optimize paths. The researchers neglected that the state values between nodes on the original map can also be considered as a priori experience. Therefore, it is novel and meaningful to propose a path planning algorithm based on the OccupancyGrid-Map module robot that can be applied to UGVs.

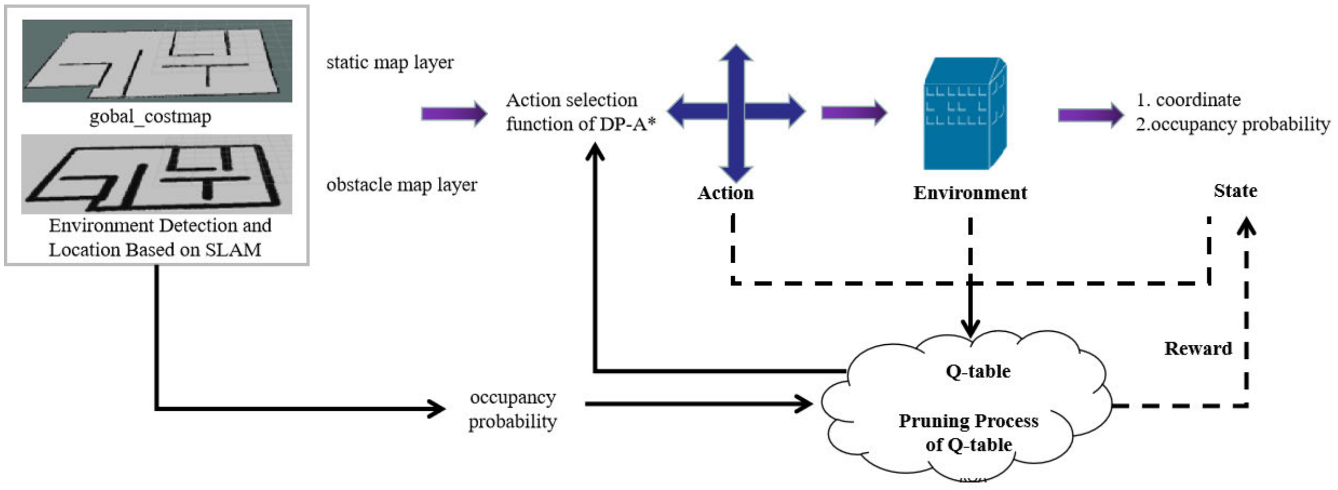


Fig. 2. DP-A*: Working mechanism with experience pools.

III. Improved A* Algorithm Combined With RL

The robot may need to explore the same target point many times or pass through the explored area due to task requirements. In traditional global planning algorithms, once the complex environment changes, the local planning algorithm needs to make a decision within a certain period of time and then re-plan the global path. However, this process is not repeatable. When the robot performs the same task or needs to pass the same target point next time, it will still repeat the previous operation. So the robot performs the same task many times, it will waste a lot of search time and reduce the efficiency.

Using reinforcement learning to improve the path search performance of the A* algorithm is called DP-A*, which can help the UGV record and learn changes in unknown environment when performing the path exploration task, and improve the autonomous learning ability in the path planning process. Fig. 2 shows the working mechanism with experience pools of the DP-A* algorithm.

A. Environment Detection and Location Based on SLAM

In practice, the path planning of unmanned vehicles cannot only rely on the algorithm level, which is unreliable and incomplete for itself and the task. Generally speaking, the path planning of robots dealing with various tasks often relies on various sensors such as depth cameras, millimeter-wave radar, and 2D/3D lidar to provide data sources (real-time data and preparation data) for the algorithm. To realize path planning, there must be three elements: positioning, mapping and navigation. The first two parts of robots often rely on SLAM (simultaneous localization and mapping), and then the path planning is developed again based on the mapping and localization of SLAM [31], [32]. In this case, robot path planning can be divided into: global path planning and local path planning (real-time). Let's briefly introduce the differences between the two. The global path planner usually calculates the fixed path of the obstacle offline in the static environment, while the local path planner only makes up for the deficiency of the global path planner [33], [34]. The algorithm usually calculates the path of the obstacle online in the dynamic environment according to the lidar and other sensors.

SLAM algorithm itself only completes the localization and map construction of the robot, which is not completely equivalent to the actual navigation. This part is based on the SLAM algorithm, which is known in the industry as motion planning [35], [36]. There are at least two levels of motion planning modules, one called global planning, which is a little bit like our in-car navigators. It requires pre-planning a route on a known map and the robot's position, which is usually provided by SLAM. There are two kinds of global planning algorithms in the global planning function package provided by ROS:

1) Dijkstra: A classical algorithm from one vertex to each of the remaining vertices using the greedy algorithm strategy, solves the shortest path problem in the entitled graph [37], [38]. The main feature is to start from the center to the external node until the extension to the goal node.
2) A*: A heuristic search algorithm, which improvement based on Dijkstra, or a combination of Dijkstra and best-first search (BFS). It's widely used in all kinds of mobile robots and games, such as real-time strategy games such as Starcraft and Warcraft, which use this algorithm to calculate the movement of units.

Next, we will introduce and compare the algorithm in detail, explain its advantages and disadvantages, and gradually improve the algorithm. The A * algorithm has a strong search ability for static weight graphs and is used for path planning for many traditional robots. Its search efficiency depends mainly on the determination of the heuristic function [39]. In The Robot Operating System (ROS), A* algorithm and Dijistra algorithm needs to be based on the known environment and mapping through SLAM, as shown in Fig. 3. As the heuristic information of traditional A* algorithms tends to be node weights or distances, the heuristic search has limited information and the intelligence lacks information about the

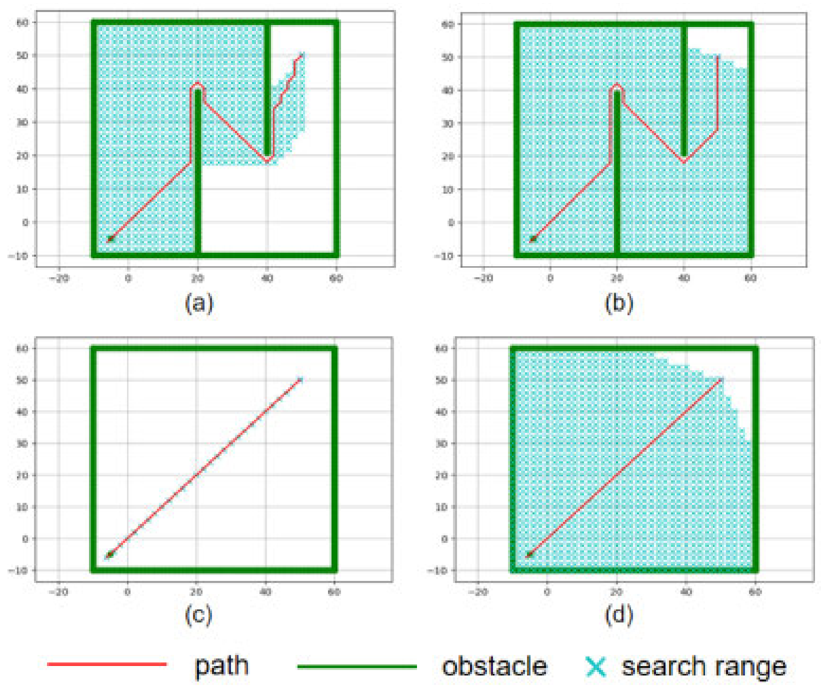


Fig. 3. Comparison of search area between A* and Dijkstra.

temporal state in the space. The more heuristic information used for Agent decision making, the more efficient the search will be. Therefore, it is better to introduce additional environmental information and robot state information in practical applications to reduce the search workload with guaranteeing the best path.

