Received XX Month, XXXX; revised XX Month, XXXX; accepted XX Month, XXXX; Date of publication XX Month, XXXX; date of current version XX Month, XXXX.

Digital Object Identifier 10.1109/OJVT.2024.1234567

A 3D Spatial Information Compression Based Deep Reinforcement Learning Technique for UAV Path Planning in Cluttered Environments

Zhipeng Wang, Soon Xin Ng and Mohammed El-Hajjar

[1]School of Electronics and Computer Science, University of Southampton, Southampton SO17 1BJ, United Kingdom
Email:{z.wang@soton.ac.uk, sxn@ecs.soton.ac.uk, meh@ecs.soton.ac.uk }

Mohammed El-Hajjar would like to acknowledge the support of the Future Telecoms Research Hub, Platform for Driving Ultimate Connectivity (TITAN), sponsored by the Department of Science Innovation and Technology (DSIT) and the Engineering and Physical Sciences Research Council (EPSRC) under Grants EP/X04047X/1, EP/Y037243/1 and EP/X04047X/2.

ABSTRACT Unmanned aerial vehicles (UAVs) can be considered in many applications, such as wireless communication, logistics transportation, agriculture and disaster prevention. The flexible maneuverability of UAVs also means that the UAV often operates in complex 3D environments, which requires efficient and reliable path planning system support. However, as a limited resource platform, the UAV systems cannot support highly complex path planning algorithms in lots of scenarios. In this paper, we propose a 3D spatial information compression (3DSIC) based deep reinforcement learning (DRL) algorithm for UAV path planning in cluttered 3D environments. Specifically, the proposed algorithm compresses the 3D spatial information to 2D through 3DSIC, and then combines the compressed 2D environment information with the current UAV layer spatial information to train UAV agents for path planning using neural networks. Additionally, the proposed 3DSIC is a plug and use module that can be combined with various DRL frameworks such as Deep Q-Network (DQN) and deep deterministic policy gradient (DDPG). Our simulation results show that the training efficiency of 3DSIC-DQN is 4.028 times higher than that directly implementing DQN in a $100 \times 100 \times 50$ 3D cluttered environment. Furthermore, the training efficiency of 3DSIC-DDPG is 3.9 times higher than the traditional DDPG in the same environment. Moreover, 3DSIC combined with fast recurrent stochastic value gradient (FRSVG), which can be considered as the most state-of-the-art DRL algorithm for UAV path planning, exhibits 2.35 times faster training speed compared with the original FRSVG algorithm.

INDEX TERMS 3D path planning, Deep Reinforcement Learning, 3D Spatial Information Compression, Unmanned Aerial Vehicles, Training Efficiency.

I. Introduction

IN the past decade, with the rapid development of unmanned aerial vehicles (UAVs), UAVs have demonstrated their strong potential in a wide range of application fields, such as weather forecasting, low altitude economy, logistics transportation, disaster prevention and control, agriculture, safety assurance, and wireless communication [1]–[8]. Rotor-UAVs have extremely high flexibility, and can carry modules with different functions according to the application scenarios [9]. However, UAVs have clear shortcomings. The main challenge is limited energy, which leads to limited endurance of UAVs, requiring them to complete tasks in the shortest possible time. Therefore, autonomous navigation of UAVs has always been one of the main challenges in this research field.

A. Background and Motivation

At present, there is a lot of research work on autonomous navigation and path planning for UAVs, where different optimization methods are used. The A-star algorithm is a commonly used navigation and path planning algorithm for ground vehicles and various ground operation robots [10].

Although the A-star algorithm is considered one of the most effective static global path planning algorithms, its efficiency and robustness is unsatisfactory when working in complex environments [11]. The Dijkstra algorithm, similar to the A-star algorithm, is also a static global path planning algorithm. In [12], the Dijkstra algorithm was used to assist UAVs in trajectory planning, which improved the 3D deployment efficiency of UAVs. *Although the results showed that using the Dijkstra algorithm is better than using algorithms such as K-mean, the delay in path planning is still significant.*

There are other sampling based path planning algorithms for the UAV such as rapidly-exploring random tree (RRT) [13], [14], probabilistic road-map (PRM) [15], and artificial potential field (APF) [16], [17] algorithms, all of which are sampling based static global path planning algorithms that require knowledge of all environmental information. The research on path planning based on these three algorithms has produced rich results in 2D environments [16]–[20], but limited research in 3D environments [21]. The CTopPRM algorithm was proposed in [22] that can perform path planning in 3D space, and the probability of finding a path is 20% higher than traditional PRM algorithms. *However, this result cannot meet the requirement of path planning task for UAVs in 3D environments.*

In addition, some biologically inspired optimization algorithms have been used to solve path planning problems. The ant colony optimization (ACO) algorithm is a typical example, and [23] proposed an ACO-APF algorithm that improves the convergence speed of path planning and reduces the triggering probability of local optimal solutions. A 3D path planning algorithm was proposed in [24] for UAVs based on ACO and it was shown to be effective in disaster prevention and control scenarios. *However, the convergence efficiency and path planning quality of ACO algorithm are not satisfactory when facing complex environments.*

Furthermore, particle swarm optimization (PSO) has been used to solve path planning problems. For example, [36] proposed a mobile robot path planning algorithm that combines A-star and PSO, which is about 16% faster than the A-star algorithm in path planning time. A hybrid-PSO algorithm was proposed in [37], which not only reduces the possibility of falling into local optima but also improves the convergence efficiency of path planning. Additionally, it has better robustness than traditional PSO algorithms applied to UAV path planning in 3D environments. On the other hand, a genetic algorithm (GA) based UAVs 4D path planning algorithm was proposed in [38], which considers both 3D UAVs paths and multi UAV conflict problems, which shows high robustness in discrete 3D space. As mentioned in [39], a '2.5D' modeling method which can generate a very coarse 3D model based on the 2D footprint and the height of obstacles can work for 3D UAV path planning. However, when there are complex obstacles in the environment, such as overpasses and tunnels, using obstacle height to construct a 2D matrix can damage the optimal path of the UAV. In addition, this method also has limitations in complex indoor environments.

Deep reinforcement learning (DRL) is considered one of the promising algorithms for solving UAV path planning problems [40], which allows UAV agents to automatically track and learn changes in the environment, while some classic path planning algorithms cannot plan paths in dynamic environments. Among all the DRL algorithms, Deep Q-Network (DQN), actor-critic (AC), and deep deterministic policy gradient (DDPG) are three commonly used algorithms for solving UAV path planning problems, especially when path planning tasks are considered as Markov decision process (MDP). An improved DQN algorithm was proposed in [41], which introduced an empirical value evaluation network to effectively eliminate path planning drift and improve the accuracy of path planning. The introduction of experience replay [42] has improved the network convergence stability of DRL algorithms such as DQN, DDPG, and AC, while introducing the problem of key experience sparsity when using DRL algorithms for path planning tasks [43]–[45]. A cumulative reward model was proposed in [32] to reduce network divergence caused by sparse rewards, and also introduced a region segmentation algorithm to further reduce the possibility of intelligent agents trained by multiple DRL algorithms falling into local optima. Besides, some works combined DRL algorithms with sampling based algorithms to achieve better path planning performance and higher training efficiency. For example, bidirectional artificial potential field deep Q-network (B-APFDQN) was proposed in [46], which combined APF and DQN to improve the training efficiency of autonomous UAV agents.

Although the DRL algorithm is powerful, the training process consumes a lot of time and computing resources, which can be feasible in a two-dimensional environment. However, when the environment expands to three dimensions, the search space rapidly increases, the consumption of time and the demand for computing resources can not be ignored. Researchers have made great efforts in the path planning problem of UAVs in 3D environments. For example, [25] proposed an improved sparse A-star algorithm, which has been proven to be capable of 3D path planning for UAVs. A 3D tangent (3D-TG) method based on obstacle geometry information was proposed in [26], which can complete UAV path planning in a 3D urban environment. In [30], a high efficient state decomposition deep deterministic strategy gradient algorithm has been proposed, which has improved convergence rate, navigation performance, and generalization ability compared to traditional DDPG algorithms. In [35], the authors proposed a fast recurrent stochastic value gradient (FRSVG) that can perform path planning in continuous 3D environments. The experimental results show that FRSVG has higher training efficiency and path planning performance compared to DDPG and traditional SVG algorithms.

In the UAV 3D path planning, the problem of large search space and sparse data remains a challenge. If effective

TABLE 1: Novelty comparison with the state-of-the-art literature.

	our paper	[25]	[26]	[27]	[28]	[29]	[30]	[31]	[32]	[33]	[34]	[35]
Deep Q-Network	✓			✓					✓			
Deep Deterministic Policy Gradient	✓					✓	✓		✓	✓		✓
Information Compression	✓		✓									
3D environment	✓	✓	✓		✓	✓	✓	✓		✓	✓	✓
Continuous Workspace	✓		✓		✓	✓	✓	✓	✓	✓	✓	✓

dimensionality reduction can be applied to 3D spatial information, it can greatly reduce the search space and alleviate various problems caused by data sparsity. Principal component analysis (PCA) is a commonly used data dimensionality reduction method in machine learning and related fields, which tunes the Rayleigh quotient to find a suitable projection to reduce the dimensionality of the dataset [47]. The PCA algorithm has shown good performance in classification and regression problems [48]. For example, [49] uses PCA to classify ground targets, and [50] uses PCA algorithm to predict the trajectory of UAVs. However, when using DRL technology to solve path planning problems, the data learned by the neural network is the real experience obtained by the UAV agent through interaction with the simulator or the environment, which is difficult to linearly compress through PCA algorithm. A more reasonable approach is to reduce the size of the search space to obtain more dense and effective data for training neural networks. However, the PCA algorithm cannot accomplish this task. Similarly, nonlinear manifold dimensionality reduction methods such as Locally Linear Embedding (LLE) and t-Distributed Stochastic Neighbor Embedding (t-SNE) [51] are also unable to reduce the dimensionality of 3D environmental information. Moreover, the dimension of the experiences obtained by the interaction between UAV agents and environment can not be reduced by either PCA nor t-SNE. *At the time of writing this manuscript, research on the path planning problem of UAVs in 3D environments is still very limited and mainly focus on improving algorithms rather than information compression.*

In this work, we propose a 3D spatial information compression (3DSIC) method that utilizes the topological features of spatial information for compression and simplification, reducing information redundancy while retaining key information. This simplifies the navigation problem in 3D space into a 2D path planning problem.

