drones

MDPI

Article

A Hybrid Human-in-the-Loop Deep Reinforcement Learning Method for UAV Motion Planning for Long Trajectories with Unpredictable Obstacles

Sitong Zhang [1], Yibing Li [1,*], Fang Ye [2], Xiaoyu Geng [1], Zitao Zhou [1] and Tuo Shi [3]

[1] Key Laboratory of Advanced Marine Communication and Information Technology, The College of Information and Communication Engineering, Harbin Engineering University, Harbin 150001, China; zst9626@hrbeu.edu.cn (S.Z.); geng1219@hrbeu.edu.cn (X.G.)
[2] National Key Laboratory of Underwater Acoustic Technology, The College of Information and Communication Engineering, Harbin Engineering University, Harbin 150001, China; yefang@hrbeu.edu.cn
[3] School of Computer Science and Technology, College of Intelligence and Computing, Tianjin University, Tianjin 300350, China; shituo@tju.edu.cn
* Correspondence: liyibing@hrbeu.edu.cn

Abstract: Unmanned Aerial Vehicles (UAVs) can be an important component in the Internet of Things (IoT) ecosystem due to their ability to collect and transmit data from remote and hard-to-reach areas. Ensuring collision-free navigation for these UAVs is crucial in achieving this goal. However, existing UAV collision-avoidance methods face two challenges: conventional path-planning methods are energy-intensive and computationally demanding, while deep reinforcement learning (DRL)-based motion-planning methods are prone to make UAVs trapped in complex environments—especially for long trajectories with unpredictable obstacles—due to UAVs' limited sensing ability. To address these challenges, we propose a hybrid collision-avoidance method for the real-time navigation of UAVs in complex environments with unpredictable obstacles. We firstly develop a Human-in-the-Loop DRL (HL-DRL) training module for mapless obstacle avoidance and secondly establish a global-planning module that generates a few points as waypoint guidance. Moreover, a novel goal-updating algorithm is proposed to integrate the HL-DRL training module with the global-planning module by adaptively determining the to-be-reached waypoint. The proposed method is evaluated in different simulated environments. Results demonstrate that our approach can rapidly adapt to changes in environments with short replanning time and prevent the UAV from getting stuck in maze-like environments.

Keywords: unmanned aerial vehicles; collision avoidance; global path planning; DRL-based motion planning

Citation: Zhang, S.; Li, Y.; Ye, F; Geng, X; Zhou, Z; Shi, T. A Hybrid Human-in-the-Loop Deep Reinforcement Learning Method for UAV Motion Planning for Long Trajectories with Unpredictable Obstacles. *Drones* **2023**, *7*, 311. https://doi.org/10.3390/drones7050311

Academic Editor: Carlos Tavares Calafate

Received: 31 March 2023
Revised: 2 May 2023
Accepted: 3 May 2023
Published: 6 May 2023

1. Introduction

An Unmanned Aerial Vehicle (UAV), as a system with high maneuverability, can provide a wide range of applications for humans' daily life. For example, UAVs can be used for search and rescue [1,2], relays in communication systems [3–5], public health emergency management [6–8], IoT data collection [9], and more. Regardless of the task type, an effective autonomous-navigation method can guide the UAV to its destination in a safe, smooth, and cost-efficient manner, which is always a fundamental problem in task completion.

The classic path-planning algorithms such as Dijkstra, A*, PRM, and RRT* [10,11] can handle the prototypical navigation problems of finding a path from one point to another while avoiding obstacles. These algorithms assume that obstacles are perfectly known and stationary and that the planned path can be followed accurately. This is referred to as offline path planning because planning is finished in advance of execution. However, the main weakness of these algorithms is that they are incapable of dealing with uncertainty and

changes in the environment. Furthermore, the path planned by these classic algorithms only includes location information and does not take into account the velocity constraint. This could lead to a new issue in which the planned path does not obey the robot's dynamics.

To tackle these problems, more recent works have concentrated on improving the classic path-planning algorithms. Some of them, such as [12–14], are geared toward kinodynamic problems. The goal of kinodynamic path planning is to create a robot's motion within kinematic constraints, such as avoiding obstacles, and dynamic constraints, such as maximum velocity bounds [15]. Therefore, such methods always build a graph with all edges executable by the robot at the front-end path finding process or have trajectory optimization at the back-end. Other works such as [16–19] focus on online replanning solutions. These methods always reduce the offline planner's execution time, enabling it to continually replan in real-time as changes to the surroundings are detected. In addition, a great number of works [20–24] have been developed considering both robot dynamics and uncertainty in the environment. However, the works mentioned so far suffer from the fact that they are energy-intensive and require a considerable amount of computational resources. Quadrotors, being relatively small-scale UAVs, have limited electric power and computational resources [25,26], so it is not easy to carry out the above methods onboard.

As a result, in recent years, various DRL-based obstacle-avoidance solutions [27–38] have gotten a lot of attention due to their real-time, kinodynamic, and energy-efficient features. DRL is a computational approach for learning how to map states to actions to obtain the optimized policy [39]. The DRL agent is not given any exact labels or known maps; instead, it uses trial-and-error to figure out the best action set. When it comes to UAV navigation, DRL has a series of advantages. First, all the DRL-based methods emphasize training the policy by interacting with the environment directly. This means that actions are calculated in real-time using current sensor readings, ensuring the ability to respond to the changes. Second, unlike traditional path-planning approaches that output waypoints, the DRL framework returns the UAV's actions directly, such as providing velocity and acceleration. Thus, it is inherently subject to dynamics constraints. Third, DRL is based on neural networks, which is an end-to-end approach and has lower computational complexity. Such a property makes these methods more energy-efficient. Nevertheless, the lack of global information makes DRL-based algorithms run the risk of being stuck in maze-like scenarios. When the trajectory is long and the environment is full of uncertain obstacles, a maze-like scenario usually occurs. As shown in Figure 1, the UAV departs from Point A and heads towards the goal. When it arrives at Point B, the UAV's laser ranger detects an obstacle ahead, and it returns. Because of its limited field-of-view sensing, it immediately discovers that the surroundings are without obstacles and then goes back, which eventually leads to the UAV hovering around in the corner. Based on the results of the state-of-art [38], although the DRL network is modified to train a policy for UAVs to escape from maze-like obstacles, there is still a 10% failure rate when the trajectory grows longer.

Based on the analysis above, it is still a challenging problem to energy-effectively navigate the UAV to avoid unpredictable obstacles for long trajectories. To address such a problem, we propose the DRL Enhanced Global Long Trajectories Planning Framework (DGlobal for short). The DGlobal comprises two key modules: a DRL training module and a global-planning module. The training module learns a DRL policy for obstacle-avoiding motion planning. The global-planning module first generates an optimal collision-free global path based on previous knowledge. Such a path is separated into a few waypoints while keeping the major characteristics of the path. Then, the well-trained policy in the training module navigates the UAV among the waypoints until it at last reaches the destination. However, the DGlobal Framework still faces the following two challenges.

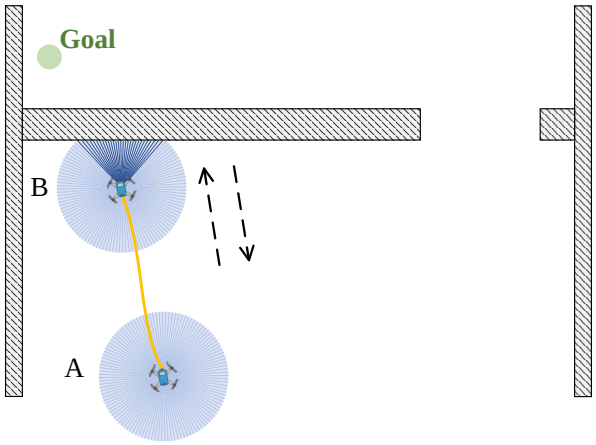


Figure 1. Illustration of a UAV being trapped in a maze-like situation.

1. To effectively train a DRL policy in a complex environment. We use a robust and accurate physics engine, Gazebo [40], to simulate the environment so that the trained policy can be more practical. However, training a DRL policy in such an accurate and complex environment is time-consuming and requires a large volume of computing resources.

2. To dynamically update the waypoints based on unpredictable obstacles. The waypoints generated by the global-planning module can be blocked by unpredictable obstacles that will mislead the UAV and make it unable to reach the destination.

Although the authors in [41,42] also considered the combination of global planning (RRT algorithm, for example) and DRL-based planning, their major idea is to use the DRL method to build the RRT tree instead of regarding the global-planning method as guidance in their work, and they did not consider either how to avoid unpredictable obstacles or how to improve training efficiency.

In this work, we not only propose the DGlobal Framework but also solve the above two challenges. First, inspired by the human-in-the-loop concept [43], we adopt a Human-in-the-Loop DRL (HL-DRL) Training Module to use the demonstration replay to accelerate the learning process. Second, we propose a goal-updating algorithm that can detect unpredictable obstacles and determine which waypoint is to be reached.

The main contributions of this work are summarized as follows.

(1) We propose the DRL Enhanced Global Long Trajectories Planning Framework to navigate a UAV to avoid unpredictable obstacles for long trajectories.
(2) We adopt the human-in-the-loop concept and use the HL-DRL module to train a policy for UAV obstacle avoidance in real-time.
(3) We establish a global-planning module for waypoint guidance for UAV navigation in maze-like environments.
(4) We use Gazebo [40] in conjunction with ROS as the simulator and build different types of environments to validate the efficiency of the proposed method.

The rest of this paper is organized as follows. Section 2 surveys related works. Section 3 formulates the navigation problem as a Markov Decision Process (MDP) and introduces some background knowledge of DRL. The proposed method is elaborated in Section 4. In Section 5, we describe the simulation details and analyze the experiment results. Last, Section 6 concludes the paper.

2. Related Works

Path planning and trajectory planning are both important for unmanned vehicles, such as planar vehicles [44], underwater vehicles [45], and UAVs. Path planning aims to find an optimal path from a starting point to a goal point, while trajectory planning focuses on converting a path that has already been determined into an actual executable motion

trajectory for unmanned vehicles. In the following, we summarize existing path-planning and trajectory-planning methods for both planar vehicles and UAVs, and more details can be found in corresponding surveys [46,47].

2.1. Path and Trajectory Planning for Planar Vehicles

As mentioned in [44], path and trajectory planning are crucial for planar vehicles, for example, self-driving cars, electronic vehicles [48], and robotic vacuum cleaners.

The classic path-planning algorithms such as Dijkstra, A*, PRM, and RRT* [10,11] can help vehicles find a path from one point to another while avoiding obstacles. However, such a path always ignores the feasibility of the dynamics. In recent years, authors have paid more attention to trajectory planning that considers vehicle motion. Xiong et al. [12] developed an A*-based trajectory processing and tracking method that considers the vehicle contour when configuring the raster map and uses the Bessel curve to smooth the path. A trajectory-planning algorithm called Smooth-RRT* was proposed in [13] that can generate a smoothly curved trajectory satisfying kinodynamic constraints and asymptotic optimality. The authors in [14] proposed DT-RRT for two-wheeled differential mobile robot navigation.

A large volume of current research has also focused on the capability to replan as the environment changes. In [16], RRTX was designed for real-time navigation in dynamic situations. When new sensor data update the environment model, a graph-rewiring cascade refines the search graph and repairs the shortest path to the goal sub-tree. An RRT*-based algorithm for semi-autonomous vehicles was developed in [17]. The authors created a replanning technique to update the path rapidly without full revision of the RRT* solution. The authors in [49] proposed a goal-driven navigation strategy that combines both global and local planners.

Although the above works have achieved remarkable performance in the navigation of planar vehicles, they cannot be adopted to navigate 3D UAVs. Compared to path (trajectory) planning for planar vehicles, planning for UAVs is more complex, not only because of the 3D environment but also because of the complex dynamic model and limited energy capacity of UAVs.

2.2. Path and Trajectory Planning for UAVs

In this section, we compare works on UAV path (trajectory) planning in detail, discuss the limitations of existing works, and further explain the motivation for this work. We compare the existing works in five aspects, including the feasibility of the dynamics, energy efficiency, long-trajectory orientation, the capability of avoiding dynamic obstacles, and the ability to escape from maze-like obstacles. The comparison is illustrated in Table 1.