The search process using heuristic information is called heuristic search. The most significant difference between algorithms that use heuristic information search is the difference in the heuristic function used to determine the optimal node. For example, the local search method is to select the "best node" in the search process abandon other siblings and father nodes, and keep searching. The result of this search is obvious because other nodes are discarded, the best node may also be discarded because the best node is solved only in the stage is not necessarily the global best. Best priority algorithm to find the path to the relatively more accurate, it in the search, did not discard nodes (unless the node is dead), in every step of the valuation of the current node and the node before $f(n)$ compare with a "best of nodes", this can effectively prevent the loss of the "best node".

The heuristic information of the A* algorithm is represented by a special valuation function $f(n)$:

$$f(n) = g(n) + h(n) \tag{1}$$

where, $h(n)$ is the estimated cost of node n from the end point, and $h'(n)$ is the actual cost of node n from the end point. According to the characteristics, we can know that the A* algorithm has faster search efficiency compared with Dijkstra, and is very influenced by heuristic functions. The influence of heuristic function on the A* algorithm is as follows:

1) If $h(n) - g(n) \approx h(n)$, the search process can be regarded as $g(n) = 0$, and the A* algorithm degenerates into best-first search.
2) If $h(n) < h'(n)$: The solution is the optimal solution. However, the smaller $h(n)$, the more extended paths are needed to search the path, so the slower the running speed is.
3) If $h(n) > h'(n)$: Then A* algorithm is not guaranteed to find The shortest path, but it runs faster.
4) If $h(n) = h'(n)$: This is the most ideal case: the solution is the optimal solution and takes the least time. However, it is difficult to do this in the actual application scenario or the scenario without information. Because before reaching the end point, it is difficult to describe the cost of reaching the end point in the current state.

B. Q-Table Based on Reinforcement Learning

Reinforcement learning(evaluation learning) has evolved from the fields of machine learning, neural networks and parametric adaptive control theory [40], [41]. It is used to describe and solve the problem that an inexperienced agent uses its own increased knowledge of the environment to obtain the maximum benefit of a task through a process of constant interaction with the environment. Different from traditional machine learning and neural network, reinforcement learning does not require any data to be given in advance, but enhances its cognitive ability of the environment by continuously exploring the environmental information(state) and recording it to obtain feedback (reward) under the execution of each operation(action).

Agent in a "trial and error" approach to learning, through the guidance and environment, interact to obtain the reward

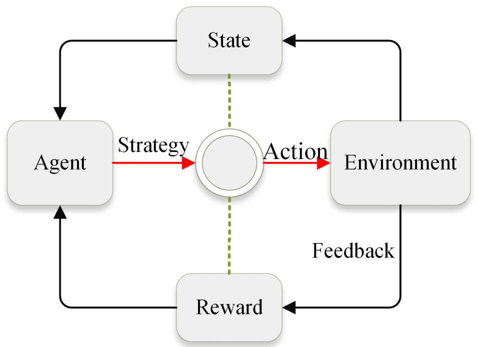


Fig. 4. Five elements of reinforcement learning.

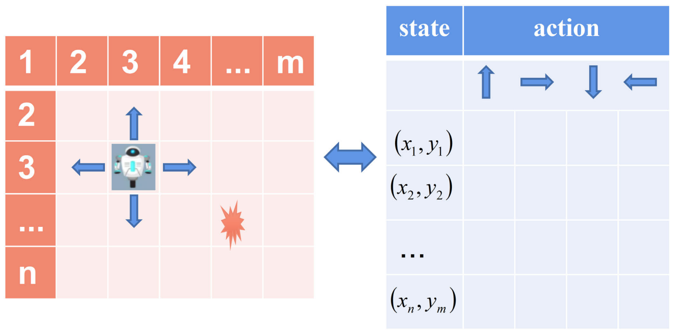


Fig. 5. Association between environmental characteristics and Q-table.

behavior. The purpose is to enable the agent to enhance the cognitive ability of the environment through feedback from the environment in order to better optimize the strategy. Reinforcement learning to learning as a testing evaluation process, the Agent choose an action for the strategy, after the action is executed, it will reach a new state according to the environment. Due to the change of state, it will generate an enhanced feedback signal (reward or punishment) to the agent. The same is true in the later process. The agent selects the action according to the policy, and the environment will give the enhanced feedback signal of the corresponding action according to the change. The differences of reinforcement learning algorithms include strategy function, learning mode and reinforcement feedback signal update mode. As shown in Fig. 4, it's the interrelation between the five elements of reinforcement learning Generally speaking, reinforcement learning models can be regarded as Markov decision processes (MDP).

Q-Learning algorithm is the basic algorithm for reinforcement learning that does not require complex neural networks and machine learning [42], [43]. It requires only a two-bit table to record the reward values obtained for actions in each state, so Q-Learning has been widely used since its introduction. After constant environmental exploration, the agent can generate a Q-table for decision making [35]. The structure of Q-table and environment is shown in Fig. 5, where the environment can be regarded as a two-dimensional map of $m \times n$.

In the two-dimensional table (Q-table) used to record reward values in the Q-Learning algorithm, the number of columns indicates how many actions are available in a state (which is usually determined), while the number of rows is usually determined by the environment characteristics, the intelligent performance of reinforcement learning usually requires a large order of magnitude and multiple exploration processes. The Q-table consists of three fundamental elements: $state\,(s)$, $action\,(a)$, $reward\,(r)$, then, the update strategy of Q-Learning is as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma max\left(x_a\right) \cdot Q\left(s', a'\right) - Q(s,a)\right] \tag{2}$$

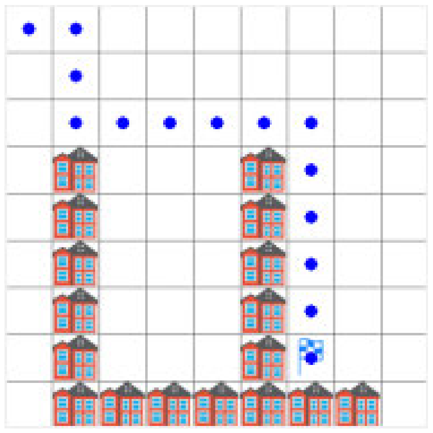
where, α and γ are the learning rate and discount factor of the agent, respectively. The formula shows that the larger the learning rate alpha, the less the effect of previous training is retained. The discount factor usually takes values in the range of 0 to 1. By adjusting it, the learning system can be controlled to have the ability to predict the future of its own actions. In the extreme case, when $\gamma = 0$ it only considers the current outcome of the action. As γ approaches 1, the future payoff becomes more important in taking the optimal action. If the agent performs action at in state s at time t, it will get immediate reward$R(s_t, a_t)$ and delayed reward$max\left(Q\left(S_{t+1}, a\right)\right)$. According to the Bellman equation, after a certain number of training sessions the robot obtains a Q-table with gradually converging reward values, according to which the robot's decisions tend to be optimal [35]. The process of the Q-Learning algorithm is shown as follows:

Algorithm 1 Q-Learning Algorithm

```
INITIALIZE $Q(s,a)$, parameters:step size $\alpha$
**Loop for each episode**
    INITIALIZE $S$
    LOOP FOR EACH STEP OF EPISODE:
        CHOOSE $A$ FROM $S$ USING POLICY DERIVED FROM $Q\,(e.g., \varepsilon - greedy)$
        TAKE ACTION $A$, OBSERVE $R$, $S'$
        $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma max\left(x_a\right) \cdot Q\left(s', a'\right) - Q(s,a)\right]$
        $S \leftarrow S'$
    UNTIL $S$ IS TERMINAL
```

At the same time, we will also introduce another reinforcement learning algorithm: Sarsa algorithm. Both Sarsa algorithm and Q-Learning algorithm use Q-table to store action value functions, but their update strategies are different. The process of the Sarsa algorithm is shown as follows:

According to the different update methods of the two algorithms, it is easy to see that when using the Q-Learning algorithm,the agent,although considering the maximum value of the reward obtained for the next action, does not necessarily choose it to execute after updating the state but updates the Q-Table to reselect it. The Sarsa algorithm differs from its update approach in that each update is given five parameters: S_t, A_t, R_t, S_{t+1}, A_{t+1}. A_{t+1} is also the action that will be executed after reaching the new state S_{t+1}. It also means that each time the action is selected it will be as far away from the obstacle as possible to reduce the risk of collision.