B. Contributions

Our contributions can be summarized as follows:

- We propose a 3DSIC method to compress the 3D spatial information into 2D while retaining important environmental information, which can greatly reduce the redundancy of environmental information and the search space for path planning problems.
- We propose the 3DSIC-DQN algorithm, which combines 3DSIC and DQN algorithm. During training, UAV agents consider both compressed 3D spatial information and obstacle distribution information at the current flight altitude, enabling efficient path planning training in a cluttered 3D space.
- We consider the path planning training in cluttered continuous 3D MDP environments, then extend 3DSIC to continuous environments, and combine it with DDPG algorithm to achieve efficient path planning training algorithms in complex continuous 3D MDP environments.
- The simulation results show that the training efficiency of our proposed 3DSIC algorithm to train DQN agents is nearly four times higher than using advanced B-APFDQN algorithms.

The contributions of our paper are compared to the literature in Table 1. The rest of the paper is organized as follows. In Section II we introduce the system model. In Section III, we introduce the 3DSIC algorithm, 3DSIC-DQN and 3DSIC-DDPG algorithms. In Section IV, we demonstrate the feasibility of our proposed method through simulation results and compare with the traditional DRL techniques. In Section V, we offer our conclusions and propose potential future directions.

II. System Model and Problem Formulation

A. System model

In our proposed system, when the deployment point of the UAV is given, the UAV will plan a collision free path to move from its current position to the deployment point. Fig. 1 shows a UAV workspace modeled using the VRSink module of Simulink in Matlab. The yellow dot represents the UAV's destination, while the orange rectangular objects represent obstacles distributed in 3D space. The UAV's path planning task is to plan a path as short as possible, so that the UAV can fly from its current position to the destination marked in yellow in that 3D space. In this paper, we consider the UAV path planning problem as a MDP and consider the situation in both discrete and continuous 3D environments. A discrete 3D environment is a 3D mesh based on division of a real 3D working environment, dividing space into many fixed sized cube nodes. The motion of UAVs is seen as a

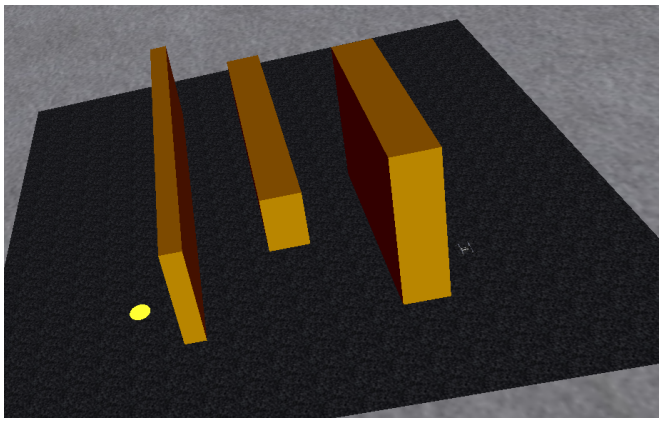

FIGURE 1: 3D view of UAV working environment in Simulink VRSink.

state transition process within these cube nodes. Each cube node has its 3D coordinates and sorting number. In order to simplify the motion process of the UAV, we only consider the motion between adjacent nodes, that is the UAV motion space $\mathbf{A} = a_t = \{\text{Forward, backward, left, right, up, down}\}$.

The continuous 3D space directly uses the 3D workspace as the emulator model, where the UAV directly uses the current 3D coordinates to represent the state $S_t(x_t, y_t, z_t)$ and complete interaction with the entire state space, where x_t,y_t and z_t are the 3D coordinates at time t. The motion and flight control in continuous 3D space are very complex and not the focus of this study. Therefore, horizontal flight speed, horizontal heading angle, and vertical flight speed are used to represent the action space $a(V_t^h, V_t^v, \vartheta_t)$, where V_t^h is horizontal flight speed and its value is limited to (0,2m/s), V_t^v is vertical flight speed and have 3 option values [-1m/s, 0, 1m/s], ϑ_t is horizontal heading angle satisfy $\vartheta_t \in (0, 2\pi)$ [35]. Hence, the state update of UAVs in a continuous 3D environment can be represented by the following formula:

$$\begin{cases} x_{t+1} = x_t + V_{t+1}^h \times \cos(\vartheta_{t+1}) \times \Delta t \\ y_{t+1} = y_t + V_{t+1}^h \times \sin(\vartheta_{t+1}) \times \Delta t \\ z_{t+1} = z_t + V_{t+1}^v \times \Delta t \end{cases} . \quad (1)$$

B. Problem Formulation

In this 3D simulation environment, the path planning result of UAV can be expressed as an ordered path vector $\boldsymbol{P} = [P_1, P_2, \ldots, P_{last}]$, where P_i represents the i-th node in the path, P_{last} represents the last node in the path. The ordered vector $\boldsymbol{P}$ satisfies the condition $P_1 = S_{start}, P_{last} = S_{end}, \forall \hat{p} \subseteq \boldsymbol{P}$ satisfies $\hat{p} \in \boldsymbol{S}, \hat{p} \notin \boldsymbol{O}$, where $\boldsymbol{S}$ is the set of all possible states, S_{start} is the start node, S_{end} is the destination node, and $\boldsymbol{O}$ is the set of all obstacle nodes.

1) Path Planning in discrete space

Explicitly, we consider the path planning problem of UAV as a MDP, which includes several main parts: state set S, action set $\mathbf{A} = a_t = \{\text{Forward, backward, left, right, up, down}\}$, transfer function $T(s'|s, a)$, and reward function $R(s'|s, a)$. The main advantage of Q-learning is that it uses the time differential (TD) method (a combination of Monte Carlo and dynamic programming) for off-line learning, and Bellman equation can be used to solve the optimal strategy of the Markov process [52], [53].

The solution of the Bellman equation is based on the optimal cumulative expectation $V^*(s)$, which can be expressed as follows [54]:

$$V^*(s) = \max_{\pi} E\left[\sum_{t=0}^{t=n} \gamma^t R(S_{t+1}, A_t, S_t|\pi, s)\right], \quad (2)$$

where γ is the discount factor, t is current time index, π is the action policy which can be presented as the probability of executing action a at state s. The Q value $Q(s, a)$ is a function of the action state value, expressed as the expected accumulation of the action reward value as follows:

$$Q(s, a) = E[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots |a, s)], \quad (3)$$

where r_{t+n} is the reward value of next n steps of the current time index t. The update of the Q value is based on the Bellman's equation:

$$\begin{aligned} Q(s_{t+1}, a_{t+1}) = (1 - \alpha)Q(s_t, a_t) + \\ \alpha[R(s_t, a_t) + \gamma \max_a(Q(s_{t+1}, a))], \end{aligned} \quad (4)$$

where α is the learning rate of the agent. If the agent executes action a_t in state s_t at time t, it will get immediate reward $R(s_t, a_t)$ and delayed reward $\max_a(Q(s_{t+1}, a))$. It is not difficult to see that Q-learning considers not only the current reward expectation, but also the maximum reward that may be obtained in the future when making decisions according to the Q value [55].

However, there is an obvious limitation of using tables to store Q values. That is, when the working area expands or the motion freedom of the agent increases, the table size will grow exponentially. In order to overcome this problem, DQN has been proposed, which uses deep neural network to map the relationship between input information and Q value [56]. This method has many advantages, such as making Q-learning work in a high-dimensional and complex environment without worrying about the size of the Q table. On the other hand, it can adapt to various input signals, which improves the flexibility of the system [57].

Furthermore, we need a loss function to guide the adjustment of neural network weight. In DQN, the loss function can be expressed as the expectation of the square of the difference between the observed value (real value) and the network predicted value:

$$\begin{aligned} L_i(\theta_i) = \Big((R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a'|\theta^-) - \\ Q(S_t, A_t|\theta)\Big)^2, \end{aligned} \quad (5)$$

where θ is the parameter set of the agent's network and it represents the agent's policy of action. DQN uses a target network, which has parameters from several steps earlier.

A target network is used to estimate Q-values at the step $t+1$. Furthermore, $(R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a'|\theta^-))$ is the observed Q value and $Q(S_t, A_t|\theta)$ is the predicted Q value of the network [58]. With the loss function, we only need to use the gradient descent method to continuously update the network weight to achieve the purpose of training [27]. In a discrete MDP environment, both the state space and action space are discrete, and the iteration index of the training process is a discrete time point. At each discrete time point, the state s_t and upcoming actions a_t of the UAV can be obtained.

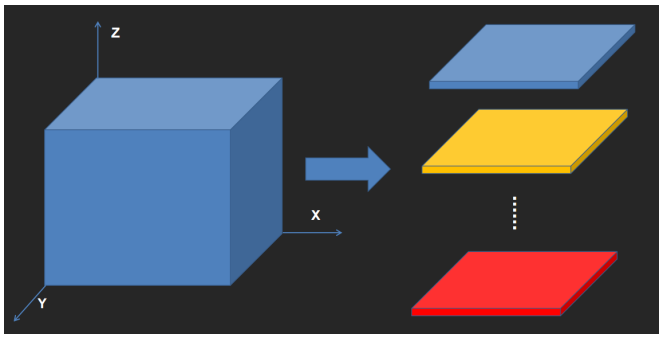

FIGURE 2: Schematic diagram of 3D space cutting.

2) Path Planning in continuous space

However, in a continuous 3D environment, the DQN algorithm faces a fatal problem. It is not difficult to see from (5) that the calculation of loss requires finding the maximum Q value among many actions, but the continuous action space is infinite and it is extremely hard to find the maximum value. The DDPG algorithm uses the Q function to replace the process of finding the maximum value in the action space:

$$\max_a(Q(s_t, a)) \rightarrow Q(s, \mu(s|\theta)), \tag{6}$$

where μ is the actor network and θ is the network weight. There are four neural networks in the DDPG algorithm, μ, μ', Q and Q'. Among them, μ and μ' are actor network and target actor network that output actions through input states, while Q, and Q' are critic network and target critic network that output evaluation value results of state transitions, which is Q values. In the DDPG algorithm, the random exploration of UAV agents is achieved by adding noise to the actions output by the actor network:

$$a_t = \mu(s_t|\theta^\mu) + \mathcal{N}, \tag{7}$$

where $\mathcal{N}$ is the noise, and θ^μ is the network weight for actor network μ. The update of the critic network is similar to the DQN network, using mean square error as the loss function:

$$L = \frac{1}{N}\sum_i \left(y_i - Q\left(s_i, a_i|\theta^Q\right)\right)^2, \tag{8}$$

where y_i is the actual Q value of s_i and a_i, $Q(s_i, a_i|\theta^Q)$ is the output of critic network. The update of the target network can be completed through soft updates after a certain delay. Similar to a 3D discrete environment, in a 3D continuous environment, although the state space and action space are continuous, we can still sample on continuous time line and determine the corresponding UAV states (3D coordinates) and actions to be executed (velocity and heading angle) at these time points.