Table 1. Comparison between existing works.

	Feasibility of the Dynamics	Energy Efficient	Long Trajectory	Maze	Dynamic Environment
[18,19]			✓	✓	✓
[20–24]	✓		✓	✓	✓
[27,29,35]	✓	✓			
[28,38],	✓	✓	✓		✓
[30,33,36,37]	✓	✓			✓
[41,42,50]	✓	✓	✓	✓	
this work	✓	✓	✓	✓	✓

2.2.1. Non-DRL Planning Methods

Ref. [18] proposed a Glow-worm Swarm Optimization (GSO) algorithm as a solution to UAV path planning in three-dimensional dynamic environments. Using GSO enables the UAV to avoid obstacles of different sizes and motions. Ref. [19] introduced some graph-based and sampling-based path-planning approaches and assessed them in a 3D

real-time platform with different complexities. Both works provide effective strategies for guiding UAVs over long trajectories and escaping maze-like obstacles, but they do not consider the feasibility of the dynamics.

More obstacle-avoidance studies have lately been carried out, taking into account both dynamic feasibility and replanning schemes. Ref. [20] designed a kinodynamic path-planning paradigm for multirotor flight that formulates a back-end refiner to improve the initial solution quality in dynamic environments. Ref. [21] proposed a trajectory replanning framework for limited-sensing quadrotor navigation tasks in dynamic and unknown environments. The authors in [22] presented multiple-RRT* as a local path finding technique that reduces planning time and applied iLQR to obtain a dynamically feasible and collision-free path. Ref. [23] proposed Bi-Risk-RRT, an efficient trajectory-planning method based on the risk algorithm and bidirectional search sampling. In [24], the uniform B-spline was adopted to represent smooth trajectories due to its highly strict convex hull property. However, these real-time kinodynamic approaches require more computation resources, and thus, it is challenging to implement them on resource-limited UAVs.

The authors in [50] proposed an interesting work to design optimal 3D trajectories for UAVs used to support wireless communication. Although the planning strategy is energy efficient, this work did not consider dynamic obstacles.

2.2.2. DRL-Based Planning Methods

With the development of techniques such as neural networks and deep learning, DRL shows great potential in solving navigation problems.

Ref. [27] developed a DRL solution for path following and obstacle avoidance, where a novel state space was defined with a short memory to solve the narrow field-of-view of LIDAR. Ref. [29] presented a Proximal Policy Optimization (PPO)-based method that can generate near-time-optimal trajectories through multiple gates for drone racing. Ref. [35] developed a fully distributed-control DRL-based solution for directing a group of UAVs flying around a target area as mobile base stations. However, none of these works are suitable for dynamic environments and long trajectories.

A DRL-based algorithm with extremely sparse rewards is proposed in [28] for UAV navigation in large-scale, complex scenarios. The authors adopted a sparse reward scheme instead of reward shaping to avoid suboptimality and to help the agent complete tasks in the early part of training. However, this work is not suitable for maze-like obstacles.

Ref. [30] proposed a fast-reacting navigation system for multirotor aerial robots in dynamic indoor situations. The agent is trained using a Deep Deterministic Policy Gradient (DDPG), and the reward function is designed using an Artificial Potential Field (APF). In [33], the reward function and the network layers' size were designed as an evolutionary automation for navigating in dynamic environments with moving obstacles. In [36,37], images captured by the monocular camera on board were used to evaluate obstacles' status, and then DRL was designed to develop the obstacle-avoidance policy for the UAV. However, due to the absence of global information, these DRL-based works are vulnerable to local minima and are not able to provide long-term navigation in maze-like scenarios.

The authors in [41,42] combined the DRL-based method and the global-planning method, using the DRL network to construct the RTT tree and generate a global navigation plan. These two works used the DRL to avoid obstacles and used the RTT tree to navigate the UAV to the destination. However, these works did not consider unpredictable obstacles.

According to the comparison between existing works, none of these works has achieved all five targets. Therefore, in this work, we aim to design an energy-efficient trajectory-planning method that can navigate the UAV over a long trajectory while avoiding unpredictable and maze-like obstacles.

3. Preliminaries

This section begins with an introduction to the Markov Decision Process (MDP) and DRL, then moves on to formulate the UAV navigation task.

3.1. MDP and DRL

MDP [51] serves as a modeling framework for decision making under uncertainty and is the foundation of DRL problems. MDP consists of a finite state set S, a finite action set A, the state transition probability $P^a_{ss'} = \mathrm{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$, the reward function $R^a_s = \mathrm{E}[R_t \mid S_t = s, A_t = a]$ ($\mathrm{E}[\cdot]$ represents the expected value of a given variable or function throughout the paper), and the discount factor $\gamma \in [0, 1]$. The policy $\pi(s) = \mathrm{P}[A_t = a \mid S_t = s]$ is the probability distribution of behaviors in the MDP and can direct the agent to choose actions based on the given states.

DRL is a learning approach that solves problems formulated as MDPs. Specifically, DRL aims to find the optimized policy using the MDP framework to define the interaction between the agent and the environment. During the training stage, the agent executes action a in current state s depending on policy π, and the environment provides a reward that drives the policy to update towards choosing more-valuable actions. DRL algorithms often employ deep neural networks to approximate the policy and value functions, which enables the learning of complex and high-dimensional state–action mappings. Therefore, DRL is suitable for solving sensor-based navigation problems.

In general, MDP provides the foundation for defining the problem, while DRL is a technique for finding the optimized policy in the context of the MDP.

3.2. UAV Navigation Task

The UAV navigation task is to control a UAV to reach a given goal location in an environment filled with unknown static obstacles based on range-sensor readings and other environmental information. It appears that navigating is a decision-making process, and it can be thought of as mapping from states to actions (as seen in Figure 2), which is suitable for being resolved using DRL. However, because the sensor range is finite, depending solely on sensors increases the risk of being trapped, which is a crucial problem to be addressed.

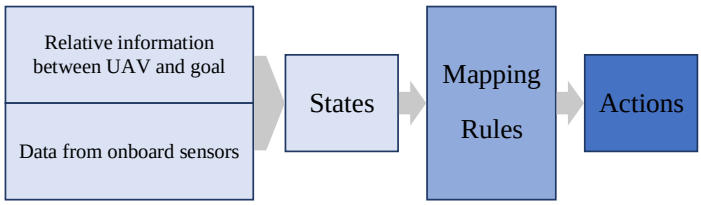


Figure 2. Mapping from states to actions.

4. The DGlobal Framework

In this section, we first take an overview of the DGlobal Framework and then give a more detailed account, including DRL settings, the DRL architecture, and the entire implementation process. The DGlobal Framework aims to address the following challenges of the UAV navigation problem.

(1) How do we navigate the UAV over a long trajectory and avoid obstacles? It is very hard for the UAV to find the destination itself since the long trajectory could be full of dynamic obstacles and traps.

(2) How do we accelerate training when the environment is rather complex? Training of the DRL module can be very slow when the solution space is rather large due to the complex training environment.

4.1. Overview

The overview of the DGlobal Framework is shown in Figure 3; it consists of the **Human-in-the-Loop-DRL (HL-DRL) training module** and the **global-planning module**. The HL-DRL training module intends to obtain a well-trained policy for UAV point-to-point collision-free flight, while the global-planning module aims to prevent the UAV from being trapped in complex environments. Compared to other local planners [18–20,22],

which need to recompute the trajectory during flight, a policy well-trained by DRL usually involves low computational cost and consumes less energy [35,52,53]. Thus, we adopt the DRL-based local planner instead of non-DRL planners.

The HL-DRL training module contains four sub-modules. First, the state space, action space, and reward function are well-designed to help the UAV better comprehend how to complete the task (①). Second, during the training process, DRL randomly selects $n/2$ tuples (*[state, action, next state, reward, done flag]*) from the demonstration replay buffer and the experience replay buffer and concatenates them to be a batch of n samples (②). In order to accelerate the training, we adopt the human-in-the-loop mechanism [43], and by introducing the demonstration replay buffer, HL-DRL is able to learn from both human experience and self-exploration. Third, the batch is then fed into an Actor–Critic network that is constructed elaborately to extract the features of the environment information more accurately (③). Forth, Twin Delayed Deep Deterministic Policy Gradients (TD3) [54], a DRL algorithm, is adopted to update the policy (④). Briefly, the TD3 algorithm has three major steps: (1) update the critic model network (Ⅰ–Ⅴ), (2) update the actor model network (Ⅵ), and (3) update all target networks (Ⅶ). The details of the TD3 algorithm are further introduced in Sections 4.4 and 4.5. The HL-DRL training module iterates the training process until it converges to an optimized policy.

The global-planning module generates a static, collision-free path based on previous knowledge, and then the RDP algorithm downsamples the path to a few key points that can guide the UAV to escape from maze-like situations. Furthermore, a novel goal-updating algorithm is designed to change the goal among waypoints, which improves the generalization of the proposed method.

Taken together, the global-planning module provides the UAV with some waypoints based on global information to avoid getting lost, and the well-trained policy obtained by the HL-DRL training module navigates the UAV between each two waypoints to deal with changes in the environment.

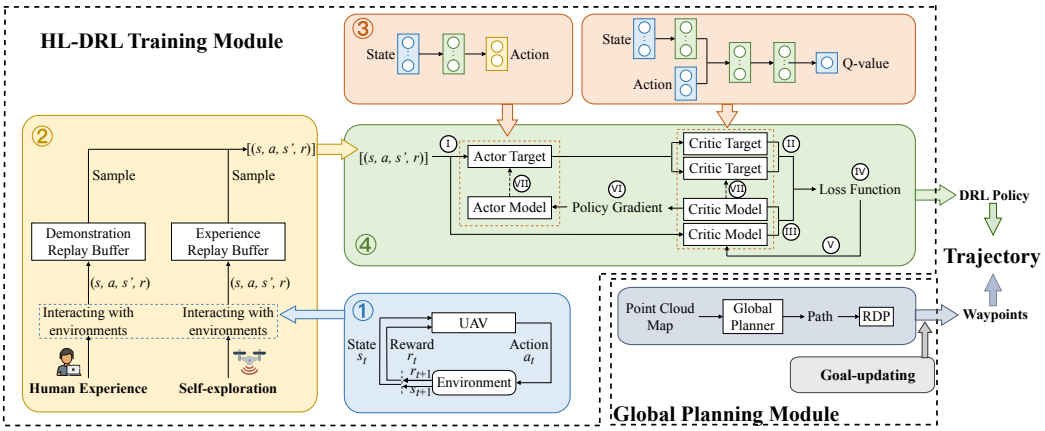


Figure 3. Overview of the DGlobal framework.

4.2. DRL Settings

4.2.1. State Space and Action Space

State Space. The information required for UAV navigation generally falls into two classes: one for reaching the target and another for avoiding collisions. First, we adopt the relative information between the goal and the UAV rather than the absolute state of the UAV to represent the state space in case the environment changes. As shown in Figure 4, (x, y) and $(x_{\text{goal}}, y_{\text{goal}})$ are the positions of the UAV and the goal, respectively. Thus, the relative position between them can be defined as $(x_{\text{goal}} - x, y_{\text{goal}} - y)$. Moreover, d_{goal} denotes the distance between the goal and the UAV, and α is the angle between the UAV velocity vector and the connecting line of the UAV and the goal. It is worth noting that these state variables are in the world frame. Second, in order to detect obstacles, we model a range sensor with

a detection angle of 360° and angular resolution of 0.5°, which corresponds to 720 rays. Figure 5a shows a UAV in a multi-obstacle environment and that it detects obstacles using the onboard range sensor; in addition, the histogram of laser scanning data is in Figure 5b. Since the scanning range of real sensors is limited by a finite detection distance, we define the saturated distance function of each ray, denoted ρ_i.

$$\rho_i = \begin{cases} l_{\max} & \text{if nothing was detected} \\ \rho_i & \text{else} \end{cases}, i = 1 \ldots 720 \tag{1}$$

As shown in Equation (1), if the length of ray i is greater than the sensing range $l_{\max}$, indicating that there are no obstacles within the detecting region, then the distance ρ_i is set to $l_{\max}$. Otherwise, if ray i hits an obstacle at point O, then ρ_i is the distance of point O from the sensor; i ranges from 1 to 720 because the range sensor has 720 samples (or rays). Furthermore, it is vital to normalize state values with different units and scales before inputting them into networks, and such pre-processing is conducive to extracting features and speeding up the convergence of DRL. Therefore, the state space $\boldsymbol{s}$ is formulated as

$$\boldsymbol{s} = [\frac{x_{goal} - x}{l_X}, \frac{y_{goal} - y}{l_Y}, \frac{d_{goal}}{(l_X^2 + l_Y^2)^{\frac{1}{2}}}, \alpha, \rho_1, \ldots, \rho_{720}]^{\mathrm{T}} \tag{2}$$

where l_X is the x-axis length of the environment, and l_Y is the y-axis length.