(a)

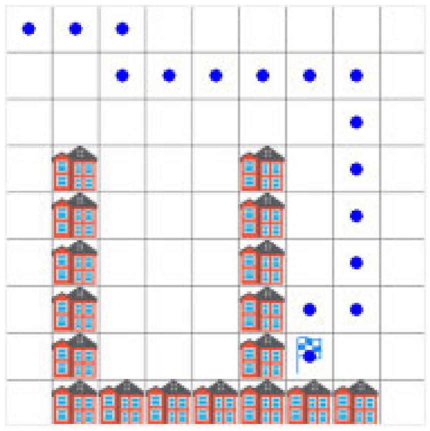

(b)

Fig. 6. Comparison of path planning characteristics between Q-Learning and Sarsa algorithm.

Algorithm 2 Sarsa Algorithm

```
INITIALIZE $Q(s,a)$, parameters:step size $\alpha$
Loop for each episode
    INITIALIZE $S$
    CHOOSE $A$ FROM $S$ USING POLICY DERIVED FROM $Q(e.g., \varepsilon - greedy)$
    LOOP FOR EACH STEP OF EPISODE:
        TAKE ACTION $A$, OBSERVE $R$, $S'$
        CHOOSE $A'$ FROM $S'$ USING POLICY DERIVED FROM $Q(e.g., \varepsilon - greedy)$
        $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \cdot Q(s',a') - Q(s,a)\right]$
        $S \leftarrow S'$, $A \leftarrow A'$
    UNTIL $S$ IS TERMINAL
```

This update method makes the agent's path to the target point increase and makes it difficult to complete the task in complex environments.

In general, Q-Learning algorithm and Sarsa algorithm are both effective path planning algorithms, with slightly different characteristics. As shown in Fig. 6a, when the Q-Learning algorithm is used, the mobile agent has the opportunity to explore towards the obstacle attachment due to the probabilistic nature of the action selection. In the case of using the Sarsa algorithm (Fig. 6b), the agent will always greedily choose the action for which the maximum reward can be obtained.

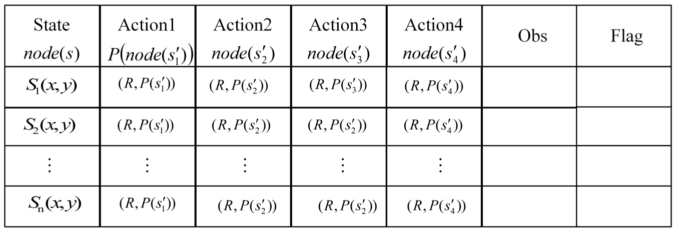
*C. Action Selection Based on DP-A**

The search efficiency of A* algorithm is higher, but its disadvantage is local optimization, and its performance is also closely related to the heuristic function. In addition, in some unknown environments, the reinforcement learning algorithm has stronger generalization ability, the disadvantage is that it requires a lot of training to gradually optimize the path. Based on the above discussion, the DP-A* algorithm is proposed. The improved ability of the DP-A* algorithm is that the UGV encounters some unknown obstacles when exploring the path, who can predict obstacles in advance and plan effective paths based on prior information. The DP-A* algorithm still uses the heuristic search strategy of the A* algorithm and the Q-table of the Q-Learning algorithm. The Q-table is used to record the missing environment information on the map, so as to provide feedback information and correct the UGV's global path planning in the next task. With the increase of exploration times, the global path planning of DP-A* algorithm will gradually approach the global optimization. The detailed process of DP-A* is shown as follows:

Algorithm 3 DP-A* Algorithm.

```
INITIALIZE $g(n)$, parameters: step size $\alpha$, Get Map of SLAM
Loop for each task
    INITIALIZE the Q-Tablepolicy derived from DP-A*
    CHOOSE $A$ FROM $S$ USING POLICY DERIVED FROM $DHA*$ AND VERDICT P OF $S - A$ FROM MAP
        GET $S$, $S'$, $P'$, CHOOSE $A$ FROM $S \rightarrow S'$
        $Q(s).P \leftarrow Q(s).P'$
        $Q(s,a).R \leftarrow Q(s,a).R + \triangle P$
        $Q(s,a).R \leftarrow Q(s,a).R + \left(Q(s,a).P + Q(s,a).P'\right)$
        $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma min(x_a) \cdot Q(s',a') - Q(s,a)\right]$
        $S \leftarrow S'$
    UNTIL $S$ IS TERMINAL
```

Here, the reinforcement learning strategy is still used to explore the path by the DP-A * algorithm, but the difference is that the UGV only plans the global path according to the information of SLAM map at the initial stage. With the accumulation of experience, the UGV's ability to predict the missing information will also increase when planning the path, which enables the UGV to effectively avoid the deadlock or long path problem caused by the lack of obstacle information on the SLAM Map when planning the global path. Therefore, the idea of finding the maximum expected value in the Q-Learning algorithm is transformed into finding the minimum path cost in the DP-A* algorithm. Eqn. 3 is that the most common expectation value is calculated of



State $node(s)$	Action1 $P(node(s_1'))$	Action2 $node(s_2')$	Action3 $node(s_3')$	Action4 $node(s_4')$	Obs	Flag
$S_1(x,y)$	$(R,P(s_1'))$	$(R,P(s_2'))$	$(R,P(s_3'))$	$(R,P(s_4'))$		
$S_2(x,y)$	$(R,P(s_1'))$	$(R,P(s_2'))$	$(R,P(s_2'))$	$(R,P(s_4'))$		
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$		
$S_n(x,y)$	$(R,P(s_1'))$	$(R,P(s_2'))$	$(R,P(s_2'))$	$(R,P(s_4'))$		

Fig. 7. Q-table structure for storing experience values.

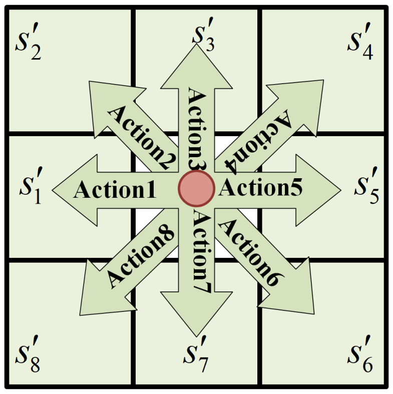


Fig. 8. Schematic diagram of robot movement on a two-dimensional map.

reinforcement learning.