III. Proposed Information Compression and Path Planning Techniques

In this section, our proposed 3DSIC algorithm will be introduced in detail. Furthermore, the 3DSIC-DQN algorithm will be introduced in detail as a scheme for applying the 3DSIC algorithm in discrete 3D space, while the 3DSIC-DDPG algorithm will be introduced as a scheme for applying the 3DSIC algorithm in continuous 3D space.

A. 3D Spatial Information Compression

When conducting path planning in a 3D space, the search space is too large, and the spatial information that is far from the current position of the UAV, cannot bring significant benefits to the current action decision. In addition, changes in altitude during UAV flight require more energy consumption than horizontal flight, so in a good action strategy UAVs will not change flight altitude unless absolutely necessary. In this consideration, the spatial information of the layers far away from the current flight layers could be the redundant information which lead to a large and complicated search space. Therefore, we can only focus on the spatial information of several layers adjacent to the current flight altitude of the UAV and compress it into 2D as prior information to reduce the complexity of the UAV path planning.

Based on the above analysis, we believe that it is completely feasible to compress 3D space information in 3D space, and the compressed information can also meet the requirements of collision free path planning for UAVs in 3D space.

Hence, we propose the following 3D spatial information compression method. The spatial information required in the path planning process in a 3D environment includes the location of the destination and the distribution of obstacles in space. For the meshed 3D space, the spatial information is first set to binary, where the non obstacle node is set to 0, and the obstacle node is set to 1. Then we cut the 3D space in the direction perpendicular to the Z axis to obtain multiple 2D planes with the same size as shown in Fig. 2. It is worth noting that the thickness cut along the Z direction is a predetermined value, and the Z-direction dimension of this 3D environment is an integer multiple of the cutting thickness. The cutting thickness can vary in different 3D environments. Generally, the more cluttered the 3D space, the smaller the required cutting thickness value, but it should not be smaller than the size of a UAV in the real world.

Each of the resultant two dimensional planes represents the distribution of obstacles at its height. The spatial information of these planes is two-dimensional, where the $n \times n$ two-dimensional plane space information as shown in Fig. 3 can be expressed as:

$$\mathbf{S}_i = \begin{pmatrix} S_{i11} & \dots & S_{i1n} \\ S_{i21} & \dots & S_{i2n} \\ \cdot & \cdot & \cdot \\ S_{in1} & \dots & S_{inn} \end{pmatrix}, i \in (1, 2, \dots, n), \tag{9}$$

where i is the number of the plane, S_{i11} presents the cube node of plane i, $X = 1$ and $Y = 1$ is free space or obstacle, if the value of S_{i11} equals to 1 means this node is obstacle and if the value of S_{i11} equals to 0 that means node S_{i11} is free space. Then, we can reduce the dimension of 2D information of each plane, and convert these 2D information into row vectors as shown in the following:

$$\mathbf{S}_i = \begin{pmatrix} S_{i11} & \dots & S_{i1n} \\ S_{i21} & \dots & S_{i2n} \\ \cdot & \cdot & \cdot \\ S_{in1} & \dots & S_{inn} \end{pmatrix} \Rightarrow \begin{pmatrix} S_{i1} & \dots & S_{in} & \dots & S_{in^2} \end{pmatrix}. \tag{10}$$

In Fig. 4 we consider an example having a size of $n \times n \times n$ in the 3D space, which is a special case of the more general case presented previously with 3D space of size $m \times n \times h$, where m, n and h are the length, width, and height of the space. Hence, we can combine the spatial information of each 2D plane after dimension reduction to obtain the 3D spatial information $\mathbf{S}$, which can be expressed as:

$$\mathbf{S} = \begin{pmatrix} S_{1,1} & \dots & S_{1,n} & \dots & S_{1,mn} \\ S_{2,1} & \dots & S_{2,n} & \dots & S_{2,mn} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ S_{h,1} & \dots & S_{h,n} & \dots & S_{h,mn} \end{pmatrix}. \tag{11}$$

Here, the reduced dimension 3D information has not been compressed. If we conduct information compression now, the compressed information will contains the obstacle distribution of all flight height. When planning the path of UAV, we only want to consider the obstacle distribution information of the layers adjacent to the current flight altitude layer of the UAV. Therefore, we need to introduce a compression field to limit the range of 3DSIC method.

Since the spatial information meshed is discrete, the compressed field also needs to be discrete. For example, if we only consider the current layer of the UAV flight altitude and the upper and lower 2 layers adjacent to the current layer, the compression field will have 5 elements. In the section IV, we will study the effect of number of layers to compress. The compression field can be expressed as a row vector β, whose dimension is determined by the number of layers to be compressed. For example, with 5 layers compression, β can be represented as:

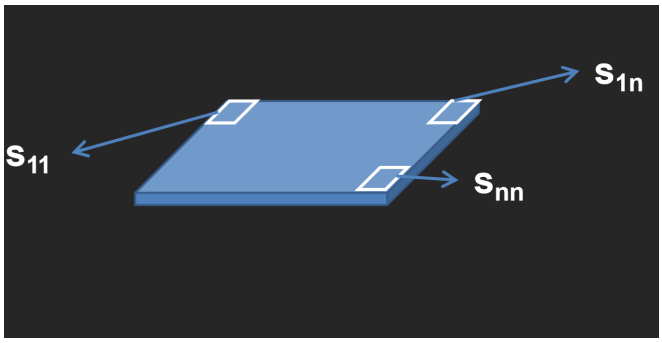

FIGURE 3: Spatial information representation of 2D plane.

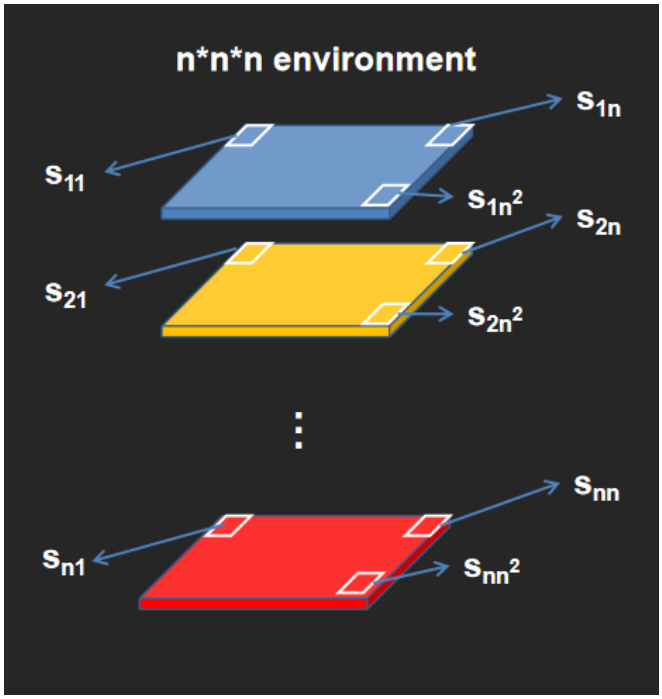

FIGURE 4: $n \times n \times n$ 3D environment information dimension reduction process.

$$\beta = \begin{pmatrix} \rho_1 & \rho_2 & \rho_3 & \rho_4 & \rho_5 \end{pmatrix}, 0 \le \rho_i \le 1, \sum_{i=1}^{5} \rho_i = 1, \tag{12}$$

where ρ_i is the compression coefficient of the i-th layer from bottom to top in the compression field.

If we do the dot product operation for the compression factor β and the spatial information of the corresponding layer after dimension reduction S, we can get:

$$\begin{aligned} \mathbf{S}_c &= \beta \cdot \mathbf{S} \\ &= \begin{pmatrix} \rho_1 & \dots & \rho_5 \end{pmatrix} \cdot \begin{pmatrix} S_{11} & \dots & S_{1n} & \dots & S_{1n^2} \\ S_{21} & \dots & S_{2n} & \dots & S_{2n^2} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ S_{51} & \dots & S_{5n} & \dots & S_{5n^2} \end{pmatrix}, \\ &= \begin{pmatrix} S_c(1) & S_c(2) & \dots & S_c(n) & \dots & S_c(n^2) \end{pmatrix} \end{aligned} \tag{13}$$

where $\mathbf{S}_c(i)$ is the compressed spatial information element. For the example of compressing 5 layers, it can be expressed as:

$$S_c(i) = \rho_1 \times S_{1i} + \rho_2 \times S_{2i} + \rho_3 \times S_{3i} + \rho_4 \times S_{4i} + \rho_5 \times S_{5i}, i \in (1, 2, \ldots, n^2). \quad (14)$$

Here, if the compressed row vector is restored to two-dimensional plane information, the two-dimensional plane information is 3D spatial information weighted by the compression factor weight of the current layer and the adjacent upper and lower four layers of spatial information. The design of the compression factor value can be based on any prior information or human experience. For example, based on the principle of not changing the flight altitude unnecessarily, the current layer has a certain weight w, while the adjacent upper and lower layers only have half of the weight of the current layer, which is $0.5w$. Following this rule, the flight altitude layer further away only has a weight of $0.25w$, and then all layers to be compressed are normalized. Taking 5-layer compression as an example, $\beta = 0.1, 0.2, 0.4, 0.2, 0.1$. Additionally, the compression factor can also be designed based on other considerations such as energy loss, expectation of cost, etc.

The physical meaning of 3DSIC can be explained as follow: if the value of a node is not 1, it means that there must be a layer in the 3D space adjacent to the current layer that can pass through the node without collision. If a node value is 1, it means that the current layer and adjacent layers cannot pass through the node.

Theoretically, the compressed spatial information can meet the requirements of collision free path planning of UAVs in the 3D space. The size of the search space is reduced from $m \times n \times h$ to $m \times n$. In a 3D environment with a compression factor β size of k and an environment of $m \times n \times h$, the compression process involves multiplying a matrix of 1 row and k columns by a matrix of k rows and $m \times n$ columns. Therefore, the time complexity is $O(k, m, n) = k \times m \times n$. In actual training, the size k of β is often less than 10, the values of m and n will not be very large either as discussed in Section IV. Explicitly, in the most complex scenario we discussed $k = 7$, $m = 100$ and $n = 100$, the time complexity $O(7, 100, 100) = 7 \times 10^4$. Similarly, the space complexity of this algorithm is low, where during the execution process, only one matrix multiplication calculation space needs to be pre-allocated. Taking the compression factor of size k for compressing environmental information on the xy plane with size $m \times n$ as an example, $S(k, m, n) = k^2 + (2 \times k + 1) \times m \times n$.