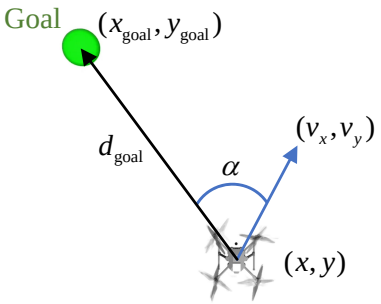
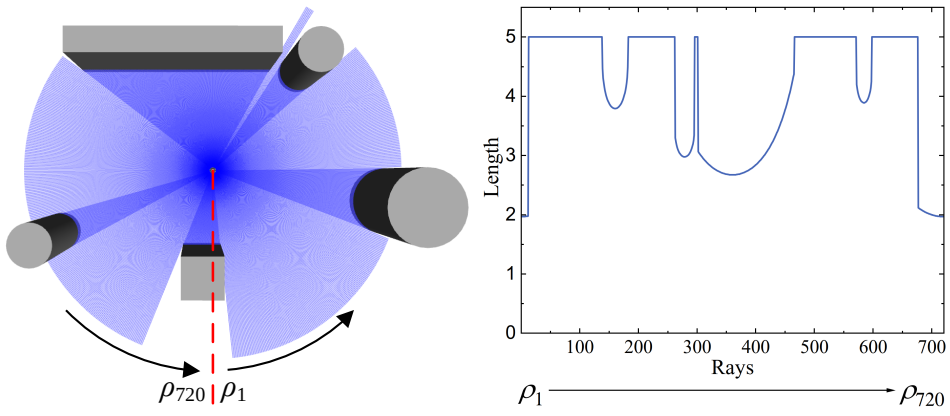


Figure 4. Illustration of the relative information between the goal and the UAV.



(**a**) (**b**)

Figure 5. Illustration of laser range scanning. (**a**) Range scans perceived with an onboard range sensor. (**b**) Histogram of scanning data sets.

Action Space. The 'XY_VEL_Z_POS' mode (velocity control in xy-plane and position control in z-direction) is taken to model the dynamics of UAVs and is commonly used for

flight control software such as PX4. In this paper, the UAV's altitude remains constant except during takeoff and landing; then the UAV dynamics are formulated as

$$\begin{aligned} x_{t+1} &= x_t + v_x \Delta t, \\ y_{t+1} &= y_t + v_y \Delta t. \end{aligned} \tag{3}$$

Thus, we use $\boldsymbol{a} = [\frac{v_x}{v_{\max}}, \frac{v_y}{v_{\max}}]^{\mathrm{T}}$, ranging from -1 to 1, to represent the action space, (v_x, v_y) is the velocity of the UAV, $v_{\max}$ is the maximum speed value, (x_t, y_t) represents the planar position of the UAV at moment t, and Δt is the time interval.

4.2.2. Reward Design and Termination Conditions

Reward design is a fundamental property of DRL and has a substantial impact on the results of DRL training. The DRL agent receives a reward r calculated using the reward function after taking action a in the current state s. According to Section 3.1, r greatly affects the value function $Q(s, a)$, which is the evaluation of the state–action pair's quality. Therefore, a well-designed reward function can guide DRL to generate a safe and effective path to the destination. In this paper, we design the reward function based on our domain knowledge of the navigation problem and introduce it under three conditions.

C1: Colliding with obstacles. If the minimum value among all ρ_i is less than or equal to the radius of the UAV, it means the UAV has hit something, denoted as

$$\min_{1 \le i \le 720} \rho_i \le \frac{l_{\mathrm{UAV}}}{2} \tag{4}$$

where l_{UAV} is the diameter of the UAV. For the UAV, colliding with obstacles could be fatal. Thus, in order to keep the UAV away from obstacles, we penalize it with a negative constant and use C1 as a termination condition.

C2: Reaching the goal. If d_{goal} is less than or equal to the sum of the UAV's radius and the goal's radius, it means the UAV has reached the goal, represented as

$$d_{\mathrm{goal}} \le \frac{l_{\mathrm{goal}} + l_{\mathrm{UAV}}}{2} \tag{5}$$

where l_{goal} is the diameter of the goal point. Therefore, to teach the UAV to reach the target, we give it a positive constant as a reward and consider C2 to be another termination condition.

C3: Neither collision nor task completion. If the UAV has neither hit an obstacle nor completed the task at the current moment, it receives a reward expressed as

$$r = r_{\mathrm{obs}} + r_{\mathrm{dis}} + r_{\mathrm{angle}} + r_{\mathrm{step}}. \tag{6}$$

First, to make the UAV maintain a certain distance from obstacles, we design the obstacle penalty as

$$r_{\mathrm{obs}} = k_1 \max\{0, d_{\mathrm{safe}} - \min_{1 \le i \le 720} \rho_i\} \tag{7}$$

where k_1 is a negative constant and d_{safe} is a constant denoting the safe distance between the UAV and obstacles. If the UAV is less than d_{safe} away from the nearest obstacle, it suffers a penalty of r_{obs}; otherwise, the obstacle is not a threat to it, and the UAV receives a zero. Second, to reduce the distance between the UAV and the goal point, d_{goal} is used to form the distance penalty

$$r_{\mathrm{dis}} = k_2 \frac{d_{\mathrm{goal}}}{(l_X^2 + l_Y^2)^{\frac{1}{2}}} \tag{8}$$

where k_2 is a negative constant. Third, to assist the UAV in moving towards the goal point, α is taken to build the angle penalty

$$r_{\text{angle}} = k_3 \frac{d_{\text{goal}}}{(l_X^2 + l_Y^2)^{\frac{1}{2}}} \cdot \frac{\alpha}{\pi} \tag{9}$$

where k_3 is a negative constant. Last, to make the UAV reach the target as soon as possible, a negative constant r_{step} is given to the UAV at each step.

4.3. Replay Buffer Design

Experience replay is a technique that can make the HL-DRL training process more stable. In Actor–Critic network-based DRL algorithms, the critic network is used to approximate the state–action value $Q(s, a)$. When updating the network, stochastic gradient descent optimization requires the training data to be independent and identically distributed (i.i.d.). Therefore, the authors in [55] designed a large buffer, also known as an experience replay buffer, that includes a set of experience tuples. The fixed-size buffer adds the latest tuple *(state, action, next state, reward, done flag)* to its end whenever the agent takes a step. After the buffer reaches the maximum size, a batch of experience tuples is sampled randomly from the buffer to update the network. As a result, experience replay can keep the training process from oscillating or diverging by breaking the correlation between the training data.

In this paper, we not only adopt the experience replay buffer but involve the "human tutor" and design a new buffer called the demonstration replay buffer. Each time HL-DRL updates, we sample some tuples from each of the two buffers and concatenate these tuples before inputting them into the networks. The following is a detailed description of the demonstration replay buffer.

The aim of demonstration replay is to speed up the convergence of training. We choose Gazebo as a simulator for HL-DRL training in this paper because it offers the ability to efficiently and accurately simulate robots in complex environments, closing the simulation-to-reality gap to a large extent. However, this feature makes it almost impossible for the UAV to find the target by exploring from scratch. Without any successful experience, it is hard for a DRL module to obtain the optimized policy. Therefore, we hire a group of human players to operate the UAV to the goal and store transitions *(state, action, next state, reward, done flag)* in the demonstration replay buffer. The data in the demonstration replay buffer can help the UAV rapidly realize how to complete the task during the pivotal initial phase of training, and since the demonstration replay buffer is prepared before the start of training, human involvement does not slow the whole training process. As shown in Figure 6, according to observation and the current state s_t in Gazebo, the human player uses the keyboard to control the UAV, and the operation is logged as a_t. Then, the environment returns the next time state s_{t+1}, reward r_t, and the done flag; s_{t+1} is obtained by reading the UAV state from Gazebo directly, and r_t is calculated using the same reward function introduced in Section 4.2.2. For every step, the current tuple $(s_t, a_t, s_{t+1}, r_t, flag)$ is saved in the demonstration replay buffer, but if a collision occurs, all data in this episode are removed. It should be noted that we collect the demonstrations from different human players to exclude the influence of the human factor and to maintain data objectivity.

In general, the demonstration replay largely overcomes the sample inefficiency issue in DRL by incorporating human demonstrations and thus makes DRL converge faster.

4.4. Network Structure

The Actor–Critic network architecture is applied to build the DRL algorithm in this paper. There are two reasons why the Actor–Critic network structure is adopted to design DGlobal. First, although more-advanced structures such as residual modules and GRU units are available, we focus on validating the effectiveness of our proposed framework. Therefore, using a simpler architecture such as an Actor–Critic network is more efficient.

Second, adopting residual modules or GRU units may introduce potential drawbacks such as overfitting, difficulty in tuning hyperparameters, and increased complexity. In particular, as our framework is designed for deployment on a UAV, increased complexity could pose challenges to the limited computational and energy resources onboard.

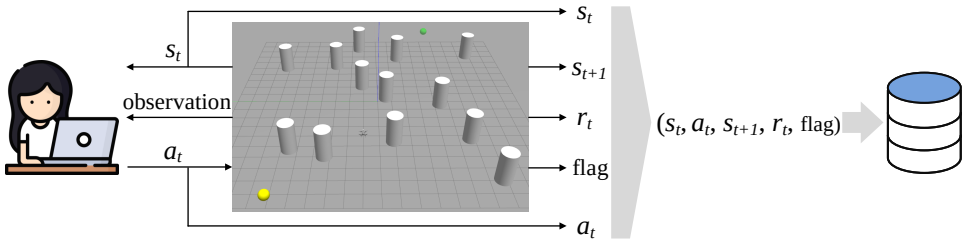


Figure 6. Demonstration collection process.

As we mentioned in Section 3.1, the actor maps the state into the action to control the agent, and the critic then provides evaluative feedback about the action based on its reward. Figure 7 shows the structures of the actor network and the critic network. The actor network is composed of two fully connected neural network layers, which are followed by a Rectified Linear Unit (ReLU) activation function and a hyperbolic tangent (tanh) activation function, respectively. The first layer receives the 724-dimensional vector s_t consisting of the 4-dimensional relative goal information and the 720-dimensional laser information and outputs a 256-dimensional vector. The second layer takes the vector as an input and maps it to the 2-dimensional vector a_t that serves as the velocity factor. For the critic network, the input s_t is first mapped to a 1024-dimensional vector by the fully connected layers with ReLU and then merged with a_t as a 1026-dimensional vector. After two layers with ReLU and one fully connected layer, the critic outputs the state–action value $Q(s, a)$.

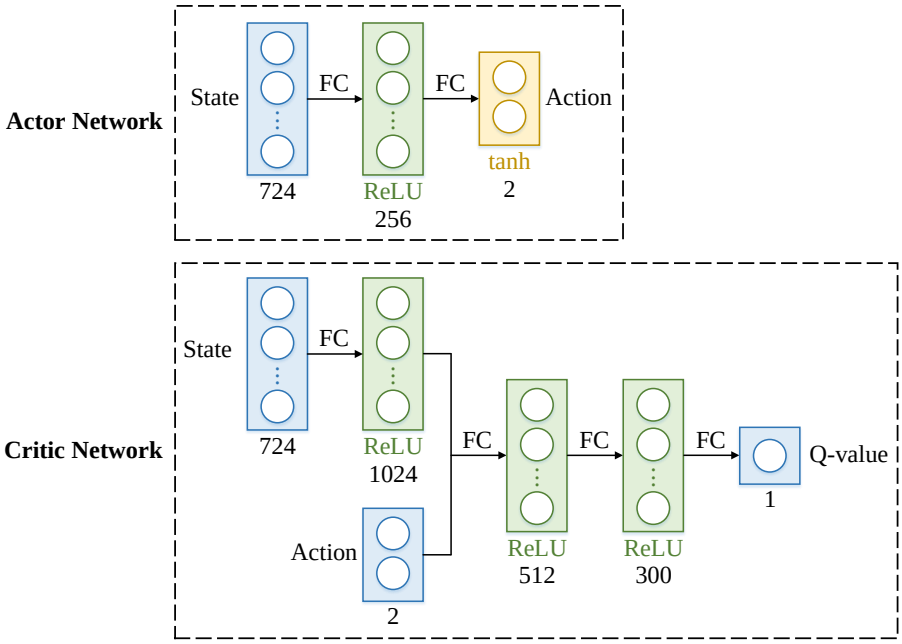


Figure 7. Architecture of the Actor–Critic network.