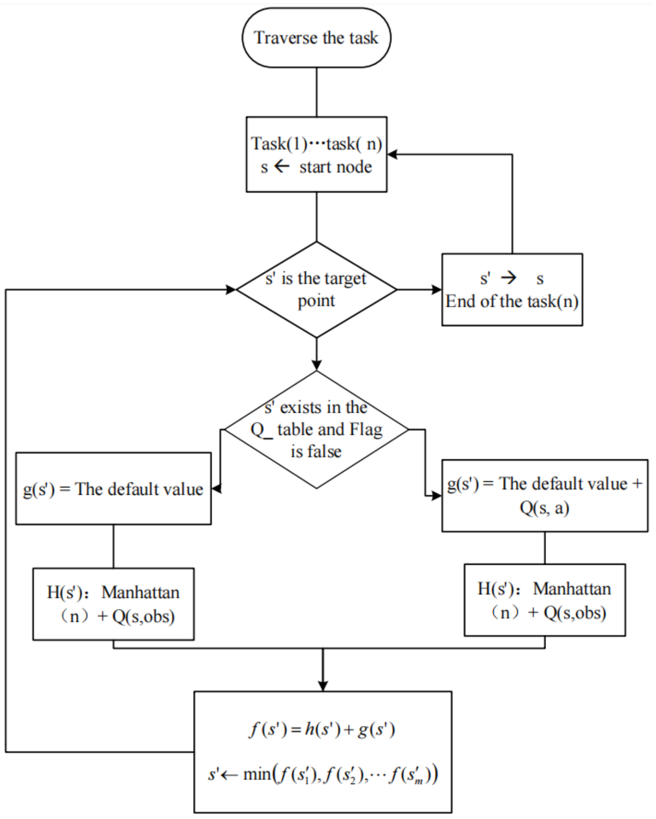
$$max_{\pi} E\left[\sum_{t=0}^{H} \gamma^{t} R\left(S_{t}, A_{t}, S_{t+1}\right) | \pi\right] \tag{3}$$

where, S represents the state, A is the action in the corresponding state, and $R(S, A)$ represents the reward value of the corresponding state and the action. In Eqn. 4, $f(s)$ is the weight of a certain state S, and our goal is to minimize the sum of weights of the entire path.

$$min \sum_{s=1}^{N} f(s) \tag{4}$$

Fig. 7 is the Q-table structure for storing experience values, it can be seen that The nodes selected on a two-dimensional map are not only calculated based on a heuristic function of distance, but also rely on exploration experience to assist decision-making. During the execution of each task, an additional value is added at each point through reinforcement learning in order to optimize the global path at the next task. In practice, this value is obtained through laser radar or derived from the state of the robot. The content of the Q-table is constantly updated with continuous exploration and changes in obstacles.

Each state s in the Q-table of the DP-A* algorithm corresponds to n actions. The number of actions is related to the movement mode. Generally, there are up to 8 movement modes in the path planning based on grid search, as shown in Fig. 8. The grayscale value of reaching the next state s' through action n is recorded in the action corresponding to each node s. Each pixel in the raster map takes on a value between [0,255], 255 for white and 0 for black, which is set to x. Eqn. 5 is used for normalization, the white area with the value of 0 indicates that the UGV can pass, the black area with the value of 1 indicates that the UGV cannot pass,and the gray area represents that the environment is unknown, and its value is between 0 and 1, as shown in Eqn.6. The execution reward and punishment value for each action is equal to p + reinforcement feedback r, as shown in Eqn.7. Flag is initialized to true, which indicating whether the status of surrounding nodes at state point s changes.



Traverse the task
Task(1)···task(n)
s ← start node
s' is the target point
s' → s
End of the task(n)
s' exists in the Q_ table and Flag is false
g(s') = The default value
g(s') = The default value + Q(s, a)
H(s'): Manhattan (n) + Q(s,obs)
H(s'): Manhattan (n) + Q(s,obs)
$f(s') = h(s') + g(s')$
$s' \leftarrow \min(f(s_1'), f(s_2'), \cdots f(s_m'))$

Fig. 9. Detailed flowchart of the DP-A* algorithm.

$$p = (255 - x)/255 \tag{5}$$
$$p' = \{-1 \mid p \geq O - th, 0 \mid p < O - th\} \tag{6}$$
$$R = p' + r \tag{7}$$

The flowchart of the DP-A* algorithm is shown in Fig. 9. If s' exists with Q-table and its flag is false, it means that the state of point s' has changed during the path exploration process, so the prediction function g (s') is Eqn. 8.

$$g(s_n') = s_n' - min[Q(S, A).R] \tag{8}$$

If s' does not exist in Q-table, which means that the state of s' has not changed or the state of s' has not changed according to experience, the prediction function g(s') of s' from the target point to that point is the default value.

$$h(s') = [(s_x' - g_x) + (s_y' + g_y)] + \Sigma R(Obs) \tag{9}$$

It is worth noting that the valuation function $h(s')$ from the current point to the end point also makes use of exploration experience to assist in the decision making. In general, the

State $node(s)$	⬆ $P(node(s'_1))$	➡ $node(s'_2)$	⬇ $node(s'_3)$	⬅ $node(s'_4)$	Obs	Flag
(1,3)	(0,0)	(0,0)	(0,0)	(0,0)	None	True
(2,3)	(0,0)	(-1,1)	(0,0)	(0,0)	3,3	False
(2,2)	(0,0)	(0,0)	(0,0)	(0,0)	None	True
(3,2)	(-1,1)	(0,0)	(0.99,0)	(0,0)	3,3	False
(3,1)	(1,0)	(1,0)	(1,0)	(1,0)	None	True

Fig. 10. The stored contents of Q-table after a complete exploration task.

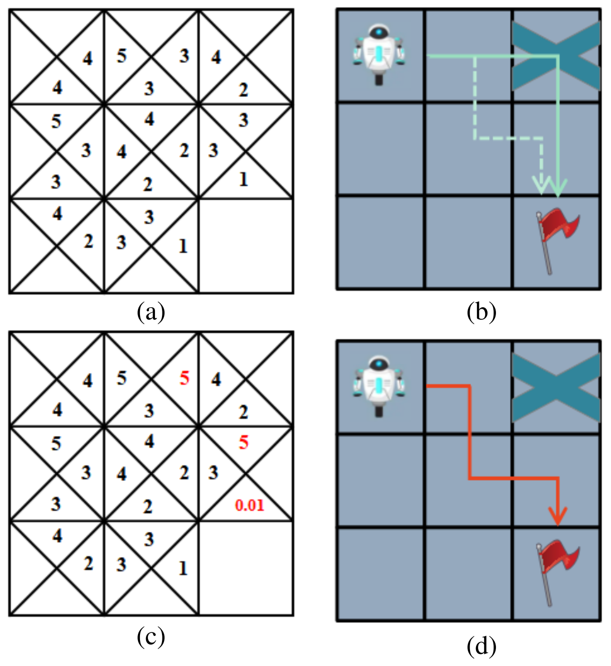


Fig. 11. The optimization process of the DP-A* algorithm.

valuation function uses the Manhattan method to estimate the cost of the current point s' to the end point, which is not limited by kinematic constraints and is applicable to most mechanical structures and realistic environments and is computationally fast since there are only two operations of addition and subtraction in this method. We add the predicted number of obstacles $\Sigma R(Obs)$ in the path based on the experience accumulated from previous exploration to make $h(s')$(Eqn.9) closer to the real cost.

$$f(s') = g(s') + h(s') \tag{10}$$

The difference between the DP-A* algorithm and the traditional A * algorithm is whether the path of the next task needs to be optimized according to the actual environment. The A* algorithm depends on the next action of the distance factor and the accuracy of the prior map, while the node weights $g(s)$ and $h(s)$ of the DP-A* algorithm are variable, as shown in Eqn.10. The update policy of the DP-A * algorithm is to mark all unknown environmental features encountered and update Q-table after a complete exploration task, as shown in Fig. 10.