B. 3DSIC-DQN in Discrete Environment

The 3D space to be compressed is the space that the UAV agent has explored rather than the real world environment. Besides, the space that has not been detected by the UAV can be assumed to be free space. The UAV agent update the space information and obtain the reward value through the interaction with the real world environment. The output of the 3DSIC algorithm is a 2D normalized map. This map is discrete, so we can directly combine 3DSIC and DQN algorithms to use the compressed normalized discrete map as the weight coefficients for the DQN reward model and penalty value model. Therefore, the final reward value for each state transition can be expressed as:

$$R(s_t, a_t, s_{t+1}) = (1 - C(s_{t+1})) \times r(s_t, a_t) + C(s_{t+1}) \times (p_e(a_t) + p_{col}(s_{t+1})), \quad (15)$$

where $R(s_t, a_t, s_{t+1})$ is the final reward value of transition (s_t, a_t, s_{t+1}, R), which will be fed to the neural network for training, $C(s_{t+1})$ is the value at position s_{t+1} in the compressed normalized discrete map, $r(s_t, a_t)$ is the reward value given by the reward model, $p_e(a_t)$ is the motion energy cost punishment of action a_t, and $p_{col}(s_{t+1})$ is the collision punishment of position s_{t+1} in the original 3D map. Generally, the collision punishment can be set as a very large negative value. The pseudo-code process of 3DSIC-DQN is shown in Algorithm 1. The initialization part of the 3DSIC-DQN algorithm is similar to traditional DQN, with the only difference being that it needs to compress spatial information using the 3DSIC algorithm at the beginning of each episode to obtain an initialized 2D map. In addition, during the training process, after each action is executed, it is necessary to first detect whether the current flight altitude has changed. If there is a change, the compressed 2D map needs to be updated through the 3DSIC algorithm. Then, the reward is evaluated and the state transition experience is generated.

C. 3DSIC-DDPG in Continuous Environment

In a 3D continuous environment, we can also divide the 3D environment into many equally sized cube nodes, and then use these nodes to compress 3D spatial information. However, since the workspace and action space are both continuous, the coordinates of the UAV may not be integers. Therefore, for each node, there is a corresponding range $C_{range}(x)$ and $C_{range}(y)$, and when the current coordinates of the UAV are in the range of node (i, j), that is when $x_t \in C_{range}(x_i)$ and $y_t \in C_{range}(y_j)$, the UAV is considered to be in node cube (i, j). It is worth noting that we need to set the duration Δt of each vertical action to precisely allow the UAV to switch altitude to the integer z index of the discrete 3D environment, which can be presented as $V^h_{t+1} \times \Delta t = \Delta z_{index}$. The pseudo-code process of 3DSIC-DDPG is shown in Algorithm 2. It is worth noting that in a continuous 3D MDP environment, each $C(\cdot)$ records compressed value of each node and a specific range constraint for each node, so the node value $C(s)$ can be determined by the x and y coordinates of s.

IV. Simulation Results

In this section, computer simulations are used to validate, analyze and discuss the algorithms proposed in this paper.

Algorithm 1 Deep Q-Network algorithm based on 3D spatial information compression in discrete 3D MDP environment

Initialize experience replay memory $\mathcal{M}$ to capacity η.
Initialize the network weight randomly to θ.
for episode = 1 to M, **do**
  Initialize sequence $s_1 = x_1$ and prepossessed sequence $\phi_1 = \phi(s_1)$
  Initialize compressed normalized map by execute 3DSIC algorithm with β and $s_1(z)$.
  for t = 1 to T, **do**
    According to the value of ϵ, choose the random exploration strategy or the current network output $A(\theta)$ to determine the current action.
    if $s_{t+1}(z) \neq s_t(z)$ **then**
      Update compressed normalized map by execute 3DSIC algorithm with β and $s_{t+1}(z)$.
    end if
    Execute the action and calculate the reward value $R(s_t, a_t, s_{t+1}) = (1 - C(s_{t+1})) \times r(s_t, a_t) + C(s_{t+1}) \times (p_e(a_t) + p_{col}(s_{t+1}))$.
    Save the transition set (s_t, a_t, R_t, s_{t+1}) into $\mathcal{M}$.
    Sample random mini-batch I from $\mathcal{M}$ and do gradient descent of (5) to update network weight θ.
  end for
end for

Algorithm 2 DDPG algorithm based on 3D spatial information compression in continuous 3D MDP environment

Initialize experience replay memory $\mathcal{M}$ to capacity η.
Randomly initialize the actor network $\mu(s|\theta^{\mu})$ with weight θ^{μ}.
Randomly initialize the critic network $Q(s, a|\theta^{Q})$ with weight θ^{Q}.
Copy networks μ and Q to get target networks $\mu^{'}$ and $Q^{'}$.
for episode = 1 to M, **do**
  Initialize a random process $\mathcal{N}$ for action exploration.
  Initialize compressed normalized map by execute 3DSIC algorithm with β and $s_1(z)$.
  for t = 1 to T, **do**
    Select action $a_t = \mu(s_t|\theta^{\mu}) + \mathcal{N}_t$ and execute a_t to get state s_{t+1}.
    if $s_{t+1}(z) \neq s_t(z)$ **then**
      Update compressed normalized map by execute 3DSIC algorithm with β and $s_{t+1}(z)$.
    end if
    Calculate reward $R_t = (1 - C(i,j)) \times r(s_t, a_t) + C(i,j) \times (p_e(a_t) + p_{col}(s_{t+1})), x_{t+1} \in C_{range}(x_i)$ and $y_{t+1} \in C_{range}(y_j)$
    Save the transition set (s_{t+1}, h_t, a_t, R_t) into $\mathcal{M}$.
    Sample random mini-batch I from $\mathcal{M}$
    Set $y_i = R_i + \gamma Q^{'}(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$.
    Update critic network by minimizing the loss: $L = \frac{1}{N}\sum_i \left(y_i - Q\left(s_i, a_i|\theta^{Q}\right)\right)^2$
    Update actor network using sampled policy gradient: $\nabla_{\theta^u} J = \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^{Q})|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{s_i}$
    Update target networks after K steps:
    $\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$
    $\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$
  end for
end for

The simulation experiments are divided into three parts. The first part uses the B-APFDQN [45], which is a high performance DQN algorithm that can do path planning tasks in 3D environment, DDPG and fast recurrent stochastic value gradient (FRSVG) [35]. The FRSVG technique is an advanced value gradient based DRL technique proposed in [35], where detailed pseudo code is presented. Our simulations use these three techniques combined with 3D spatial information compression to train UAV agent for 20 runs in a $100 \times 100 \times 50$ 3D cluttered urban environment, where we compare the training efficiency with those DRL algorithms without 3DSIC in the same 3D environment. The second part is the simulation of collision free path planning rate of DQN, DDPG and FRSVG with and without 3DSIC algorithm. The third part is the simulation of 3DSIC-DQN UAV agents training with different size of compression factor β, corresponding to compressing different number of layers.

A. Training Efficiency of 3DSIC

We consider the scenario of UAV deployment in a real urban environment as shown in the satellite image of Fig. 5. In this scenario, the UAV performs the path planning task in the central area of the city with buildings of different heights. We chose an area near the Museum of London and used a shadow index algorithm [58] to estimate the height of the building in the environment through satellite images, where the size of this area is measured as 200 meters $\times$ 200 meters. While the size of UAV should not exceed the size of discrete space node, the rotor UAV generally has a size smaller than 2 meters $\times$ 2 meters $\times$ 1 meter, and the height of buildings do not exceed 50 meters. Based on the analysis above, we discretize the continuous environment model to a $100 \times 100 \times 50$ discrete 3D environment. In this environment, the fixed flight altitude of the UAV is set at 20 meters above the ground, which is about 6 floors high. In the environment shown in Fig. 5, the UAV will start from the area with dense buildings in the top left corner and plan a collision free path to the target point in the bottom right corner of the map.

After being processed and modeled by the shadow index algorithm of [59], the working environment in the simulator is shown in Fig. 6, which is the 3D environment modeled based on the distribution of obstacles and the height of each obstacles estimated by shadow index algorithm. The 3D MDP environment for B-APFDQN is a grid world generated

FIGURE 5: Satellite image of the real experimental environment.

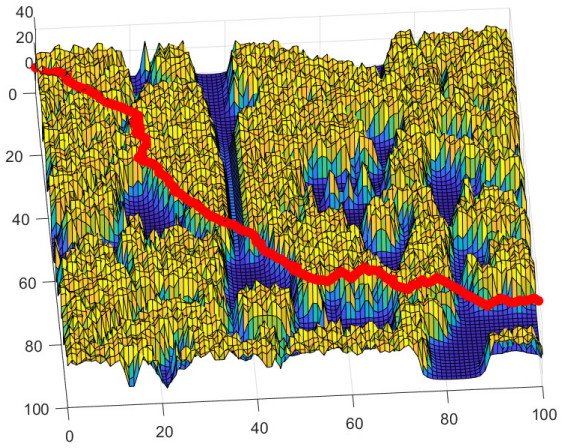

FIGURE 6: Path planning results of UAV in a simulator environment.

based on the binary map, while the 3D environment for DDPG and FRSVG is a continuous numerical space based on the binary map. After the UAV is trained by B-APFDQN, DDPG and FRSVG algorithms based on the 3DSIC, it can quickly plan a collision free flight path. The path planning results in the simulator are also shown in Fig. 6, where the red trajectory is a path planning result of the UAV agent, where the surface shows the distribution of buildings.

The hyper-parameters of B-APFDQN in the experiment are shown in Table 2, where we set the size of the mini-batch for every single training to double of the environment size index N, and the size of the experience pool is set to 100000 times of the environment size index N to ensure the experience can include sufficient transition experiences of different episodes. We change the frequency of ϵ value reduction to one reduction per episode rather than one

TABLE 2: Hyper-parameters of Deep Q-network.

Parameter	Value	Definition
Mini-batch	$2 \times N$	The sampling size of each training, N is the size of $N \times N$ environment.
γ	0.95	Discount factor
η	$N \times 100000$	Capacity of experience pool, N is the size of $N \times N$ environment.
ϵ	0.9:0.01:0.1	Possibility to select random actions, The initial value is 0.9, and the minimum value is 0.1. The decline step of each episode is 0.01.

reduction after a certain number of agent exploration steps, which will be 0.9 initially, with a decrease of 0.01 for each episode, until it dropped to 0.1. Such changes are conducive to the exploration of agents in large scale environments.