4.5. Training Process

We use TD3 [54] to address the UAV navigation problem in this paper. Although other DRL algorithms can also be adopted in the DGlobal framework, we still choose TD3 since it is a state-of-art DRL algorithm and has been widely used to train navigation policies for unmanned vehicles [49,56,57]. TD3 is also based on the Actor–Critic architecture, similar to

DDPG; however, it offers some improvements over DDPG, as shown in Figure 8. The actor target network receives s' and outputs a', as in DDPG, but the difference is that TD3 adds a little disturbance to a' before inputting it into the critic target network (①).

$$a' = \text{clip}(\pi(s') + N, a_{\min}, a_{\max}) \tag{10}$$

N denotes the clipped noise and $a_{\min}$ and $a_{\max}$ are the boundary values of actions. This behavior reduces the negative impact of $Q(s', a')$ evaluation errors by smoothing out $Q(s', a')$ in the action dimension. Next, after obtaining a', TD3 uses two critic target networks to learn $Q(s', a')$ instead of one and chooses the smaller Q-value to calculate the loss function L. For critic model networks, TD3 learns two Q-values as well (②). Therefore, the loss function in TD3 can be denoted as

$$L = \sum_{j=1,2} \mathrm{E}\left[\left(r + \gamma \min_{i=1,2} Q_i'(s', a') - Q_j(s, a)\right)^2\right]. \tag{11}$$

Using the smaller Q-value to calculate the target value in L helps mitigate overestimation in the $Q(s, a)$ function. Moreover, the actor updates in the direction of a greater Q-value, which is almost the same as DDPG. However, the actor in TD3 updates with a lower frequency than the critic does, and it recommends updating the actor when the critic has been updated twice (③). This restrains the fluctuation that often appears in DDPG because it is better to update the policy after the value function has been accurately estimated.

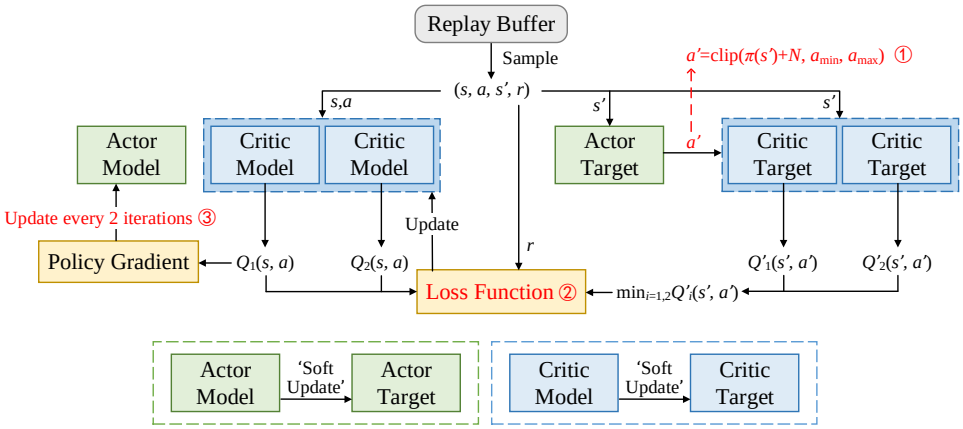


Figure 8. TD3 algorithm.

Overall, TD3 trains the DRL agent in that way until it converges into an optimized policy. However, if the UAV performs the task only relying on the DRL policy, it could get stuck in complex environments because of its limited sensing capability and the lack of global knowledge. In order to address this concern, we integrate the DRL policy with the global planner, and the details are introduced in the following part.

4.6. Entire Implementation Process

To make up for deficiencies in DRL-based motion planning, we propose a novel navigation method that incorporates global planning, as illustrated in Algorithm 1.

Phase I (line 1): The path-planning algorithm A* [10] is adopted to generate a collision-free path $\mathcal{P}$ based on the previous point cloud map and the positions of the start and the goal.

Phase II (line 2): We use RDP [58], a polyline simplification algorithm, to produce a simplified polyline that has fewer points than path $\mathcal{P}$ but still keeps its characteristics. These points are stored in set $\mathbb{P}$ in sequence and are used as waypoints in the following. In $\mathbb{P}$, the first element $\mathbb{P}[0]$ is the start point and the last element is the goal point.

Phase III (lines 3–10): The well-trained policy $\pi(\cdot)$ is used to navigate the UAV along waypoints. First, we initialize the goal to $\mathbb{P}[1]$. Then, the policy generates action a based on current state s that includes information regarding the current to-be-reached waypoint, and the UAV performs the action. Next, we update the goal (the details are described in Algorithm 2). The algorithm terminates when the goal is reached, a collision occurs, or a fixed number of steps is exhausted. If the UAV reaches the goal, a dynamically feasible and collision-free motion plan can be returned.

Algorithm 1: Proposed method

Input: p_{start}: start point's position, p_{goal}: goal point's position, PCM: point cloud map, $\pi(\cdot)$: well-trained DRL policy.
Output: Motion plan.

```
1 P = A*(p_start, p_goal, PCM);
2 ℙ = RDP(P);
3 Initialize: i_ℙ = 1, p_goal = ℙ[1];
4 while termination condition not met do
5     s = getEnv(i_ℙ, p_goal);
6     a = π(s);
7     perform a;
8     i_ℙ, p_goal = updateGoal();
9 return Motion plan
```

The goal-updating algorithm and corresponding extension methods are given in the following subsections.

Algorithm 2: Updating the goal

Input: $\mathbb{P}$: set of waypoints, $i_{\mathbb{P}}$: index of waypoints, p_{UAV}: UAV's position, l_{UAV}: UAV's diameter, l_{waypoint}: waypoint's diameter, ρ_i: laser scanning data.
Output: $i_{\mathbb{P}}$, p_{goal}.

```
1 i = i_ℙ;
2 while i < len(ℙ) − 1 do
3     d_ℙ[i] = ‖ℙ[i] − p_UAV‖_2;
4     if d_ℙ[i] ≤ (l_waypoint + l_UAV)/2, then i_ℙ = i + 1;
5     i = i + 1;
6 d_ℙ[i_ℙ] = ‖ℙ[i_ℙ] − p_UAV‖_2;
7 if d_ℙ[i_ℙ] ≤ (l_waypoint + l_UAV)/2 + ε and min_{1≤i≤720} ρ_i ≤ l_UAV/2 + ε, then i_ℙ = i_ℙ + 1;
8 p_goal = ℙ[i_ℙ];
9 return i_ℙ, p_goal
```

4.6.1. Goal-Updating Algorithm

While the updated map strategy can assist the UAV in detecting changes in the environment, when numerous waypoints are blocked by unpredictable obstacles, the UAV may have to rely solely on a local planner (the policy learned by the DRL) to navigate through the maze-like environment. In such cases, the UAV has to take risks and could become trapped in the maze. Hence, using a goal-updating algorithm to renew global waypoints is rather necessary. Algorithm 2 gives the details of updating the goal, and it has three major steps.

Step 1 (lines 1–5). We check whether the UAV has been within range of the i-th ($i \geq i_{\mathbb{P}}$) waypoint. If so, we directly set the goal to the $(i+1)$-th waypoint, which prevents the UAV from flying unnecessary distances. The index i is initialized to $i_{\mathbb{P}}$ (line 1), and we calculate the distance $d_{\mathbb{P}[i]}$ between the UAV and waypoint $\mathbb{P}[i]$ (line 3). If $d_{\mathbb{P}[i]}$ is less than or equal to

the sum of the UAV's radius and the waypoint's radius, then the UAV is deemed to have reached $\mathbb{P}[i]$; therefore, we change the goal to waypoint $\mathbb{P}[i+1]$ (line 4). Incidentally, we expand the waypoint into a region with its coordinate as the center and l_{waypoint} as the diameter because it can help the UAV handle changes in the environment. This process repeats until all remaining waypoints have been examined (line 2). As a result, we have an updated index $i_{\mathbb{P}}$.

Step 2 (lines 6–7). If **Step 1** did not return a new goal, it implies that the UAV has not reached the current goal or any subsequent waypoint. In this case, we need to evaluate whether the UAV is unable to approach the goal because random obstacles were generated within the range of the waypoint. If so, we relax the criterion for the UAV to reach the waypoint. To be specific, if the UAV is less than ϵ away from the edge of the waypoint's region and detects that the distance to the nearest obstacle is less than ϵ, then we consider that the UAV has arrived at the waypoint and we update the goal (line 7); ϵ is chosen according to the environmental setting.

Step 3 (lines 8–9). The goal point's position p_{goal} can be obtained based on $i_{\mathbb{P}}$, and both of them are returned.

The goal-updating algorithm only renews the waypoints during flight of the UAV. Between every two waypoints, the UAV is navigated by the DRL policy, and its speed can be dynamically changed. Figure 9 gives an example to illustrate the proposed method: (a) The A* algorithm plans the collision-free path (the blue dotted line) based on the previous information, and then the RDP algorithm simplifies the path and generates the waypoints (green circles). (b) The UAV departs from the start point and flies along the waypoints. (c) After the UAV has arrived at W2, the goal is switched to W3; however, the DRL policy navigates the UAV to the other side to stay away from obstacles. (d) Then, the UAV reaches W4. According to Algorithm 2 (lines 2–5), the goal is set to W5, although the UAV has not reached W3. (e) After the UAV has arrived at W6, the goal is changed to W7, where a new obstacle appears. (f) Therefore, according to Algorithm 2 (line 7), we relax the criterion for the UAV reaching W7 by expanding the range of W7. (g) Next, the UAV passes W8 and finally reaches the goal.

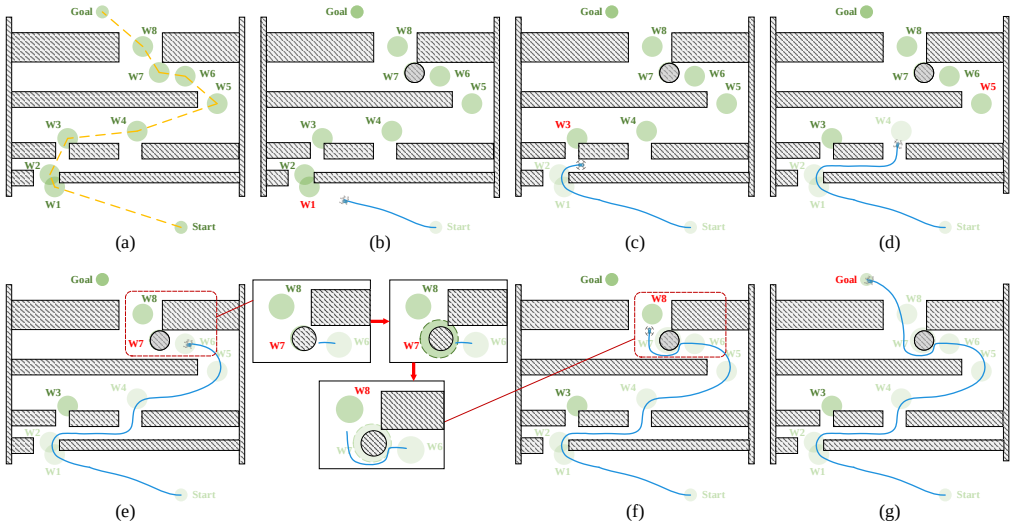


Figure 9. Illustration of the proposed method: the yellow dotted line is the path planned by A*, the green circles are waypoints generated by RDP, and the blue line is the trajectory of the UAV).