Fig. 11 shows the optimization process of the DP-A* algorithm when there is only one unknown obstacle on the map. At the first path exploration, the UGV has no

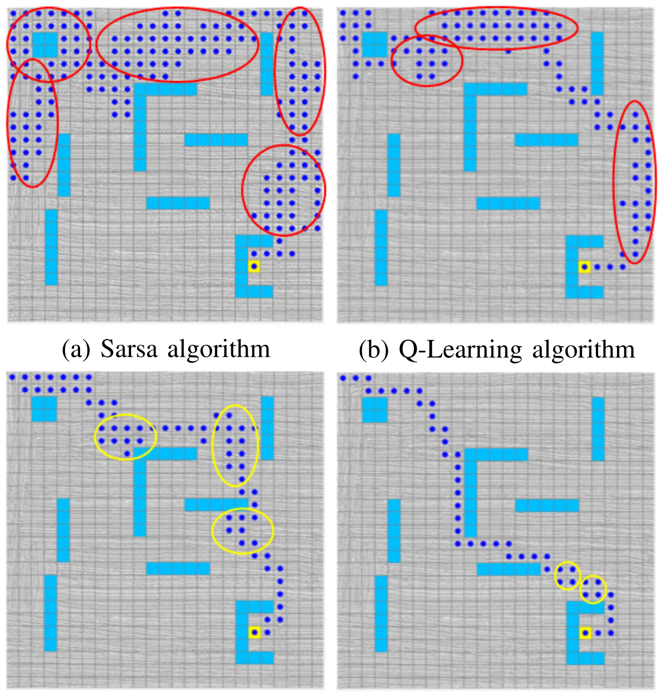
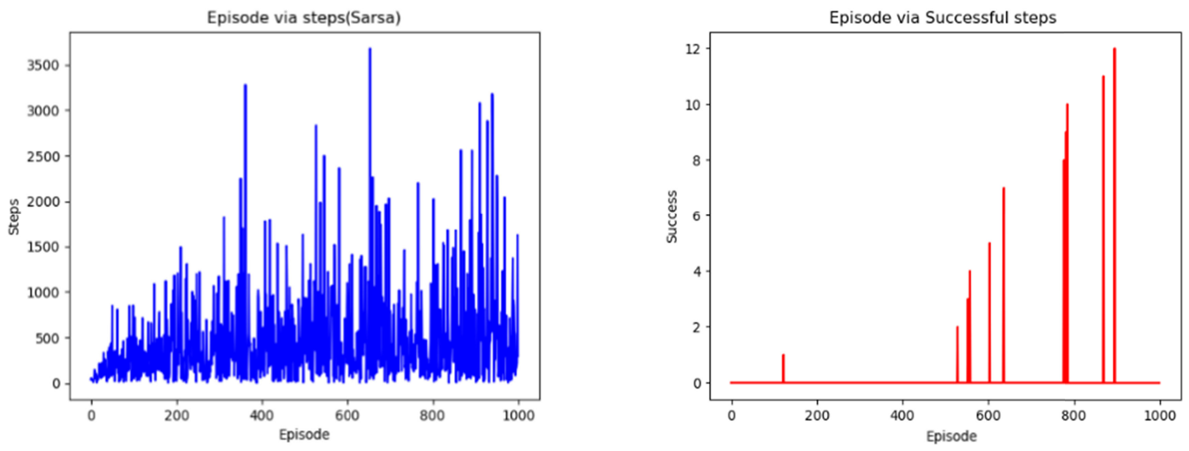

(a) Sarsa algorithm (b) Q-Learning algorithm
(c) DP-A* algorithms (d) A* algorithms

Fig. 12. Path planning results on the Map-A.

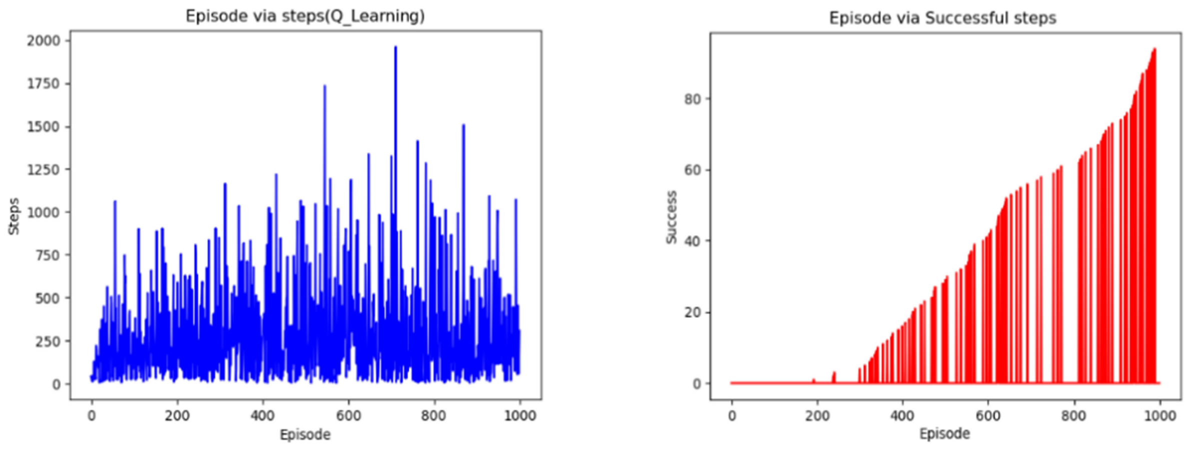
exploration experience and will calculate the weight of each node according to the A* algorithm, as shown in Fig. 11a. However, some unknown obstacles will be encountered in the actual path exploration, the UGV can bypass the obstacles to reach the target point with the help of local path planning strategy, as shown in Fig. 11b. After completing the first path exploration task, the exploration experience is obtained and recorded on Q-table, and the areas marked with purple will be deleted because Flag is true. Finally, only the state around the unknown obstacles will be stored in the Q-table, and each action that reaches the target point will receive a reward value, which can speed up the convergence of the Q-table. At the second path exploration, the UGV calculates and updates the weight of the points around the obstacle based on the accumulated experience. When encountering some new unknown obstacles, the UGV uses the global planning strategy of the DP-A* algorithm to avoid obstacles and make path planning to reach the target point, as shown in Fig. 11c and Fig. 11d.