In this experiment, the network for DQN has four inputs, which include the 3D coordinates and previous action executed, and has one output of action, while employing two fully connected hidden layers, while each layer contains 128 neurons. The activation function uses rectified linear unit (ReLu), where the actor neural network structure of 3DSIC-DQN and B-APFDQN is shown in Fig. 7. On the other hand, the neural network for 3DSIC-DDPG and FRSVG has the same structure, which has 3 inputs including the 3D coordinates and it has 3 outputs corresponding to the vertical velocity, the horizontal velocity and the horizontal direction angle, with two fully connected hidden layers, where each layer contains 256 neurons. The actor neural network structure of DDPG and FRSVG is shown in Fig. 8.

Considering the fairness of experimental comparison, we adopted DNN as the hidden layer, which allows for the use of exactly the same neural network structure before and after spatial information compression, thus eliminating the performance and training efficiency impact caused by differences in neural network structure. The hyper-parameters of the neural network training for DDPG and FRSVG are shown in Table 3. For the energy consumption of the UAV, [60] derived a simplified energy consumption model for UAVs, where the energy consumption of UAVs vertical motion can be expressed as:

$$P_{vertical}(t) = P_0\left(1+\frac{3\|\mathbf{V}_i+\mathbf{a}t\|^2}{U_{tip}^2}\right) + P_1\left(\sqrt{1+\frac{\|\mathbf{V}_i+\mathbf{a}t\|^4}{4v_0^4}}-\frac{\|\mathbf{V}_i+\mathbf{a}t\|^2}{2v_0^2}\right)^{\frac{1}{2}},$$

where $\mathbf{V}_i$ is the initial velocity and a is the acceleration or deceleration, P_0 and P_1 are functions of speed related to the physical properties of the UAV and the flight environment, the tip speed of the rotor blade U_{tip} and the mean rotor induced velocity v_0. In the horizontal direction, the power consumption at a speed of $\mathbf{v}(t)$ is

$$P_{horizontal}(t) = \|\mathbf{F}\|\|\mathbf{v}(t)\|,$$

where $\mathbf{F}$ is the pulling force, $\mathbf{v}(t) = \|\mathbf{V}_i + \mathbf{a}t\|$ is the instantaneous velocity at time t . When flying at a constant speed of V , $\|\mathbf{F}\| = d_0\rho sAV^2/2$ equals to the fuselage drag $D = \rho SV^2/2$, where S is the rotor solidity, A is the rotor disc area, ρ is the air density.

In the reward model for all DRL agents, the penalty value for all horizontal movements is -0.5, the penalty value for vertical movements is -2, and the penalty value of collision equals to the negative value of the reward needed for the UAV to reach its destination. This setting reduces the frequency of UAV changing flight altitude and keep the UAV away from potential collision. However, this traditional reward model can lead to the problem of sparse rewards. [32] proposes a cumulative reward model that can effectively solve the problem of sparse rewards caused by traditional reward models. The representation of cumulative reward model in 2D environment is:

$$Reward(x,y,l) = \frac{C}{\sqrt{(x-x_d)^2+(y-y_d)^2}\times D(x,y,l)}, \quad (16)$$

where (x, y) is the destination coordinates, l is the size of the safety range, C is a reward constant, and D is the spatial obstacle density in the adjacent space of the current node. In the cumulative reward model, the collision penalty value is equal to $-C$. More explicitly, the value of C is half or less of the reward value for arriving at the destination, while D can be expressed as:

$$D(x,y,l) = \frac{1+Obs(x,y,l)}{S(x,y,l)}, \quad (17)$$

where $S(x, y, l)$ is the number of points in the adjacent points in size l square of the current point (x, y). The 2D form of the cumulative reward model can be directly applied to compressed 2D environments by setting the value of $Obs(x, y, l)$ to the sum of the node values within the range l of (x, y) nodes in the compressed map, because the compressed value also contains information from all compressed layers. If we consider applying the cumulative reward model to the original 3D environment, we need to extend it to 3D, which can be expressed as:

$$R(x,y,z,l) = \frac{C}{\sqrt{(x-x_d)^2+(y-y_d)^2+(z-z_d)^2}\times D(x,y,l)}, \quad (18)$$

where z is the current z coordinate and z_d is the z coordinate of the destination.

Fig. 9 shows the average learning curve of 30 runs of B-APFDQN agent and 30 runs of 3DSIC-DQN agent, where we can see that the runs 3DSIC-DQN agent converge on average at 792.3 episodes while the B-APFDQN agent converge on average at 3191.4 episodes. Hence, the training efficiency of the 3DSIC-DQN scheme is 4.028 times than that of the B-APFDQN scheme. It is not difficult to see from the Fig. 9 that 3DSIC-DQN not only converges quickly but also has a smooth curve, while B-APFDQN's learning curve converges slowly and fluctuates greatly. The convergence value of B-APFDQN is slightly lower than that of 3DSIC-DQN after adding the same ratio of random exploration after the curves of the two algorithms converged. This indicates that the path planning quality of the 3DSIC-DQN agent may

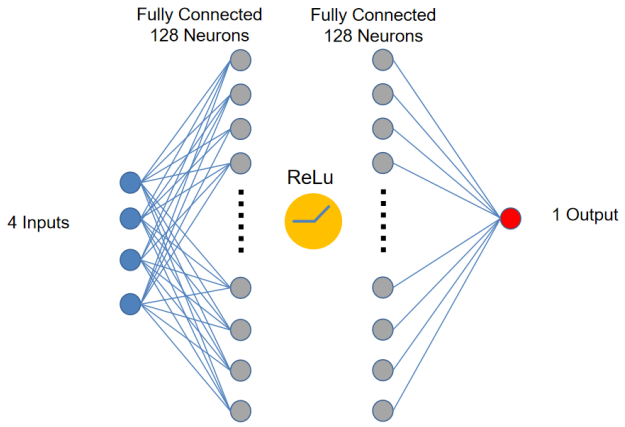

FIGURE 7: Actor neural network structure of 3DSIC-DQN and B-APFDQN.

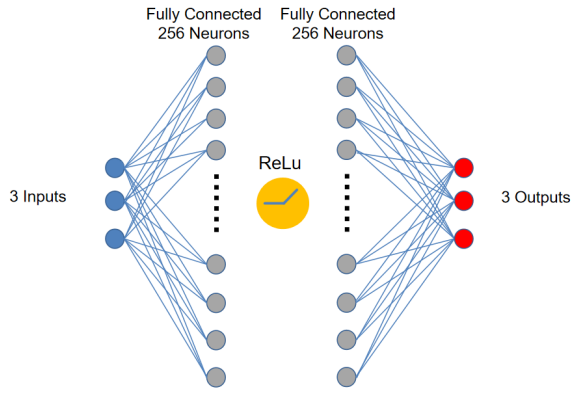

FIGURE 8: Actor neural network structure of 3DSIC-DDPG and FRSVG.

TABLE 3: Hyper-parameters of the DDPG and FRSVG model.

Parameter	Value	Definition
Mini-batch	$2 \times N$	The sampling size of each training, N is the size of $N \times N$ environment.
γ	0.95	Discount factor
η	$N \times 100000$	Capacity of experience pool, N is the size of $N \times N$ environment.
τ	0.001	Learning rate of actor and critic networks
σ_{Init}	1.0	Initial exploration variance
σ_{Min}	0.05	Final exploration variance
M	10000	Maximum number of training episodes
K	10	Steps delay of target networks update

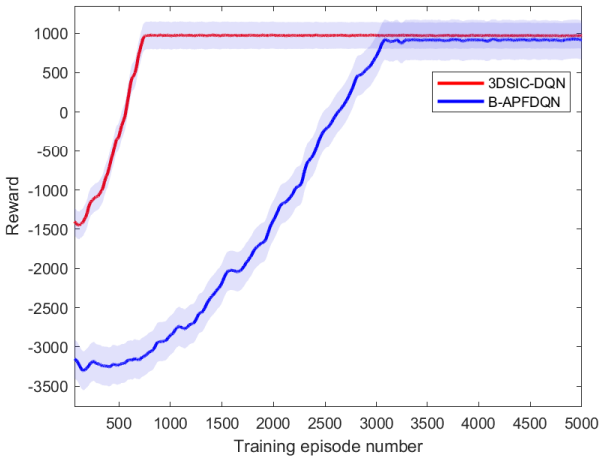

FIGURE 9: Learning curve of 3DSIC-DQN and DQN agents in path planning training.

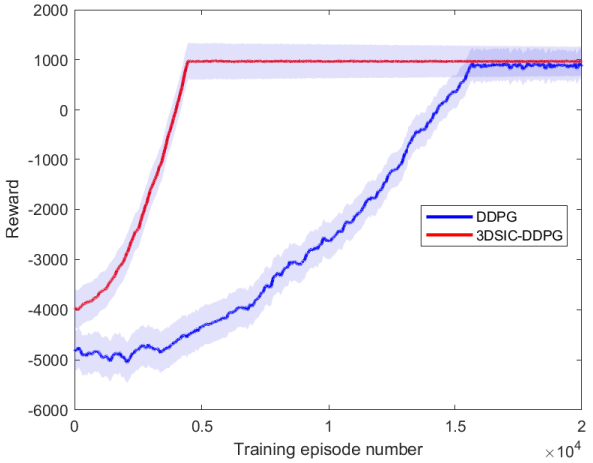

FIGURE 10: Learning curve of 3DSIC-DDPG and DDPG agents in path planning training.

be slightly better than that of the B-APFDQN. While 3DSIC-DQN and B-APFDQN use the same reward model, higher reward values mean fewer actions or lower energy were used to achieve the target. The average number of actions of 3DSIC-DQN path planning result is 218.53 actions, while that of B-APFDQN is 226.80 actions, which proves the above analysis.

Fig. 10 shows the average learning curve of 20 runs of DDPG agent and 20 runs of 3DSIC-DDPG agent, where it can be seen that the 3DSIC-DDPG agent converge faster than the DDPG agent. Specifically, traditional DDPG agents converge on average at 1.73×10^4 episodes while 3DSIC-DDPG agents converge on average at 4.47×10^3 episodes. The average learning curve of the UAV agents trained with traditional DDPG algorithm fluctuates significantly, and the average convergence value after adding the same ratio of random exploration noise after convergence has more significant fluctuations compared to 3DSIC-DDPG. Besides, the average number of actions of 3DSIC-DDPG is 614.3, while that of DDPG is 617.6 actions.