4.6.2. Algorithm Extension

During Phase I of Algorithm 1, the A* algorithm is employed to generate a collision-free path $\mathbb{P}$, which serves as the basis for navigation and goal-updating. However, the presence of dynamic obstacles can cause multiple waypoints in $\mathbb{P}$ to become invalid and blocked, which can lead the UAV to the wrong locations and can lead it to ultimately

become trapped in maze-like obstacles. Therefore, we adopt the following extensions to enhance the robustness of the proposed algorithms.

First, in Algorithm 1, the A* algorithm generates a set of n paths $\mathbf{P} = \{\mathbb{P}_1, \mathbb{P}_2, ..., \mathbb{P}_n\}$, and each path is simplified by the RDP algorithm. For each two arbitrary paths $\mathbb{P}_j, \mathbb{P}_k$ $(j, k \in [1, n])$, they can have overlaps.

The second extension is in the goal-updating algorithm. Suppose that the UAV has arrived at the i-th waypoint $i_{\mathbb{P}_j}$ of path $\mathbb{P}_j$ and its next goal is the $(i+1)$-th waypoint $(i+1)_{\mathbb{P}_j}$. If the UAV cannot arrive at $(i+1)_{\mathbb{P}_j}$ within a certain time $\tau(i_{\mathbb{P}_j}, (i+1)_{\mathbb{P}_j})$,

$$\tau(i_{\mathbb{P}_j}, (i+1)_{\mathbb{P}_j}) = \sigma\left(\frac{distance(i_{\mathbb{P}_j}, (i+1)_{\mathbb{P}_j})}{v_{UAV}}\right) \qquad \sigma \geq 1 \textit{ is a user-input parameter},$$

then $(i+1)_{\mathbb{P}_j}$ is marked as an unreachable waypoint and every path containing $(i+1)_{\mathbb{P}_j}$ is removed from $\mathbf{P}$. After that, the UAV selects a new waypoint $l_{\mathbb{P}_k}$ from path $\mathbb{P}_k$ as its new goal, where $l_{\mathbb{P}_k}$ and $\mathbb{P}_k$ satisfy that $\mathbb{P}_k$ belongs to the updated $\mathbf{P}$ and $d_{\mathbb{P}_k[l]} = \min\{d_{\mathbb{P}[l']} | l' \in \bigcup_{\mathbb{P} \in \mathbf{P}} \mathbb{P}\}$.

Table 2 lists the major symbols used in this section.

Table 2. Symbol table.

Symbol	Description
d_{goal}	Distance between the goal and the UAV.
α	Angle between the UAV velocity vector and the connecting line of the UAV and the goal.
ρ_i	Length of the laser ray i, denoting the distance from the UAV to surrounding obstacles.
$\boldsymbol{s}$	State space.
$\boldsymbol{a}$	Action space.
r	Reward function.
r_{obs}	Penalty related to obstacles.
r_{dis}	Penalty related to d_{goal}.
r_{angle}	Penalty related to α.
r_{step}	Penalty given to the UAV at each step.
$\pi(\cdot)$	DRL policy.
$\mathcal{P}$	Path generated by A* algorithm.
$\mathbb{P}$	Set of waypoints generated by RDP algorithm.
$i_{\mathbb{P}}$	Index of waypoints in $\mathbb{P}$.
$d_{\mathbb{P}[i]}$	Distance between the UAV and the waypoint $\mathbb{P}[i]$.
ϵ	Constant used to expand the region of the waypoint.

5. Simulation Results

In this section, experiments are carried out to evaluate the performance of the method proposed in this paper. First, we introduce the simulation setup and show the training results. Then, we test the proposed method in different situations and analyze the testing results. Last, we prove the effectiveness of the goal-updating algorithm (Algorithm 2). In addition, we implement all algorithms on a workstation with an 11th-generation Intel i7 Processor, NVIDIA GeForce RTX 3060 GPU, and Ubuntu 20.04 system.

5.1. Simulation Setup and Training Results

We use Gazebo in combination with ROS as the simulator, which efficiently and accurately simulates the dynamics of the UAV and largely closes the simulation-to-reality gap. Figure 10 shows the training environment. The yellow ball and the green ball, respectively, indicate the start and the goal points, and any obstacles are represented by grey cylinders in this configuration. As mentioned in Section 4.2.1, the UAV is equipped

with a range sensor with a 360° field of view and a range of 0.08 m to 5 m. We model the dynamics of the UAV using the PX4 'XY_VEL_Z_POS' mode, which controls the velocity in the xy-plane and the position in the z-direction. As our simulation is intended to test the effectiveness of our proposed method in navigating the UAV through long trajectories with maze-like environments, we fix the height of the UAV at 2 m to prevent it from circumventing obstacles by adjusting its height. At the start of each flight, the UAV ascends to a height of 2 m, and upon reaching its destination, it descends to the ground for landing. During the flight, the UAV's speed can be dynamically changed within a maximum speed limit of 2 m/s.

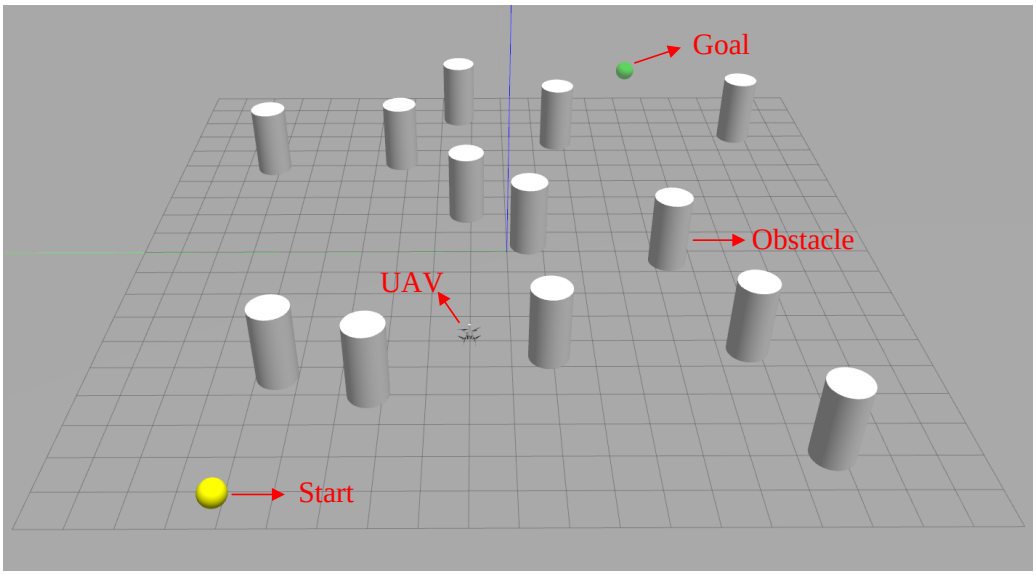


Figure 10. Training environment.

At the beginning of each training episode, the weights of the networks are initialized at random. The start and the goal are randomly generated within certain areas, and obstacles are randomly placed within the remaining area. The episode ends if the UAV reaches the goal, a collision occurs, or the steps reach the maximum number. The training parameters are listed in Table 3 and are chosen based on a combination of domain knowledge, empirical observations, and best practices in TD3 [54]. It is worth noting that the setting of training parameters can have a significant impact on the training process and performance [59]. Therefore, we perform preliminary experiments and hyperparameter tuning to select the parameters that provided stable learning and good performance for our specific problem.

To show the effectiveness of demonstration replay in this paper, we train TD3 without demonstration replay and TD3 with different ratios of demonstration replay batch size to experience replay batch size for comparison. The agent in TD3 with demonstration replay can learn from human experience and self-exploration, whereas the agent in TD3 without demonstration replay only learns by exploring from scratch. Multiple training experiments are carried out with different initialization seeds, and Figure 11 shows the reward curves during the training process from one of the multiple training runs. We smooth curves using the average-filtering method to observe the trend, denoted as

$$g(x) = \frac{1}{M} \sum_{i=-M/2}^{M/2} f(x+i) \tag{12}$$

where $f(\cdot)$ is the original curve and $g(\cdot)$ is the smoothed one. M represents the width of the smoothing window. At the beginning of training, the UAV explores the environments at random, so the reward of each episode is quite low. As training continues, the policy updates and the exploration noise is reduced. Therefore, the UAV receives a greater

reward value by making full use of the policy. As we can see, the curve of TD3 without demonstration replay continues to fluctuate a great deal and does not converge until training ends. The reason is that the agent in TD3 without demonstration replay can only learn from self-exploration, which rarely contains positive examples; thus, the agent constantly learns from negative examples, leading to the slow convergence problem. When the ratio of demonstration replay batch size to experience replay batch size is 1, the reward curve converges around the 700th episode. This is because demonstration replay provides numerous effective examples for the UAV to learn. Moreover, a balanced proportion between human demonstrations and the agent's own experiences enables effective learning and efficient convergence. When the ratio is 1/3, where experience replay data have a larger share in the training samples, the reward curve shows a converging trend, but the convergence speed is significantly slower. This indicates that relying too much on the agent's own experiences may not fully exploit the benefits of human demonstrations, potentially leading to a slower learning process. When the ratio is 3, the demonstration replay data occupy a large proportion of the training samples. It can be seen that the training results are even worse than those obtained by TD3 without demonstration replay. The cause is that the agent overly relies on human demonstrations, which are biased to some extent. This implies the importance of balancing the agent's own experiences with human demonstrations to avoid overfitting or being restricted by the limitations of the demonstration data.

Table 3. Training parameters.

Parameter	Value
Max Episodes	1000
Max Steps	1000
Discount Factor γ	0.99
Soft Update Factor	0.01
Critic Learning Rate	10^{-5}
Actor Learning Rate	10^{-5}
Delay Steps	2
Policy Noise	0.2
Noise Bound	0.5
Experience Replay Buffer Size	10^4
Demonstration Replay Buffer Size	2000
Experience Replay Batch Size	128
Demonstration Replay Batch Size	128
Safe Distance to Obstacles d_{safe}	1 m
Max Sensing Range $l_{\max}$	5 m
Max Speed Value of the UAV $v_{\max}$	2 m/s
Diameter of the UAV l_{UAV}	0.4 m

In the following sections, we evaluate the performance of the DGlobal in various environments with and without unpredictable obstacles.

5.2. Navigation in Environments without Unpredictable Obstacles

To test the performance of navigating in different situations, we build the following six types of static environments: **Env1**, **Env2**, and **Env3** are all 20 m $\times$ 20 m environments, where Env1 includes some cylinder-shaped obstacles, Env2 has a trap, and Env3 is a labyrinth. **Env4** is a 20 m $\times$ 100 m corridor that is full of obstacles. **Env5** and **Env6** are 100 m $\times$ 100 m environments full of barriers and mazes.

The following two existing algorithms are implemented in these environments to compare to the DGlobal method.

(1) Local Astar (LAS for short). LAS generates safe paths using the local information obtained from laser range scanning. The LAS adopted in this simulation is implemented by an open-source project [60].

(2) TD3-based navigation (TD3 for short). TD3 [54] uses a well-trained policy to generate actions according to laser scanning data and is real-time online planning.

The DGlobal method uses both the result of global A* and the well-trained policy of DRL to plan in real-time, as elaborated in Section 4.6.