IV. Experimental Results and Analysis

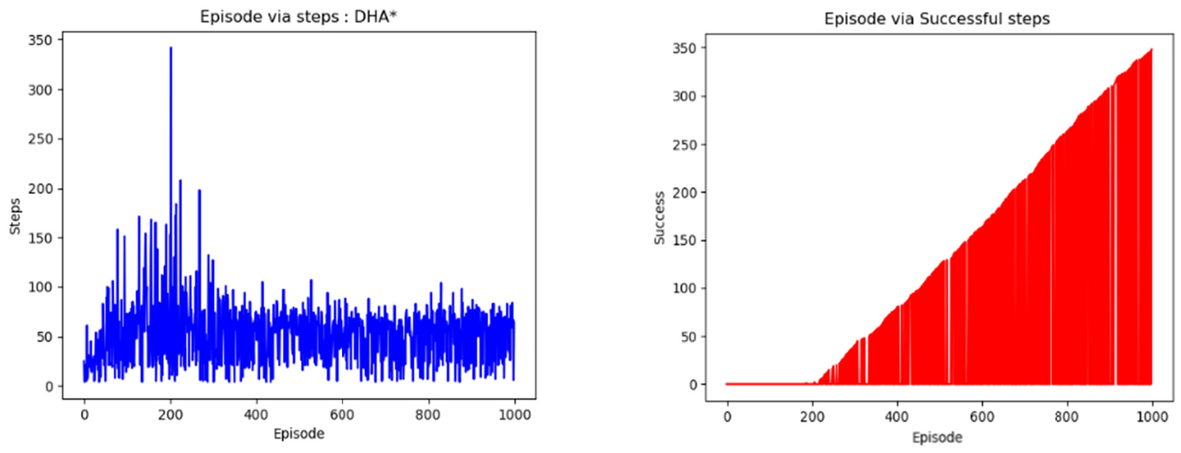
In order to simulate and test three algorithms, Q-Learning, Sarsa and DP-A*, which is conducted using three maps as shown in Fig. 14. The map scale is 25×25 in the simulation test, and some different numbers of obstacles are randomly generated on it, but their location is unknown in advance. UGV will continue to repeat the path exploration task on the map to verify and test the learning ability of the above algorithms. The path planning results within 1,000 episodes on the Map-A are shown in Fig. 12, the path length and success rate of the above three algorithms are shown in Fig. 13, and some



(a) Sarsa algorithm



(b) Q-Learning algorithm



(c) DP-A* algorithms

Fig. 13. Path length and success rate of three algorithms on the Map-A.

statistical results of three algorithms on three different maps are shown in Table II.

Fig. 12 shows the path planning results of the four algorithms after 1000 training sessions on the same map. For comparison with related algorithms in reinforcement learning, these four algorithms end tasks and restart path planning when encountering unknown obstacles. This rule excludes the effect of local planning algorithms on the result and directly compares the effect of different empirical exploration strategies on the experimental result. The experiment specified that all agents only knew the start and end coordinates. Due to the lack of extensive training, the Sarsa algorithm (Fig. 12a) and the Q-Learning algorithm (Fig. 12b) explored path results with a large number of redundant nodes. In addition, the Sarsa algorithm will have a large number of wobbly nodes in an environment with many unknown obstacles, as the online update strategy of the Sarsa algorithm during path planning will result in always trying to move away from

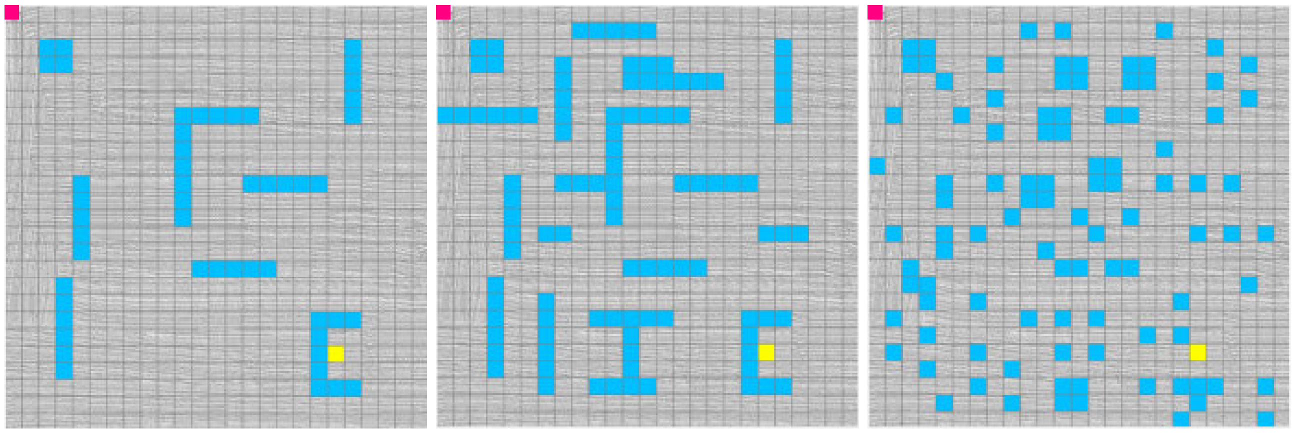

(a) Map-A (b) Map-B (c) Map-C

Fig. 14. Three maps with unknown obstacles.

TABLE II
STATISTICAL RESULTS OF THREE ALGORITHMS ON THREE DIFFERENT MAPS

Algorithms	Map	Average path length	success rate	time	Max path length	Min path length	Episodes
DP-A*		55	73.55%	90.50s	115	43	1,000
Q_learning	(A)	156	23.50%	289.33s	226	89	1,000
Sarsa		205	10.20%	210.50s	267	115	1,000
Algorithm	Map	Average path length	success rate	time	Max path length	Min path length	Episodes
DP-A*		47	86.87%	265.80s	295	43	3,000
Q_learning	(B)	179	65.55%	656.75s	895	56	3,000
Sarsa		185	60.50%	852.33s	1,026	102	3,000
Algorithm	Map	Average path length	success rate	time	Max path length	Min path length	Episodes
DP-A*		44	95.75%	480.50s	356	43	6,000
Q_learning	(C)	86	86.75%	946.57s	389	43	6,000
Sarsa		112	79.81%	1,125.24s	425	56	6,000

obstacles when making action decisions, which will make the algorithm require more training time and have a large number of redundant nodes in a narrow environment. The path planned by the conventional A* algorithm is shown in Fig. 12d. Because the A * algorithm adds a local obstacle avoidance strategy in this experiment, that is, when the UGV encounters an obstacle, the operation can return to the previous step. It can be seen that with the help of the local obstacle avoidance algorithm, the A * algorithm can reach the target point, but in a real environment with unknown obstacles, there is likely to be a local deadlock. The DP-A* algorithm can usually produce paths that are close to the global optimum for the same number of training sessions due to the guidance of dynamic heuristic functions and the influence of empirical self-optimisation strategies. The DP-A* algorithm (Fig. 12c) can usually produce paths that are close to the global optimum for the same number of training sessions due to the guidance of dynamic heuristic functions and the influence of empirical self-optimisation strategies.

As shown in Fig. 13, the left side of the figure illustrates the path length which is the execution step of reaching the end point or the step of forced termination due to falling into an infinite loop, the right side of the figure illustrates the number of successes to reach the target point within 1,000 episodes. From the length of the path to the target point and the total number of successes, it can be concluded that the Sarsa algorithm (Fig. 13a) and the Q-Learning algorithm (Fig. 13b) have a higher path length and overall success rate per exploration of the environment with a relatively small number of explorations, due to the blind and random nature of reinforcement learning algorithms in the absence of training times. As the online update strategy of the Sarsa algorithm will always directly avoid dangerous actions, the length of the Sarsa algorithm path planning is almost greater than 500 steps. The Q-Learning algorithm will have a greater probability of a full exploration mechanism for unknown environments due to the offline update strategy, which will mean that the algorithm will have a faster convergence rate for optimal path exploration. At the same time, their average path length of the Q-Learning algorithm is more than 250 steps, while that of the DP-A* algorithm (Fig. 13c) is mostly within 100 steps. Due to the blindness and randomness of learning and less training time, the success rate of Sarsa and Q-Learning is also far lower than that of the DP-A* algorithm after 200 episodes. Therefore, with the help of heuristic function and experience liberalization strategy, the DP-A* algorithm can accumulate a lot of experience in the early stage of exploring the path and achieve faster convergence.

As shown in Table II, the proposed DP-A* algorithm has better performance than the Q-Learning algorithm and the Sarsa algorithm on different maps. Firstly, the success rate of the DP-A* algorithm is higher from 73.55% to 95.75%, while that of the Sarsa algorithm is from 10.2% to 79.81%, and that of the Q-Learning algorithm is from 23.5% to 86.75%.