Fig. 11 shows the average learning curve of 20 runs of FRSVG agent and 3DSIC-FRSVG agent, where we compare our proposed 3DSIC scheme to the FRSVG scheme proposed in [35]. As can be seen in Fig. 11 the training efficiency of the combined 3DSIC-FRSVG is higher than the original FRSVG in 3D environment. It is worth noting that in the simulation results, the learning curve of 3DSIC-FRSVG is only retained up to 10000 episodes, and the subsequent part is based on the results of average padding. Based on this simulation result, the FRSVG algorithm using 3DSIC has improved training efficiency by nearly 2 times compared to the original algorithm, and the learning curve is as smooth as the original algorithm. Therefore, 3DSIC has been proven to be a plug and use module combined with different DRL algorithms for UAV path planning tasks in 3D environments.

B. Collision Free Path Planning Rate of 3DSIC

In the path planning tests conducted after the agents of 3DSIC-DQN, 3DSIC-DDPG, and 3DSIC-FRSVG satisfy the convergence conditions, they can all achieve collision free path planning. This can also be seen from the learning curve, as the huge penalty value of collisions can cause a significant decrease in the reward curve, which is not reflected in their learning curve. However, this is only the result of the training part, and in actual deployment, a certain amount of random exploration is often added to track the dynamic changes in the environment. In the DQN algorithm, random exploration is achieved through the ϵ-greedy algorithm, while in the DDPG and FRSVG algorithms, random exploration is

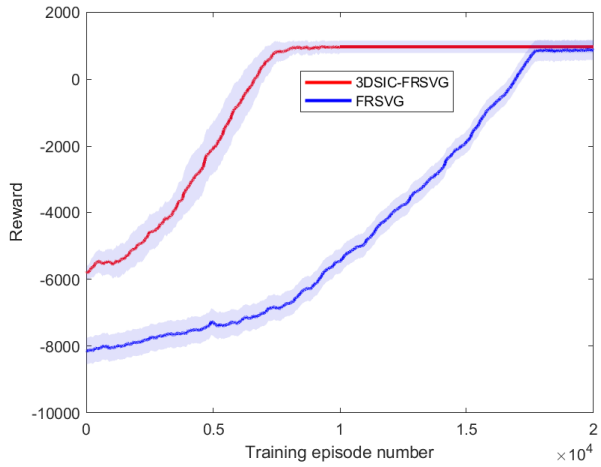

FIGURE 11: Learning curve of 3DSIC-FRSVG and FRSVG agents in path planning training.

TABLE 4: Collision free path planning probability of different DRL algorithms.

Type of agent	number of collision paths	Total number of test	Collision free rate
3DSIC-DQN	181	3000	93.96%
B-APFDQN	164	3000	94.53%
3DSIC-DDPG	217	2000	89.15%
DDPG	149	2000	92.55%
3DSIC-FRSVG	105	2000	94.75%
FRSVG	52	2000	97.40%

achieved by adding random noise to the output of the action network. In order to compare the collision free path planning probabilities of different algorithms in actual deployment, we conducted 100 path planning tests on each trained 3DSIC-DQN, B-APFDQN, 3DSIC-DDPG, DDPG, 3DSIC-FRSVG, and FRSVG agent, and recorded the number and probability of collision free path planning. The experimental results are shown in Table 4, where we added random exploration with a probability of 0.1 for 3DSIC-DQN and B-APFDQN. Random noise with a mean of 0 and a variance of 0.05 was added to 3DSIC-DDPG, DDPG, 3DSIC-FRSVG, and FRSVG. Additionally, for the fairness of the simulations, the random generator seeds used for tests with the same serial number are the same. It is not difficult to see from the simulation results that the application of 3DSIC does have an impact on the probability of collision free path planning. The highest among them is the DDPG algorithm, which reduces the probability of collision free path planning by 3.4%. Meanwhile, the success rate of collision free path planning in 3DSIC-DQN is only reduced by 0.57% compared to B-APFDQN, and its impact can be almost ignored.

TABLE 5: Different compression factors

Parameter	Value	Definition
β_{l1}	NaN	No use of 3DSIC
β_{l3}	(0.2, 0.6, 0.2)	compress factor for 3 layers.
β_{l5}	(0.1, 0.2, 0.4, 0.2, 0.1)	compress factor for 5 layers.
β_{l7}	(0.05, 0.1, 0.15, 0.4, 0.15, 0.1, 0.05)	compress factor for 7 layers.
β_{f1}	(0.1, 0.2, 0.4, 0.2, 0.1)	compress factor 1
β_{f2}	(0.05, 0.2, 0.5, 0.2, 0.05)	compress factor 2

It can be proved that the combination of 3DSIC algorithm and DRL algorithms greatly improve the training efficiency, and the reduction in the quality of path planning is limited.

C. Training Efficiency Comparison of Different 3DSIC Compression Factor

In order to better compare the performance of 3DSIC schemes that compress different numbers of layers, we designed a $20\times20\times20$ complex forest environment as shown in Fig. 12, which can obtain more number of simulation runs than $100 \times 100 \times 50$ environment, where black dot represents the start point, the red dot denotes the destination, and other filled dots are the obstacles at different height. In this simulation experiment, the UAV agents are trained using different number of compression layers, not using compression, and using the same compression layer but different compression factors, with 1 layer (only considering the current flight altitude), 3 layers, 5 layers, and 7 layers. More specifically, the size and value of the compression factor β used in training UAV agents will be set different, which can be find in Table 5, and those compression factors will be used to train 10 runs of UAV agent and draw their average learning curves to compare training efficiency under different compression factors.

We compared the learning curve of 1 layer with $\beta_{l1} = (1)$, 3 layers with $\beta_{l3} = (0.2, 0.6, 0.2)$, 5 layers with $\beta_{l5} = (0.1, 0.2, 0.4, 0.2, 0.1)$ and 7 layers with $\beta_{l7} = (0.05, 0.1, 0.15, 0.4, 0.15, 0.1, 0.05)$. Let us consider β_{l3} as an example, the middle element in the compression factor vector represents the compression weight of the current layer, while the first and third elements represent the compression weight of the above and below layers of the current layer, respectively. The average learning curve of 10 runs of training agent for different compression layers are show in Fig. 13. The simulation results show that using more compression layers leads to an improvement in learning efficiency, but the improvement becomes less significant as the number of

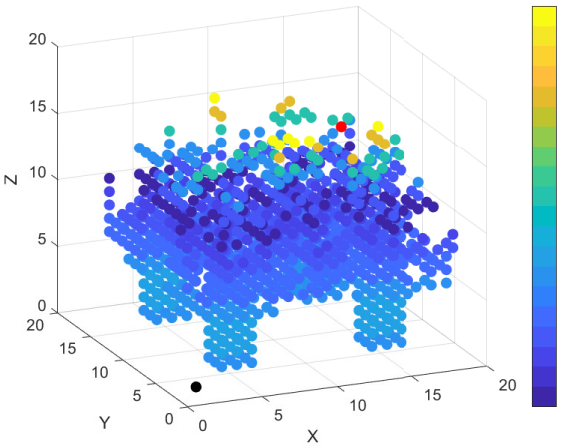

FIGURE 12: Cluttered 3D forest environment.

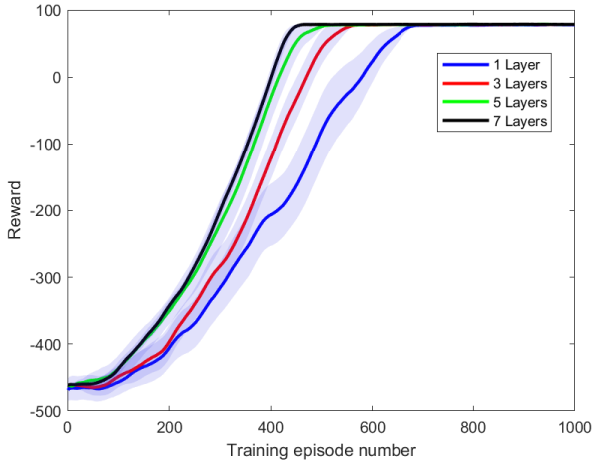

FIGURE 13: Learning curve of DQN-3DSIC with different compression layers.

layers increases. When the 3DSIC algorithm is not used, changes in flight altitude may lead to a significant slowdown in the learning rate, as shown by several "gentle steps" in the blue curve in Fig. 13. Moreover, in terms of the time required for training, more training time for each episode is required when more compression layers are considered.

Additionally, for the same number of compression layers, different compression factors may have some impact on the training efficiency, although not significant, which is shown in Fig. 14. Explicitly, in Fig. 14, it can be seen that the two learning curves are very similar with compression factor 1 using $\beta_1 = (0.1, 0.2, 0.4, 0.2, 0.1)$ and compression factor 2 using $\beta_2 = (0.05, 0.2, 0.5, 0.2, 0.05)$. Therefore, selecting the compression factors, and the number of layers to be compressed represents a trade off of performance and training time requirements. We suggest to use fewer compression layers in indoor dense and complex environments, which not only improves compression efficiency, but also reduces the receptive field of interaction between agents and the environment, which is beneficial for agents to consider planning for local situations. For open and sparse obstacle environments, we recommend using more compression layers. Although it will increase the demand for computing resources, it will be more accurate for global planning. If the compression factor is designed based on energy consumption, then when the energy consumption of vertical motion is much greater than that of horizontal motion, fewer compression layers should be considered, and the weight of the current layer should be increased and reduce the weight of heights away from the current layer. If the energy consumption of increasing height is greater than that of decreasing height, then the layers with a higher height than the current layer should have a smaller weight than the layers with a lower height than the current layer.