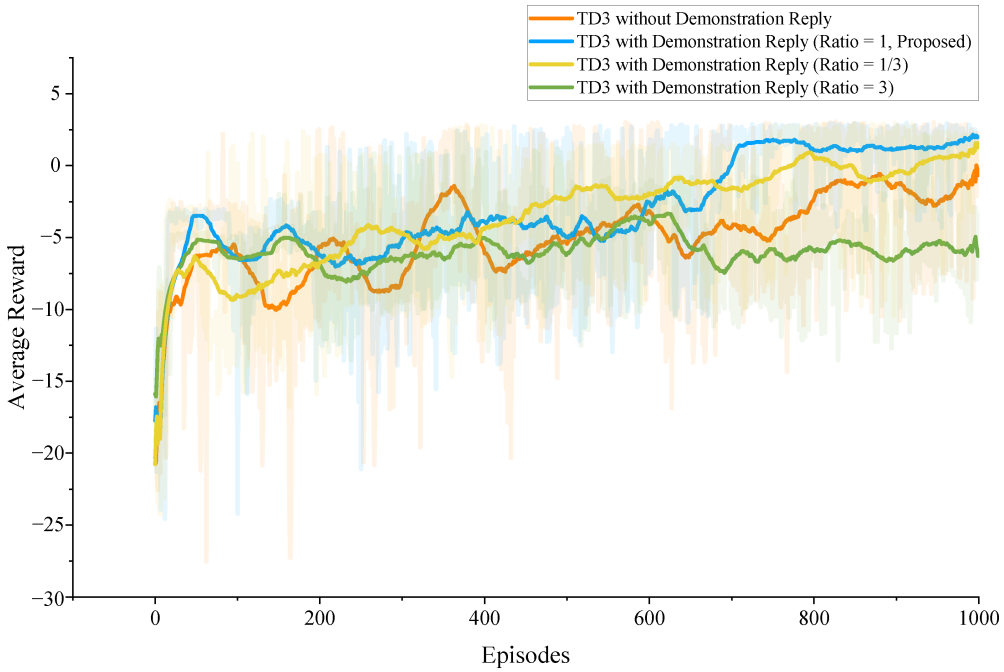


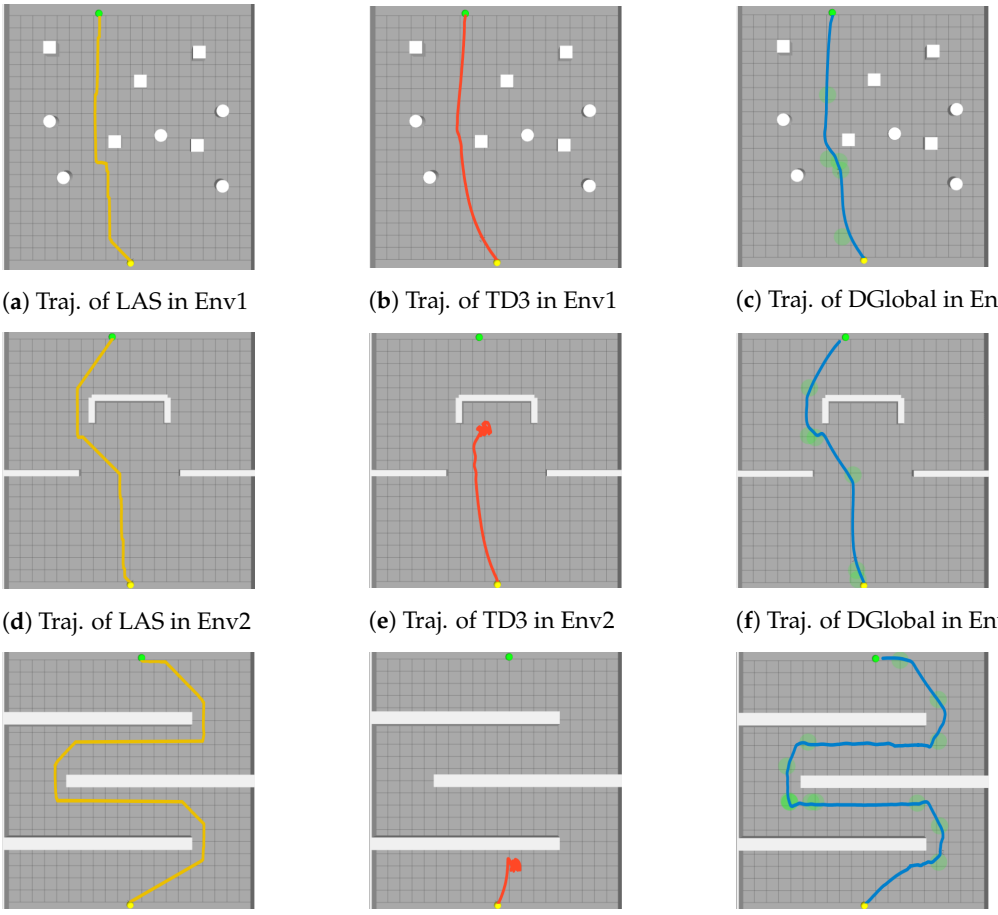
Figure 11. Convergence curves of rewards obtained from the training environment.

The following two subsections introduce the performance of these three algorithms in Env1–Env6. For each environment, the algorithms are executed multiple times with different start and goal points as inputs. To ensure a fair comparison, the generation seed is fixed for each round, guaranteeing that all three methods are carried out in the same environmental configuration. We employ the metric "**average flight time**" as an indicator of the average energy consumption across multiple test experiments, as longer flight durations intuitively correlate with increased energy consumption. It is worth mentioning that the global planner calculation is not taken into the "average flight time" of the UAV in DGlobal because the global planner is part of the preprocessing phase, which is analogous to the DRL training process. Additionally, for the same scenario, the global planner does not need to recompute at the beginning of each navigation since DGlobal can effectively handle random obstacles that may appear (which will be discussed and analyzed in Section 5.3).

5.2.1. Navigation in Env1, Env2, and Env3

Typical cases of navigation results in Env1–Env3 are illustrated in Figure 12, and Figure 13 shows the average flight time of these three algorithms under different environments. All methods successfully planned collision-free trajectories in Env1, as seen in Figure 12a–c. Moreover, TD3, which does not consider global information, has almost the same flight time as DGlobal, indicating that the DRL policy is efficient when navigating the UAV in relatively simple environments. In Env2, as shown in Figure 12d–f, LAS found a safe path due to its completeness and optimality. The UAV using TD3 lost its way because of the limited sensing range and lack of global knowledge, whereas the UAV using DGlobal finally took 56.3 s to reach the goal thanks to waypoint guidance. Similar results are obtained in Env3, as shown in Figure 12g–i. The UAV navigated by TD3 was trapped in the U-shaped obstacle; however, DGlobal avoided this situation by incorporating global

planning. Consequently, compared to TD3, DGlobal can prevent a UAV with limited sensing ability from being stuck in maze-like environments, and compared to LAS, DGlobal can navigate the UAV to the goal point as soon as possible.



(**a**) Traj. of LAS in Env1 (**b**) Traj. of TD3 in Env1 (**c**) Traj. of DGlobal in Env1

(**d**) Traj. of LAS in Env2 (**e**) Traj. of TD3 in Env2 (**f**) Traj. of DGlobal in Env2

(**g**) Traj. of LAS in Env3 (**h**) Traj. of TD3 in Env3 (**i**) Traj. of DGlobal in Env3

Figure 12. Typical cases of navigation results using LAS, TD3, and DGloal in Env1, Env2, and Env3 (Traj. is short for trajectory).

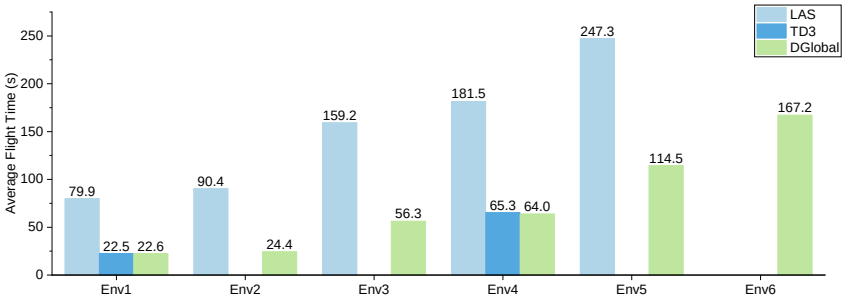
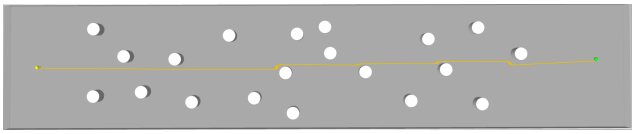


Figure 13. Average flight times of the UAV using LAS, TD3, and DGlobal in Env1–Env6.

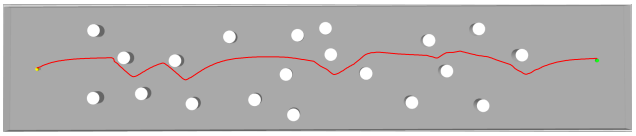
5.2.2. Navigation in Env4 (Corridor-like Environment)

Env4 is adopted to evaluate the performance of the DGlobal method in a long trajectory scenario where the UAV needs to fly through a corridor to its destination. We test all three algorithms in Env4 plenty of times by changing the number of obstacles. Figure 14 illustrates typical cases of navigation results, and Figure 13 presents the flight time. The experimental results show that all methods successfully planned collision-free trajectories in Env4. Additionally, similar to Env1, the flight time of TD3 and DGlobal are quite close, which demonstrates the effectiveness of TD3 in non-maze-like environments. Although the flight path of LAS is shorter than that of TD3 and DGlobal, it spends the longest time. The

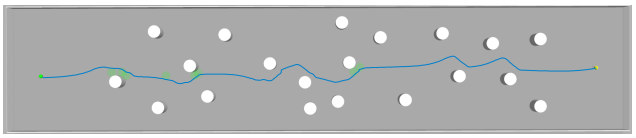
reason is that the LAS algorithm always needs to hover and calculate the next waypoint, which is time-consuming. This group of simulations implies that TD3 and DGlobal can save more energy than LAS in a corridor-like environment.



(**a**) Traj. of LAS in Env4



(**b**) Traj. of TD3 in Env4



(**c**) Traj. of DGlobal in Env4

Figure 14. Typical cases of navigation results using LAS, TD3, and DGloal in Env4 (Traj. is short for trajectory).

5.2.3. Navigation in Env5 and Env6 (Maze-like Environments)

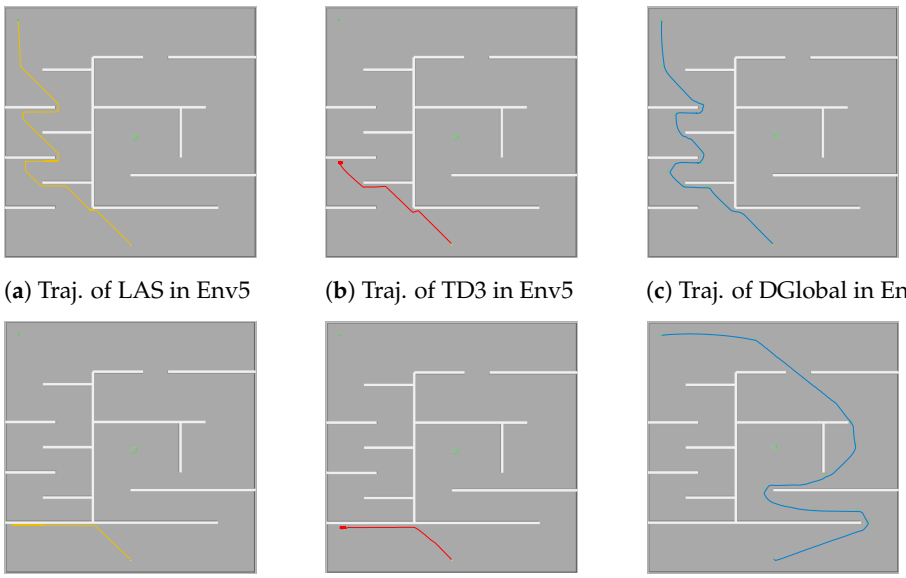
Env5 and Env6 are two 100 m × 100 m complex, maze-like environments. In Env5, there are two ways to arrive at the destination, while in Env6, the UAV can only get to the destination by flying through the right side of the maze. Typical cases of navigation results are shown in Figure 15, and the average flight time of TD3, LAS, and DGlobal are illustrated in Figure 13. According to Figure 15a–c, we can see that the TD3 method cannot navigate the UAV to the destination in Env5. As illustrated in Figure 15b, the UAV is trapped in the maze. As discussed in Section 1, navigation in a maze-like environment is challenging for the DRL-based method. Although the DGlobal method is also based on the DRL, it can get out of the maze by following the waypoints obtained from the global A*. On the other hand, the well-trained flight policy can also help the UAV fly smoothly and save more time. The time consumed by the DGlobal method is only half of the time cost of LAS, which means the DGlobal is more energy efficient than LAS.