Secondly, the DP-A* algorithm has better stability and robustness in terms of average path, the average path of DP-A* algorithm is 55 steps within 1,000 episodes, while the Sarsa algorithm is 205 steps, and the Q-Learning algorithm is 156 steps. Finally, the DP-A* algorithm has better adaptability to complex environment than the other two algorithms, which can realize exploring maps, optimizing paths, memorizing obstacles and formulating strategies at the same time.

V. Conclusion

As UGVs become more and more popular and used in various industries, the scenarios they face will be more and more complex, and some unknown obstacles often appear on the path. Robots only rely on external sensors (such as laser radar, RGBD camera) to effectively realize local path planning and cross obstacles, which cannot ensure that the following path is globally optimal. However, the traditional A* algorithm can only perform global path planning based on raster maps and cannot optimize the path iteratively in dynamic environments with unknown obstacles, wasting system resources and time, the path may be deadlocked in the most serious case. This paper fills this research gap by proposing a new dynamic parametric A* algorithm (DP-A*) for UGVs in repetitive task environments that only rely on SLAM maps. The contribution of the algorithm consists of at least two aspects: first, the original evaluation function of the A* algorithm is modified in combination with the update strategy and learning method of reinforcement learning to remember the coordinates of unknown obstacles in the environment. Second, it uses Q-table as the auxiliary guidance of experience reuse strategy and generates decision factors to overcome the disadvantage that A* algorithm only depends on grid map to calculate node weights. Using this framework, we conducted an experiment with multiple exploration tasks to test the self-learning and path planning ability of the algorithm when encountering unknown obstacles. A series of simulations are carried out to verify the improved DP-A* algorithm, the experimental results show that the DP-A* algorithm is significantly better than the A* algorithm, the Q-Learning algorithm, and the Sarsa algorithm in terms of path exploration efficiency when encountering discrete and continuous unknown obstacles. In addition, the convergence speed of the DP-A* algorithm has been improved by about 40%, and the average path length of the DP-A* algorithm is reduced by 30% than that of other algorithms in the experiment.

In the future, we will be engaged in the following aspects:

1) Enhancement of signal transmission between UGVs. We should design a holistic strategy to coordinate multiple sensor signals with path planning algorithms and reinforcement learning. Explore the impact of reinforcement learning and neural networks on robotic action decisions using sensor data.

2) Explore more complex neural network architectures to reduce feature dimensions. In this paper, the simplest reinforcement learning algorithm is used to complete the simulation experiment. We will explore more complex neural network structures for more complex path exploration tasks, and the network models should have better portability.

References

[1] C. Richter, A. Bry, and N. Roy, “Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments,” in *Proc. Robot. Res., 16th Int. Symp. (ISRR)*, M. Inaba and P. Corke Eds. Cham, Switzerland: Springer, 2016, pp. 649–666.

[2] M. Kamel, T. Stastny, K. Alexis, and R. Siegwart, “Model predictive control for trajectory tracking of unmanned aerial vehicles using robot operating system,” in *Robot Operating System (ROS): The Complete Reference*, vol. 2, A. Koubaa, Ed. Cham, Switzerland: Springer, 2017, pp. 33–39.

[3] Z. Zhang, J. Wu, J. Dai, and C. He, “Optimal path planning with modified A-Star algorithm for stealth unmanned aerial vehicles in 3D network radar environment,” *Proc. Inst. Mech. Eng., G, J. Aerosp. Eng.*, vol. 236, no. 1, pp. 72–81, Jan. 2022.

[4] D. Lyu, Z. Chen, Z. Cai, and S. Piao, “Robot path planning by leveraging the graph-encoded Floyd algorithm,” *Future Gener. Comput. Syst.*, vol. 122, pp. 204–208, Sep. 2021.

[5] P. K. Das, H. S. Behera, and B. K. Panigrahi, “Intelligent-based multi-robot path planning inspired by improved classical Q-learning and improved particle swarm optimization with perturbed velocity,” *Eng. Sci. Technol., Int. J.*, vol. 19, no. 1, pp. 651–669, Mar. 2016.

[6] Q. Liu, Y. Zhang, M. Li, Z. Zhang, N. Cao, and J. Shang, “Multi-UAV path planning based on fusion of sparrow search algorithm and improved bioinspired neural network,” *IEEE Access*, vol. 9, pp. 124670–124681, 2021.

[7] N. Wang and H. Xu, “Dynamics-constrained global-local hybrid path planning of an autonomous surface vehicle,” *IEEE Trans. Veh. Technol.*, vol. 69, no. 7, pp. 6928–6942, Jul. 2020.

[8] M. Liu, F. Colas, F. Pomerleau, and R. Siegwart, “A Markov semi-supervised clustering approach and its application in topological map extraction,” in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2012, pp. 4743–4748.

[9] M. Sniedovich, “Dijkstra’s algorithm revisited: The dynamic programming connexion,” *Control Cybern.*, vol. 35, no. 3, pp. 599–620, 2006.

[10] J. Y. Hwang, J. S. Kim, S. S. Lim, and K. H. Park, “A fast path planning by path graph optimization,” *IEEE Trans. Syst., Man, Cybern., A, Syst. Humans*, vol. 33, no. 1, pp. 121–129, Jan. 2003.

[11] L. S. Liu et al., “Path planning for smart car based on Dijkstra algorithm and dynamic window approach,” *Wireless Commun. Mobile Comput.*, vol. 2021, Feb. 2021, Art. no. 8881684.

[12] M. Parulekar, V. Padte, T. Shah, K. Shroff, and R. Shetty, “Automatic vehicle navigation using Dijkstra’s algorithm,” in *Proc. Int. Conf. Adv. Technol. Eng. (ICATE)*, Jan. 2013, pp. 1–5, doi: 10.1109/ICAdTE.2013.6524721.

[13] A. Stentz, “Optimal and efficient path planning for partially-known environments,” in *Proc. IEEE Int. Conf. Robot. Autom.*, May 1994, pp. 3310–3317, doi: 10.1109/ROBOT.1994.351061.

[14] D. Ferguson and A. Stentz, “Using interpolation to improve path planning: The field D* algorithm,” *J. Field Robot.*, vol. 23, no. 2, pp. 79–101, Mar. 2006.

[15] A. Nash, S. Koenig, and M. Likhachev, “Incremental Phi*: Incremental any-angle path planning on grids,” in *Proc. 21st Int. Joint Conf. Artif. Intell. (IJCAI)*, 2009, pp. 1824–1830.

[16] M. Elbanhawi and M. Simic, “Sampling-based robot motion planning: A review,” *IEEE Access*, vol. 2, pp. 56–77, 2014.

[17] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566–580, Aug. 1996.

[18] S. M. LaValle and J. J. Kuffner, “Randomized kinodynamic planning,” in *Proc. IEEE Int. Conf. Robot. Autom.*, vol. 1, May 1999, pp. 473–479.

[19] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, Jan. 2011.

[20] S. Karaman and E. Frazzoli, “Optimal kinodynamic motion planning using incremental sampling-based methods,” in *Proc. 49th IEEE Conf. Decis. Control (CDC)*, Dec. 2010, pp. 7681–7687, doi: 10.1109/CDC.2010.5717430.

[21] O. Salzman and D. Halperin, “Asymptotically near-optimal RRT for fast, high-quality motion planning,” *IEEE Trans. Robot.*, vol. 32, no. 3, pp. 473–483, Jun. 2016, doi: 10.1109/TRO.2016.2539377.