Fig. 15 shows the average learning curve of 10 runs of agent training with 2D traditional reward model, 10 runs

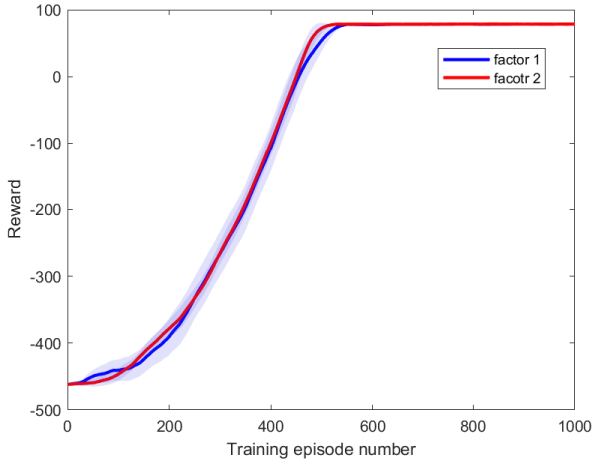

FIGURE 14: Learning curve of DQN-3DSIC with different compression factors.

of agent training with 3D traditional reward model, 10 runs of training with 2D cumulative reward model and 10 runs of agent training with 3D cumulative reward model in a $20 \times 20 \times 20$ 3D environment shown in Fig. 12. All agents in the simulation use β_1 as the compression factor. To minimize the impact of random exploration, each agent with the same index number uses the same random number generator seed. It is not difficult to see that using the cumulative reward model in the compressed 2D environment has the fastest convergence speed. Traditional reward model with the 2D environment after compression converged earlier than the cumulative reward model with the original 3D environment. This is because the small environment size does not have serious reward sparsity problem.

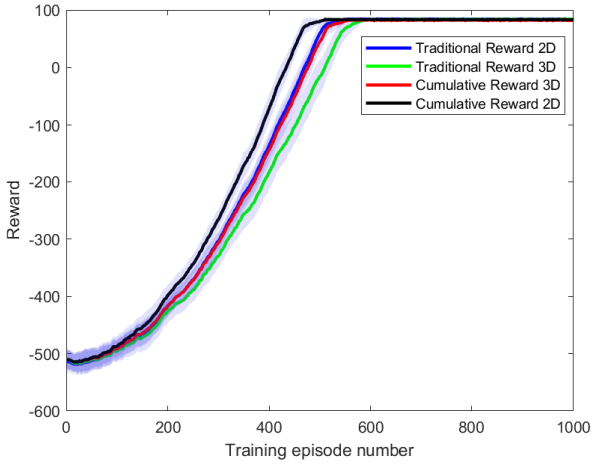

FIGURE 15: Learning curve of DQN-3DSIC with different reward models in $20 \times 20 \times 20$ 3D environment.

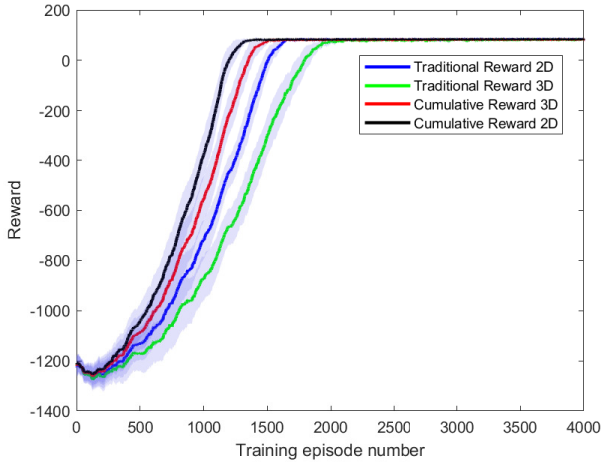

FIGURE 16: Learning curve of DQN-3DSIC with different reward models in $40 \times 40 \times 40$ 3D environment.

As the size of the environment increases, the rewards of data in the experience pool become sparser. Fig. 15 shows the average learning curves of 40 runs UAV agent with 4 different reward modes trained in a $40 \times 40 \times 40$ 3D environment. Similarly, each reward model trained 10 runs of the UAV agent to obtain an average learning curve. From Fig. 15, it can be observed that when the environment size is expanded to 40, the cumulative reward model brings a significant improvement in training efficiency. Whether combined with the compressed 2D environment or the original 3D environment, the cumulative reward models exhibits higher learning efficiency than the traditional reward models. Therefore, the problem of excessive search space and sparse rewards is related. More specifically, data that is not sparse in low dimensional or small size spaces will exhibit sparsity in large size or high-dimensional spaces.

V. Conclusions

In this paper, we proposed a 3D spatial information compression method for significantly reducing the search space for path planning problems of UAVs in 3D MDP environments. This method reduces the redundancy of environmental information while keeping the key spatial information. Our proposed 3D spatial information compression method can be combined with various DRL algorithms to train UAV agents more efficiently. Therefore, the 3DSIC algorithm can adapt to both continuous and discrete 3D MDP environments. The simulation experiment results show that the training efficiency of the DQN agent using 3DSIC for compression is 4.028 times higher than that of the B-APFDQN algorithm without 3DSIC, when considering discrete 3D space. The efficiency of training UAV agents using 3DSIC-DDPG algorithm is 3.9 times higher than that of training UAV agents using traditional DDPG algorithm, when considering a continuous 3D MDP environment. The FRSVG algorithm, which has been proven to be efficient, attains an improved training efficiency when combined with the 3DSIC algorithm. Finally, using more compression layers exhibits higher training efficiency, but this improvement becomes less pronounced as the number of layers increases.

REFERENCES

[1] Jeongeun Kim, Seungwon Kim, Chanyoung Ju, and Hyoung Il Son. Unmanned Aerial Vehicles in Agriculture: A Review of Perspective of Platform, Control, and Applications. *IEEE Access*, 7:105100–105115, 2019.

[2] Mingze Zhang, Yifeng Xiong, Soon Xin Ng, and Mohammed El-Hajjar. Content-Aware Transmission in UAV-Assisted Multicast Communication. *IEEE Transactions on Wireless Communications*, 22(11):7144–7157, 2023.

[3] Suttinee Sawadsitang, Dusit Niyato, Puay Siew Tan, Ping Wang, and Sarana Nutanong. Shipper Cooperation in Stochastic Drone Delivery: A Dynamic Bayesian Game Approach. *IEEE Transactions on Vehicular Technology*, 70(8):7437–7452, 2021.

[4] Mingze Zhang, Mohammed EI-Hajjar, and Soon Xin Ng. Intelligent Caching in UAV-Aided Networks. *IEEE Transactions on Vehicular Technology*, 71(1):739–752, 2022.

[5] Wu Yi, Chen Liming, Kong Lingyu, Zhang Jie, and Wang Miao. Research on application mode of large fixed-wing UAV system on overhead transmission line. In *2017 IEEE International Conference on Unmanned Systems (ICUS)*, pages 88–91, 2017.

[6] Mingze Zhang, Yifeng Xiong, Soon Xin Ng, and Mohammed El-Hajjar. Deployment of Energy-Efficient Aerial Communication Platforms With Low-Complexity Detection. *IEEE Transactions on Vehicular Technology*, 72(9):12016–12030, 2023.

[7] Chuanzheng Li, Chuang Xue, and Yue Bai. Experimental investigation on aerodynamics of nonplanar rotor pairs in a multi-rotor UAV. In *2019 14th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pages 911–915, 2019.

[8] Xukai Zhong, Yiming Huo, Xiaodai Dong, and Zhonghua Liang. Deep Q-Network Based Dynamic Movement Strategy in a UAV-Assisted Network. In *2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall)*, pages 1–6, 2020.

[9] Caiwu Ding and Lu Lu. A Tilting-Rotor Unmanned Aerial Vehicle for Enhanced Aerial Locomotion and Manipulation Capabilities: Design, Control, and Applications. *IEEE/ASME Transactions on Mechatronics*, 26(4):2237–2248, 2021.

[10] Haoxin Liu and Yonghui Zhang. ASL-DWA: An Improved A-Star Algorithm for Indoor Cleaning Robots. *IEEE Access*, 10:99498–99515, 2022.

[11] Gang Tang, Congqiang Tang, Christophe Claramunt, Xiong Hu, and Peipei Zhou. Geometric A-Star Algorithm: An Improved A-Star Algorithm for AGV Path Planning in a Port Environment. *IEEE Access*, 9:59196–59210, 2021.

[12] Nelapati Lava Prasad and Barathram Ramkumar. 3-D Deployment and Trajectory Planning for Relay Based UAV Assisted Cooperative Communication for Emergency Scenarios Using Dijkstra's Algorithm. *IEEE Transactions on Vehicular Technology*, 72(4):5049–5063, 2023.

[13] Jie Qi, Hui Yang, and Haixin Sun. MOD-RRT*: A Sampling-Based Algorithm for Robot Path Planning in Dynamic Environment. *IEEE Transactions on Industrial Electronics*, 68(8):7244–7251, 2021.

[14] Reza Mashayekhi, Mohd Yamani Idna Idris, Mohammad Hossein Anisi, Ismail Ahmedy, and Ihsan Ali. Informed RRT*-Connect: An Asymptotically Optimal Single-Query Path Planning Method. *IEEE Access*, 8:19842–19852, 2020.

[15] Pritam Ojha and Atul Thakur. Real-Time Obstacle Avoidance Algorithm for Dynamic Environment on Probabilistic Road Map. In *2021 International Symposium of Asian Control Association on Intelligent Robotics and Industrial Automation (IRIA)*, pages 57–62, 2021.

[16] Zhenhua Pan, Chengxi Zhang, Yuanqing Xia, Hao Xiong, and Xiaodong Shao. An Improved Artificial Potential Field Method for Path Planning and Formation Control of the Multi-UAV Systems. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 69(3):1129–1133, 2022.

[17] Jiayi Sun, Jun Tang, and Songyang Lao. Collision Avoidance for Cooperative UAVs With Optimized Artificial Potential Field Algorithm. *IEEE Access*, 5:18382–18390, 2017.

[18] Cong Zhao, Yifan Zhu, Yuchuan Du, Feixiong Liao, and Ching-Yao Chan. A Novel Direct Trajectory Planning Approach Based on Generative Adversarial Networks and Rapidly-Exploring Random Tree. *IEEE Transactions on Intelligent Transportation Systems*, 23(10):17910–17921, 2022.

[19] Jie Qi, Hui Yang, and Haixin Sun. MOD-RRT*: A Sampling-Based Algorithm for Robot Path Planning in Dynamic Environment. *IEEE Transactions on Industrial Electronics*, 68(8):7244–7251, 2021.

[20] Jiankun Wang, Wenzheng Chi, Chenming Li, Chaoqun Wang, and Max Q.-H. Meng. Neural RRT*: Learning-Based Optimal Path Planning. *IEEE Transactions on Automation Science and Engineering*, 17(4):1748–1758, 2020.

[21] Jingcheng Zhang, Yuqiang An, Jianing Cao, Shibo Ouyang, and Lei Wang. UAV Trajectory Planning for Complex Open Storage Environments Based on an Improved RRT Algorithm. *IEEE Access*, 11:23189–23204, 2023.