As shown in Figure 15d–f, both LAS and TD3 cannot navigate the UAV to the destination. The reason is that they are all merely based on local information. TD3 and LAS guide the UAV to fly through the left of the maze, which is a straightforward way to close the distance to the destination based on local information. However, it is also a trap of Env6. The DGlobal, on the other hand, can avoid the trap by adopting the global information from the Global A* and can arrive at the destination in the end.

5.3. Navigation in Environments with Unpredictable Obstacles

In this section, we involve unpredictable obstacles in maze-like environments, Env2, Env3, and Env6 (Env5 is similar to Env6 and is omitted), to evaluate the performance of the DGlobal algorithm. We use UEnv2, UEnv3, and UEnv6 to represent environments with unpredictable obstacles. Based on the experimental results in the above sections, the navigation strategies generated by LAS and TD3 are either time-consuming or lead the UAV into traps, so it is unfair to compare them with DGlobal in more complicated environments. Therefore, we conduct a comparison with the state-of-the-art algorithm **FRSVG(0) (FRSVG for short)** [38]. FRSVG is able to make decisions based on historical observations and actions by using recurrent neural networks. It also addresses the challenges that the UAV easily gets trapped when using DRL methods to solve navigation problems in large-scale and

long-trajectory environments. Thus, it serves as an appropriate comparison for DGlobal. The test experiments are carried out 200 times with obstacles being generated randomly. Each time, FRSVG and DGlobal perform with the same distribution of obstacles to maintain fairness. Table 4 shows the success rate (SR), collision rate (CR), and lost rate (LR) of the two algorithms across 200 tests. We define SR, CR, and LR as the respective percentages of episodes in which the UAV successfully reaches the target point, experiences a collision with obstacles, and becomes trapped or lost.



(**a**) Traj. of LAS in Env5 (**b**) Traj. of TD3 in Env5 (**c**) Traj. of DGlobal in Env5

(**d**) Traj. of LAS in Env6 (**e**) Traj. of TD3 in Env6 (**f**) Traj. of DGlobal in Env6

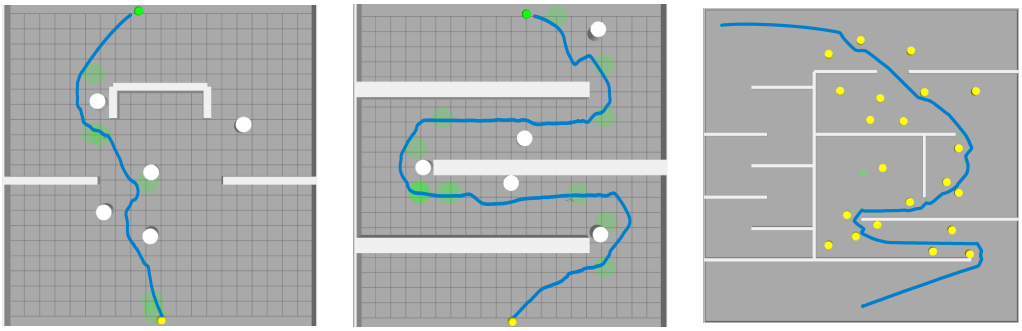
Figure 15. Typical cases of navigation results using LAS, TD3, and DGloal in Env5 and Env6 (Traj. is short for trajectory).

Focusing on the DGlobal method, the data in Table 4 demonstrate that DGlobal achieves a higher success rate compared to FRSVG in all three environments (98.0%, 98.5%, and 95.5% in UEnv2, UEnv3, and UEnv6, respectively). Additionally, as shown in Figure 16, the DGlobal method successfully guides the UAV to avoid these obstacles and arrive at the destination. The capability to deal with unpredictable obstacles is a result of training with DRL. We also notice that the UAV updates its goal by using the goal-updating algorithm when flying through UEnv3 and UEnv6 (the effectiveness of the goal-updating algorithm will be analyzed in the following section).

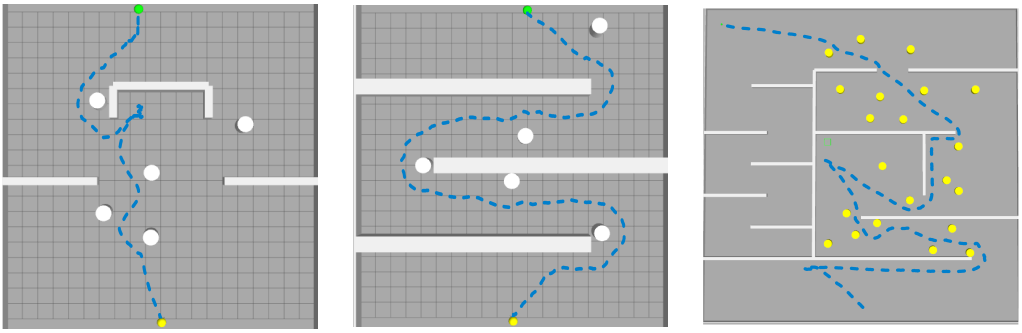
In contrast, the FRSVG agent achieves lower success rates in all three environments (96.5%, 98.0%, and 89.5% in UEnv2, UEnv3, and UEnv6, respectively). Figure 16 provides typical successful instances of the FRSVG algorithm. Although it solves navigation problems in unpredictable scenarios with the help of recurrent networks, it may still generate some unnecessary trajectories during flight, especially in long-trajectory scenarios such as UEnv6. This indicates that while recurrent networks help the agent to capture temporal dependencies in past observations and actions, they may have difficulty in capturing long-term dependencies. Moreover, the lack of a global planner also leads to unnecessary trajectories, as the agent may not be aware of efficient solutions in the larger context.

Table 4. Test results in UEnv2, UEnv3, and UEnv6.

Environment	FRSVG [38]			DGlobal		
	SR (%)	CR (%)	LR (%)	SR (%)	CR (%)	LR (%)
UEnv2	96.5	1.5	2.0	98.0	2.0	0
UEnv3	98.0	0.5	1.5	98.5	1.5	0
UEnv6	89.5	2.5	8.0	95.5	3.0	1.5



(**a**) Traj. of DGlobal in UEnv2 (**b**) Traj. of DGlobal in UEnv3 (**c**) Traj. of DGlobal in UEnv6.



(**d**) Traj. of FRSVG in UEnv2. (**e**) Traj. of FRSVG in UEnv3. (**f**) Traj. of FRSVG in UEnv6.

Figure 16. Typical cases of navigation results in UEnv2, UEnv3, and UEnv6 (Traj. is short for trajectory).

5.4. The Effect of the Goal-Updating Algorithm

We conduct an analysis of the effectiveness of the proposed goal-updating method (Algorithm 2) and its extension (Section 4.6.2) by designing a traditional goal-updating method as a comparison and implementing all of the models in environments with unpredictable obstacles (cylinders).

First, we compare the traditional method with the proposed goal-updating method (the left in Figure 17). In the traditional goal-updating method, the goal is set to the $(i+1)$-th waypoint only when the UAV has reached the i-th waypoint. Thus, the UAV flies through waypoints W2, W3, and W4 accordingly and is finally trapped at W5 since W5 is blocked by an unpredictable obstacle and the DRL policy keeps the UAV away from obstacles at all times. In contrast, the proposed goal-updating algorithms perform much better. The UAV skips W2 and W3, flies through W4 directly, then passes W5 and finally reaches the destination. The reason is that when the UAV is at P1, its distance from W2 is less than $\epsilon + (l_{\text{waypoint}} + l_{\text{UAV}})/2$ and the distance from the nearest obstacle is less than $\epsilon + l_{\text{UAV}}/2$. Therefore, according to the proposed goal-updating method, the goal is changed to W3. Then, the distance between the UAV and W3 is calculated at the next step, which also meets the condition of updating the goal; thus, W4 is the goal now. When the UAV is at P2, there is an obstacle overlapping W5. Therefore, the proposed method relaxes the criterion for the UAV reaching W5 by expanding the range of W5. The proposed goal-updating method can help the UAV complete the task even if new obstacles appear at waypoints, indicating that it improves generalization to environmental changes while retaining the information from waypoints.

Next, we compare the proposed goal-updating method with its extension described in Section 4.6.2. We establish a maze-like environment, and there is a door in the maze. The waypoint (W7) generated by the global planner indicates that the optimal path should go through the door. The proposed goal-updating method makes the UAV trapped in the maze since the door has been blocked by an unpredictable obstacle. However, at P3, the extension version of the goal-updating method changes to another backup waypoint and leads the UAV to the destination in the end.

Our simulations imply that the extension version of the goal-updating method performs the best. However, it also requires additional memory to store backup paths.

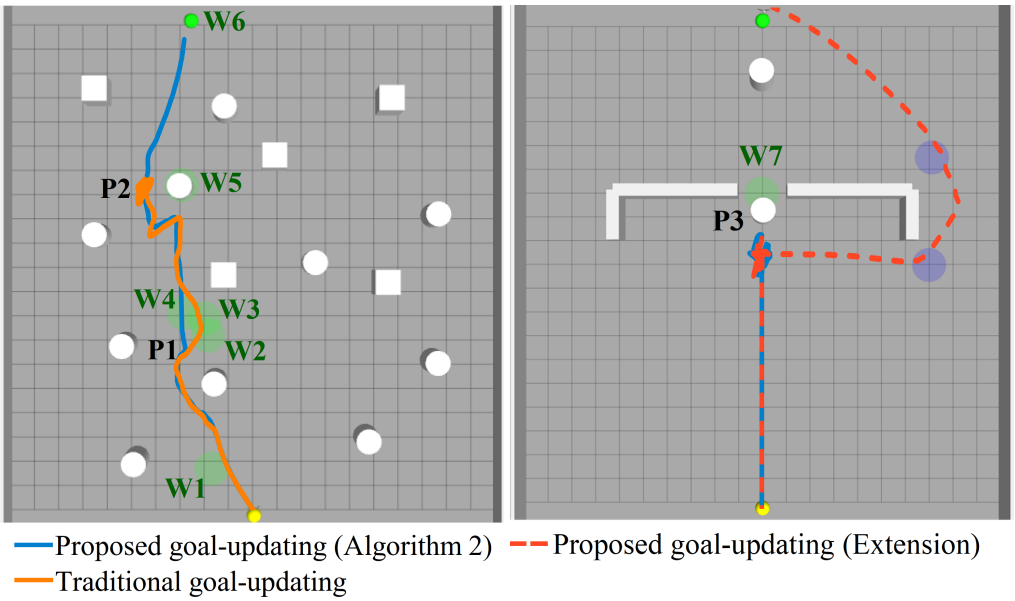


Figure 17. Comparison of trajectories using traditional goal-updating, the proposed goal-updating (Algorithm 2), and its extension (right).

6. Conclusions

This paper proposed a hybrid UAV obstacle-avoidance method by integrating global planning with DRL-based motion planning. It works in three steps. First, we established an HL-DRL training module for real-time UAV navigation and designed a demonstration replay buffer to allow the UAV to learn from both human experience and self-exploration. Second, we developed a global-planning module that produces a few points as waypoint guidance for the UAV. Third, we designed a novel goal-updating algorithm to connect the HL-DRL training module and the global-planning module. According to simulation results, the proposed method incorporates the advantages of both modules. In contrast to global path-planning methods, this method can rapidly adapt to changing environments with short replanning time. Compared to DRL-based motion-planning methods, it can keep the UAV from being trapped in complex situations.

In future research, we plan to expand our current 2D motion-planning approach to 3D motion planning and enhance the method's generalization capabilities to better adapt to more complex and large-scale environments.

Author Contributions: Conceptualization, S.Z.; methodology, S.Z.; software, S.Z.; validation, S.Z.; analysis, S.Z.; writing—original draft preparation, S.Z.; writing—review and editing, T.S.; visualization, S.Z.; supervision, Y.L., F.Y., X.G. and Z.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported by the National Natural Science Foundation of China (under grants No. 62202326 and No. 52271311), the Foundation of the National Defense Key Laboratory (2021-JCJQ-LB-066-12), and the Heilongjiang Touyan Innovation Team Program.

Data Availability Statement: The data used to support this study have not been made available.