[22] D. Li and J. Hu, “Mitigating motion sickness in automated vehicles with frequency-shaping approach to motion planning,” *IEEE Robot. Autom. Lett.*, vol. 6, no. 4, pp. 7714–7720, Oct. 2021, doi: 10.1109/LRA.2021.3101050.

[23] M. Abdel-Basset, R. Mohamed, S. Saber, S. Askar, and M. Abouhawwash, "Modified flower pollination algorithm for global optimization," *Mathematics*, vol. 9, no. 14, p. 1661, Jul. 2021.

[24] K. B. Lee and J. H. Kim, "Multiobjective particle swarm optimization with preference-based sort and its application to path following footstep optimization for humanoid robots," *IEEE Trans. Evol. Comput.*, vol. 17, no. 6, pp. 755–766, Dec. 2013.

[25] L. Chen, X. Hu, B. Tang, and Y. Cheng, "Conditional DQN-based motion planning with fuzzy logic for autonomous driving," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 4, pp. 2966–2977, Apr. 2022.

[26] D. Xu, Y. Fang, Z. Zhang, and Y. Meng, "Path planning method combining depth learning and sarsa algorithm," in *Proc. 10th Int. Symp. Comput. Intell. Design (ISCID)*, Dec. 2017, pp. 77–82.

[27] Y. Liu, W. Zhang, F. Chen, and J. Li, "Path planning based on improved deep deterministic policy gradient algorithm," in *Proc. IEEE 3rd Inf. Technol., Netw., Electron. Autom. Control Conf. (ITNEC)*, Mar. 2019, pp. 295–299.

[28] I. Goswami et al., "Extended Q-learning algorithm for path-planning of a mobile robot," in *Simulated Evolution and Learning*, K. Deb, A. Bhattacharya, N. Chakraborti, P. Chakroborty, S. Das, and J. Dutta, Eds. Berlin, Germany: Springer, 2010, pp. 379–383.

[29] A. Konar, I. G. Chakraborty, S. J. Singh, L. C. Jain, and A. K. Nagar, "A deterministic improved Q-learning for path planning of a mobile robot," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 43, no. 5, pp. 1141–1153, Sep. 2013.

[30] E. S. Low, P. Ong, and K. C. Cheah, "Solving the optimal path planning of a mobile robot using improved Q-learning," *Robot. Auto. Syst.*, vol. 115, pp. 143–161, May 2019.

[31] L. Yonggang et al., "A mobile robot path planning algorithm based on improved A* algorithm and dynamic window approach," *IEEE Access*, vol. 10, pp. 57736–57747, 2022.

[32] G. Dubbelman and B. Browning, "COP-SLAM: Closed-form online pose-chain optimization for visual SLAM," *IEEE Trans. Robot.*, vol. 31, no. 5, pp. 1194–1213, Oct. 2015.

[33] P.-Y. Lajoie, B. Ramtoula, Y. Chang, L. Carlone, and G. Beltrame, "DOOR-SLAM: Distributed, online, and outlier resilient SLAM for robotic teams," *IEEE Robot. Autom. Lett.*, vol. 5, no. 2, pp. 1656–1663, Apr. 2020.

[34] B. Fu et al., "An improved A* algorithm for the industrial robot path planning with high success rate and short length," *Robot. Auton. Syst.*, vol. 106, pp. 26–37, Aug. 2018.

[35] Z. Qin, X. Chen, M. Hu, L. Chen, and J. Fan, "A novel path planning methodology for automated valet parking based on directional graph search and geometry curve," *Robot. Auton. Syst.*, vol. 132, Oct. 2020, Art. no. 103606, doi: 10.1016/j.robot.2020.103606.

[36] M. Luo, X. Hou, and J. Yang, "Surface optimal path planning using an extended Dijkstra algorithm," *IEEE Access*, vol. 8, pp. 147827–147838, 2020.

[37] Q. Fan, G. Cui, Z. Zhao, and J. Shen, "Obstacle avoidance for microrobots in simulated vascular environment based on combined path planning," *IEEE Robot. Autom. Lett.*, vol. 7, no. 4, pp. 9794–9801, Oct. 2022.

[38] M. Lin, K. Yuan, C. Shi, and Y. Wang, "Path planning of mobile robot based on improved A* algorithm," in *Proc. 29th Chin. Control Decis. Conf. (CCDC)*, May 2017, pp. 3570–3576.

[39] Y. Li, J. Chang, C. Kong, and W. Bao, "Recent progress of machine learning in flow modeling and active flow control," *Chin. J. Aeronaut.*, vol. 35, no. 4, pp. 14–44, Apr. 2022.

[40] P. Ong, D. D. V. S. Chin, C. S. Ho, and C. H. Ng, "Modeling and optimization of cold extrusion process by using response surface methodology and Metaheuristic approaches," *Neural Comput. Appl.*, vol. 29, no. 11, pp. 1077–1087, Jun. 2018.

[41] X. Wang, Z. Ning, S. Guo, M. Wen, L. Guo, and H. V. Poor, "Dynamic UAV deployment for differentiated services: A multi-agent imitation learning based approach," *IEEE Trans. Mobile Comput.*, vol. 22, no. 4, pp. 2131–2146, Apr. 2023.

[42] X. Wang, Z. Ning, S. Guo, M. Wen, and H. V. Poor, "Minimizing the age-of-critical-information: An imitation learning-based scheduling approach under partial observations," *IEEE Trans. Mobile Comput.*, vol. 21, no. 9, pp. 3225–3238, Sep. 2022.

[43] Z. Ning et al., "Blockchain-enabled intelligent transportation systems: A distributed crowdsensing framework," *IEEE Trans. Mobile Comput.*, vol. 21, no. 12, pp. 4201–4217, Dec. 2022.

Xingli Gan was born in Tangshan, Hebei, China, in 1981. He received the B.S. degree in water supply and sewerage engineering and the Ph.D. degree in navigation, guidance, and control engineering from Harbin Engineering University, in 2004 and 2008, respectively. From 2008 to 2011, he was a Research Assistant with the 54th Research Institute of China Electronics Technology Group Corporation, China. Since 2009, he has been a Researcher with the State Key Laboratory of Satellite Navigation System and Equipment Technology, China. He is currently an Associate Professor with the School of Information and Electronic Engineering, Zhejiang University of Science and Technology, Zhejiang, China. He has authored one book and more than 40 articles. He has invented more than 15 inventions. His research interests include satellite navigation, pseudolites, visual positioning technology, artificial intelligence, and location service application. He is a Senior Member of the China Electronics Society. He has received the first prize (GLAC) and the provincial and ministerial awards in China.

Zhihui Huo received the B.S. degree in Internet of Things engineering from the Henan University of Urban Construction, Pingdingshan, Henan, China, in 2020. He is currently pursuing the M.S. degree with the School of Information and Electronic Engineering, Zhejiang University of Science and Technology, Hangzhou, Zhejiang. His research interests include autonomous driving of ground vehicles, decision-making, and other autonomous systems.

Wei Li was born in Qingyang, Gansu, China, in 1980. He received the master's and Ph.D. degrees from the University of Chinese Academy of Sciences in 2009 and 2014, respectively. Since 2009, he has been with the National Time Service Center, Chinese Academy of Sciences, where he is currently a Senior Engineer. As the Chief Engineer of the Joint Time-keeping Project between Space Station and Beidou System, he is responsible for the tasks of project construction, and research on the joint timekeeping, time comparison and performance evaluation of the satellite atomic clock autonomous systems. He has authored more than 15 articles. He has two standards and holds two patents. His research interests include timekeeping, time comparison, and time performance evaluation.