[22] Matej Novosad, Robert Penicka, and Vojtech Vonasek. CTop-PRM: Clustering Topological PRM for Planning Multiple Distinct Paths in 3D Environments. *IEEE Robotics and Automation Letters*, 8(11):7336–7343, 2023.

[23] Yanli Chen, Guiqiang Bai, Yin Zhan, Xinyu Hu, and Jun Liu. Path Planning and Obstacle Avoiding of the USV Based on Improved ACO-APF Hybrid Algorithm With Adaptive Early-Warning. *IEEE Access*, 9:40728–40742, 2021.

[24] Yuting Wan, Yanfei Zhong, Ailong Ma, and Liangpei Zhang. An Accurate UAV 3-D Path Planning Method for Disaster Emergency Response Based on an Improved Multiobjective Swarm Intelligence Algorithm. *IEEE Transactions on Cybernetics*, 53(4):2658–2671, 2023.

[25] Zhe Zhang, Jian Wu, Jiyang Dai, and Cheng He. A Novel Real-Time Penetration Path Planning Algorithm for Stealth UAV in 3D Complex Dynamic Environment. *IEEE Access*, 8:122757–122771, 2020.

[26] Huan Liu, Guohua Wu, Ling Zhou, Witold Pedrycz, and Ponnuthurai Nagaratnam Suganthan. Tangent-Based Path Planning for UAV in a 3-D Low Altitude Urban Environment. *IEEE Transactions on Intelligent Transportation Systems*, 24(11):12062–12077, 2023.

[27] Jiehong Wu, Ya'nan Sun, Danyang Li, Junling Shi, Xianwei Li, Lijun Gao, Lei Yu, Guangjie Han, and Jinsong Wu. An Adaptive Conversion Speed Q-Learning Algorithm for Search and Rescue UAV Path Planning in Unknown Environments. *IEEE Transactions on Vehicular Technology*, 72(12):15391–15404, 2023.

[28] Fatemeh Rekabi-Bana, Junyan Hu, Tomáš Krajník, and Farshad Arvin. Unified Robust Path Planning and Optimal Trajectory Generation for Efficient 3D Area Coverage of Quadrotor UAVs. *IEEE Transactions on Intelligent Transportation Systems*, 25(3):2492–2507, 2024.

[29] Hao Xie, Dingcheng Yang, Lin Xiao, and Jiangbin Lyu. Connectivity-Aware 3D UAV Path Design With Deep Reinforcement Learning. *IEEE Transactions on Vehicular Technology*, 70(12):13022–13034, 2021.

[30] Lijuan Zhang, Jiabin Peng, Weiguo Yi, Hang Lin, Lei Lei, and Xiaoqin Song. A State-Decomposition DDPG Algorithm for UAV Autonomous Navigation in 3-D Complex Environments. *IEEE Internet of Things Journal*, 11(6):10778–10790, 2024.

[31] Hu Teng, Ishtiaq Ahmad, Alamgir Msm, and Kyunghi Chang. 3D Optimal Surveillance Trajectory Planning for Multiple UAVs by Using Particle Swarm Optimization With Surveillance Area Priority. *IEEE Access*, 8:86316–86327, 2020.

[32] Zhipeng Wang, Soon Xin Ng, and Mohammed EI-Hajjar. Deep reinforcement learning assisted uav path planning relying on cumulative reward mode and region segmentation. *IEEE Open Journal of Vehicular Technology*, 5:737–751, 2024.

[33] Chao Wang, Jian Wang, Yuan Shen, and Xudong Zhang. Autonomous Navigation of UAVs in Large-Scale Complex Environments: A Deep Reinforcement Learning Approach. *IEEE Transactions on Vehicular Technology*, 68(3):2124–2136, 2019.

[34] Lixin Lyu, Hong Jiang, and Fan Yang. Improved Dung Beetle Optimizer Algorithm With Multi-Strategy for Global Optimization and UAV 3D Path Planning. *IEEE Access*, 12:69240–69257, 2024.

[35] Yuntao Xue and Weisheng Chen. A UAV Navigation Approach Based on Deep Reinforcement Learning in Large Cluttered 3D Environments. *IEEE Transactions on Vehicular Technology*, 72(3):3001–3014, 2023.

[36] Changsheng Huang, Yanpu Zhao, Mengjie Zhang, and Hongyan Yang. APSO: An A*-PSO Hybrid Algorithm for Mobile Robot Path Planning. *IEEE Access*, 11:43238–43256, 2023.

[37] Zhenhua Yu, Zhijie Si, Xiaobo Li, Dan Wang, and Houbing Song. A Novel Hybrid Particle Swarm Optimization Algorithm for Path Planning of UAVs. *IEEE Internet of Things Journal*, 9(22):22547–22558, 2022.

[38] Yu Wu, Kin Huat Low, Bizhao Pang, and Qingyu Tan. Swarm-Based 4D Path Planning For Drone Operations in Urban Environments. *IEEE Transactions on Vehicular Technology*, 70(8):7464–7479, 2021.

[39] Mehdi Maboudi, MohammadReza Homaei, Soohwan Song, Shirin Malihi, Mohammad Saadatseresht, and Markus Gerke. A Review on Viewpoints and Path Planning for UAV-Based 3-D Reconstruction. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 16:5026–5048, 2023.

[40] Ronglei Xie, Zhijun Meng, Lifeng Wang, Haochen Li, Kaipeng Wang, and Zhe Wu. Unmanned Aerial Vehicle Path Planning Algorithm Based on Deep Reinforcement Learning in Large-Scale and Dynamic Environments. *IEEE Access*, 9:24884–24900, 2021.

[41] Liangheng Lv, Sunjie Zhang, Derui Ding, and Yongxiong Wang. Path Planning via an Improved DQN-Based Learning Policy. *IEEE Access*, 7:67319–67330, 2019.

[42] David Silver Volodymyr Mnih, Koray Kavukcuoglu. Human-level control through deep reinforcement learning. *Nature*, page 529–533, 2015.

[43] Shimin Gong, Meng Wang, Bo Gu, Wenjie Zhang, Dinh Thai Hoang, and Dusit Niyato. Bayesian Optimization Enhanced Deep Reinforcement Learning for Trajectory Planning and Network Formation in Multi-UAV Networks. *IEEE Transactions on Vehicular Technology*, 72(8):10933–10948, 2023.

[44] Minah Seo, Luiz Felipe Vecchietti, Sangkeum Lee, and Dongsoo Har. Rewards Prediction-Based Credit Assignment for Reinforcement Learning With Sparse Binary Rewards. *IEEE Access*, 7:118776–118791, 2019.

[45] Matvey Gerasyov and Ilya Makarov. Dealing With Sparse Rewards Using Graph Neural Networks. *IEEE Access*, 11:89180–89187, 2023.

[46] Fuchen Kong, Qi Wang, Shang Gao, and Hualong Yu. B-APFDQN: A UAV Path Planning Algorithm Based on Deep Q-Network and Artificial Potential Field. *IEEE Access*, 11:44051–44064, 2023.

[47] Zhuoyong Shi, Guoqing Shi, Jiandong Zhang, Dinghan Wang, Tianyue Xu, Longmeng Ji, and Yong Wu. Design of UAV Flight State Recognition System for Multisensor Data Fusion. *IEEE Sensors Journal*, 24(13):21386–21394, 2024.

[48] Xiaolin Xiao and Yicong Zhou. Two-Dimensional Quaternion PCA and Sparse PCA. *IEEE Transactions on Neural Networks and Learning Systems*, 30(7):2028–2042, 2019.

[49] Lingzhi Zhu, Shuning Zhang, Qun Ma, Huichang Zhao, Si Chen, and Dongxu Wei. Classification of UAV-to-Ground Targets Based on Enhanced Micro-Doppler Features Extracted via PCA and Compressed Sensing. *IEEE Sensors Journal*, 20(23):14360–14368, 2020.

[50] Jiandong Zhang, Zhuoyong Shi, Anli Zhang, Qiming Yang, Guoqing Shi, and Yong Wu. UAV Trajectory Prediction Based on Flight State Recognition. *IEEE Transactions on Aerospace and Electronic Systems*, 60(3):2629–2641, 2024.

[51] Umit Celik and Haluk Eren. Classification of Manifold Learning Based Flight Fingerprints of UAVs in Air Traffic. *IEEE Transactions on Intelligent Transportation Systems*, 24(5):5229–5238, 2023.

[52] Amit Konar, Indrani Goswami Chakraborty, Sapam Jitu Singh, Lakhmi C. Jain, and Atulya K. Nagar. A Deterministic Improved Q-Learning for Path Planning of a Mobile Robot. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 43(5):1141–1153, 2013.

[53] Dongcheng Li, Wangping Yin, W. Eric Wong, Mingyong Jian, and Matthew Chau. Quality-Oriented Hybrid Path Planning Based on A* and Q-Learning for Unmanned Aerial Vehicle. *IEEE Access*, 10:7664–7674, 2022.

[54] Kyriakos G. Vamvoudakis and Nick-Marios T. Kokolakis. *Synchronous Reinforcement Learning-Based Control for Cognitive Autonomy*. Now Foundations and Trends, 2020.

[55] Meng Zhao, Hui Lu, Siyi Yang, and Fengjuan Guo. The Experience-Memory Q-Learning Algorithm for Robot Path Planning in Unknown Environment. *IEEE Access*, 8:47824–47844, 2020.

[56] Nan Zheng and Pinaki Mazumder. *Fundamentals and Learning of Artificial Neural Networks*, pages 11–60. 2020.

[57] Muhammad Affan, Junaid Jawaid, Syed Umaid Ahmed, Ali Isfand yar Manek, and Riaz Uddin. Solving Combinatorial Problems through Off-Policy Reinforcement Learning Methods. In *2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE)*, pages 1–5, 2020.

[58] Chunxue Wu, Bobo Ju, Yan Wu, Xiao Lin, Naixue Xiong, Guangquan Xu, Hongyan Li, and Xuefeng Liang. UAV Autonomous Target Search Based on Deep Reinforcement Learning in Complex Disaster Scene. *IEEE Access*, 7:117227–117245, 2019.

[59] Nada Kadhim and Monjur Mourshed. A Shadow-Overlapping Algorithm for Estimating Building Heights From VHR Satellite Images. *IEEE Geoscience and Remote Sensing Letters*, 15(1):8–12, 2018.

[60] Hua Yan, Yunfei Chen, and Shuang-Hua Yang. New Energy Consumption Model for Rotary-Wing UAV Propulsion. *IEEE Wireless Communications Letters*, 10(9):2009–2012, 2021.