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

References

1. Atif, M.; Ahmad, R.; Ahmad, W.; Zhao, L.; Rodrigues, J.J. UAV-assisted wireless localization for search and rescue. *IEEE Syst. J.* **2021**, *15*, 3261–3272. [CrossRef]
2. Dong, J.; Ota, K.; Dong, M. UAV-based real-time survivor detection system in post-disaster search and rescue operations. *IEEE J. Miniaturization Air Space Syst.* **2021**, *2*, 209–219. [CrossRef]
3. Yang, L.; Yao, H.; Wang, J.; Jiang, C.; Benslimane, A.; Liu, Y. Multi-UAV-enabled load-balance mobile-edge computing for IoT networks. *IEEE Internet Things J.* **2020**, *7*, 6898–6908. [CrossRef]
4. Chen, M.; Liang, W.; Li, J. Energy-efficient data collection maximization for UAV-assisted wireless sensor networks. In Proceedings of the 2021 IEEE Wireless Communications and Networking Conference (WCNC), Nanjing, China, 29 March–1 April 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 1–7.
5. Zheng, Q.; Zhao, P.; Li, Y.; Wang, H.; Yang, Y. Spectrum interference-based two-level data augmentation method in deep learning for automatic modulation classification. *Neural Comput. Appl.* **2021**, *33*, 7723–7745. [CrossRef]
6. Mohamed, N.; Al-Jaroodi, J.; Jawhar, I.; Idries, A.; Mohammed, F. Unmanned aerial vehicles applications in future smart cities. *Technol. Forecast. Soc. Chang.* **2020**, *153*, 119293. [CrossRef]
7. Khan, I.H.; Javaid, M. Automated COVID-19 emergency response using modern technologies. *Apollo Med.* **2020**, *17*, 58.
8. Jin, B.; Vai, M.I. An adaptive ultrasonic backscattered signal processing technique for instantaneous characteristic frequency detection. *Bio-Med. Mater. Eng.* **2014**, *24*, 2761–2770. [CrossRef]
9. Shi, T.; Cai, Z.; Li, J.; Gao, H. Joint Deployment Strategy of Battery-Free Sensor Networks with Coverage Guarantee. *ACM Trans. Sens. Netw. (TOSN)* **2021**, *17*, 1–29. [CrossRef]
10. Choset, H.; Lynch, K.M.; Hutchinson, S.; Kantor, G.A.; Burgard, W. *Principles of Robot Motion: Theory, Algorithms, and Implementations*; MIT Press: Cambridge, MA, USA, 2005.
11. Karaman, S.; Frazzoli, E. Sampling-based algorithms for optimal motion planning. *Int. J. Robot. Res.* **2011**, *30*, 846–894. [CrossRef]
12. Xiong, X.; Min, H.; Yu, Y.; Wang, P. Application improvement of A* algorithm in intelligent vehicle trajectory planning. *Math. Biosci. Eng.* **2021**, *18*, 1–21. [CrossRef]
13. Kang, Y.; Yang, Z.; Zeng, R.; Wu, Q. Smooth-RRT*: Asymptotically Optimal Motion Planning for Mobile Robots under Kinodynamic Constraints. In Proceedings of the 2021 IEEE International Conference on Robotics and Automation (ICRA), Xi'an, China, 30 May–5 June 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 8402–8408.
14. Moon, C.B.; Chung, W. Kinodynamic planner dual-tree RRT (DT-RRT) for two-wheeled mobile robots using the rapidly exploring random tree. *IEEE Trans. Ind. Electron.* **2014**, *62*, 1080–1090. [CrossRef]
15. Donald, B.; Xavier, P.; Canny, J.; Reif, J. Kinodynamic motion planning. *J. ACM (JACM)* **1993**, *40*, 1048–1066. [CrossRef]
16. Otte, M.; Frazzoli, E. RRTX: Asymptotically optimal single-query sampling-based motion planning with quick replanning. *Int. J. Robot. Res.* **2016**, *35*, 797–822. [CrossRef]
17. Lan, X.; Di Cairano, S. Continuous curvature path planning for semi-autonomous vehicle maneuvers using RRT. In Proceedings of the 2015 European Control Conference (ECC), Linz, Austria, 15–17 July 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 2360–2365.
18. Goel, U.; Varshney, S.; Jain, A.; Maheshwari, S.; Shukla, A. Three dimensional path planning for UAVs in dynamic environment using glow-worm swarm optimization. *Procedia Comput. Sci.* **2018**, *133*, 230–239. [CrossRef]
19. Zammit, C.; Van Kampen, E.J. Comparison of a* and rrt in real–time 3d path planning of uavs. In Proceedings of the AIAA Scitech 2020 Forum, Orlando, FL, USA, 6–10 January 2020; p. 0861.
20. Ye, H.; Liu, T.; Xu, C.; Gao, F. Integrating Fast Regional Optimization into Sampling-based Kinodynamic Planning for Multirotor Flight. *arXiv* **2021**, arXiv:2103.05519.
21. Tang, L.; Wang, H.; Liu, Z.; Wang, Y. A real-time quadrotor trajectory planning framework based on B-spline and nonuniform kinodynamic search. *J. Field Robot.* **2021**, *38*, 452–475. [CrossRef]
22. Kulathunga, G.; Devitt, D.; Fedorenko, R.; Klimchik, A. Path planning followed by kinodynamic smoothing for multirotor aerial vehicles (mavs). *Russ. J. Nonlinear Dyn.* **2021**, *17*, 491–505. [CrossRef]
23. Ma, H.; Meng, F.; Ye, C.; Wang, J.; Meng, M.Q.H. Bi-Risk-RRT Based Efficient Motion Planning for Mobile Robots. *IEEE Trans. Intell. Veh.* **2022**, *7*, 722–733. [CrossRef]
24. Tang, L.; Wang, H.; Li, P.; Wang, Y. Real-time trajectory generation for quadrotors using b-spline based non-uniform kinodynamic search. In Proceedings of the 2019 IEEE International Conference on Robotics and Biomimetics (ROBIO), Dali, China, 6–8 December 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1133–1138.
25. Yan, A.; Chen, Y.; Xu, Z.; Chen, Z.; Cui, J.; Huang, Z.; Girard, P.; Wen, X. Design of double-upset recoverable and transient-pulse filterable latches for low-power and low-orbit aerospace applications. *IEEE Trans. Aerosp. Electron. Syst.* **2020**, *56*, 3931–3940. [CrossRef]
26. Gu, Q.; Tian, J.; Yang, B.; Liu, M.; Gu, B.; Yin, Z.; Yin, L.; Zheng, W. A Novel Architecture of a Six Degrees of Freedom Parallel Platform. *Electronics* **2023**, *12*, 1774. [CrossRef]
27. Rubí, B.; Morcego, B.; Pérez, R. Quadrotor Path Following and Reactive Obstacle Avoidance with Deep Reinforcement Learning. *J. Intell. Robot. Syst.* **2021**, *103*, 62. [CrossRef]
28. Wang, C.; Wang, J.; Wang, J.; Zhang, X. Deep-reinforcement-learning-based autonomous UAV navigation with sparse rewards. *IEEE Internet Things J.* **2020**, *7*, 6180–6190. [CrossRef]

29. Song, Y.; Steinweg, M.; Kaufmann, E.; Scaramuzza, D. Autonomous drone racing with deep reinforcement learning. In Proceedings of the 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Prague, Czech Republic, 27 September–1 October 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 1205–1212.
30. Sampedro, C.; Bavle, H.; Rodriguez-Ramos, A.; De La Puente, P.; Campoy, P. Laser-based reactive navigation for multirotor aerial robots using deep reinforcement learning. In Proceedings of the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, 1–5 October 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 1024–1031.
31. Niu, H.; Ji, Z.; Arvin, F.; Lennox, B.; Yin, H.; Carrasco, J. Accelerated sim-to-real deep reinforcement learning: Learning collision avoidance from human player. In Proceedings of the 2021 IEEE/SICE International Symposium on System Integration (SII), Iwaki, Fukushima, Japan, 11–14 January 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 144–149.
32. Yan, C.; Xiang, X.; Wang, C. Fixed-Wing UAVs flocking in continuous spaces: A deep reinforcement learning approach. *Robot. Auton. Syst.* **2020**, *131*, 103594. [CrossRef]
33. Chiang, H.T.L.; Faust, A.; Fiser, M.; Francis, A. Learning navigation behaviors end-to-end with autorl. *IEEE Robot. Autom. Lett.* **2019**, *4*, 2007–2014. [CrossRef]
34. Cai, P.; Mei, X.; Tai, L.; Sun, Y.; Liu, M. High-speed autonomous drifting with deep reinforcement learning. *IEEE Robot. Autom. Lett.* **2020**, *5*, 1247–1254. [CrossRef]
35. Liu, C.H.; Ma, X.; Gao, X.; Tang, J. Distributed energy-efficient multi-UAV navigation for long-term communication coverage by deep reinforcement learning. *IEEE Trans. Mob. Comput.* **2019**, *19*, 1274–1285. [CrossRef]
36. Ma, Z.; Wang, C.; Niu, Y.; Wang, X.; Shen, L. A saliency-based reinforcement learning approach for a UAV to avoid flying obstacles. *Robot. Auton. Syst.* **2018**, *100*, 108–118. [CrossRef]
37. Xie, L.; Wang, S.; Markham, A.; Trigoni, N. Towards monocular vision based obstacle avoidance through deep reinforcement learning. *arXiv* **2017**, arXiv:1706.09829.
38. Xue, Y.; Chen, W. A UAV Navigation Approach Based on Deep Reinforcement Learning in Large Cluttered 3D Environments. *IEEE Trans. Veh. Technol.* **2022**, *72*, 3001–3014. [CrossRef]
39. Song, F.; Liu, Y.; Shen, D.; Li, L.; Tan, J. Learning Control for Motion Coordination in Wafer Scanners: Toward Gain Adaptation. *IEEE Trans. Ind. Electron.* **2022**, *69*, 13428–13438. [CrossRef]
40. Gazebo. 2021. Available online: https://gazebosim.org/home (accessed on 23 April 2021).
41. Faust, A.; Oslund, K.; Ramirez, O.; Francis, A.; Tapia, L.; Fiser, M.; Davidson, J. Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning. In Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, Australia, 21–25 May 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 5113–5120.
42. Chiang, H.T.L.; Hsu, J.; Fiser, M.; Tapia, L.; Faust, A. RL-RRT: Kinodynamic motion planning via learning reachability estimators from RL policies. *IEEE Robot. Autom. Lett.* **2019**, *4*, 4298–4305. [CrossRef]
43. Wu, J.; Huang, Z.; Huang, C.; Hu, Z.; Hang, P.; Xing, Y.; Lv, C. Human-in-the-loop deep reinforcement learning with application to autonomous driving. *arXiv* **2021**, arXiv:2104.07246.
44. Jiang, S.; Zhao, C.; Zhu, Y.; Wang, C.; Du, Y. A practical and economical ultra-wideband base station placement approach for indoor autonomous driving systems. *J. Adv. Transp.* **2022**, *2022*, 3815306. [CrossRef]
45. Liu, L.; Zhang, S.; Zhang, L.; Pan, G.; Yu, J. Multi-UUV Maneuvering Counter-Game for Dynamic Target Scenario Based on Fractional-Order Recurrent Neural Network. *IEEE Trans. Cybern.* 2022, *early access*.
46. Ait Saadi, A.; Soukane, A.; Meraihi, Y.; Benmessaoud Gabis, A.; Mirjalili, S.; Ramdane-Cherif, A. UAV path planning using optimization approaches: A survey. *Arch. Comput. Methods Eng.* **2022**, *29*, 4233–4284. [CrossRef]
47. Mir, I.; Gul, F.; Mir, S.; Khan, M.A.; Saeed, N.; Abualigah, L.; Abuhaija, B.; Gandomi, A.H. A survey of trajectory planning techniques for autonomous systems. *Electronics* **2022**, *11*, 2801. [CrossRef]
48. Yao, Z.; Yoon, H.S. Control strategy for hybrid electric vehicle based on online driving pattern classification. *SAE Int. J. Altern. Powertrains* **2019**, *8*, 91–102. [CrossRef]
49. Cimurs, R.; Suh, I.H.; Lee, J.H. Goal-driven autonomous exploration through deep reinforcement learning. *IEEE Robot. Autom. Lett.* **2021**, *7*, 730–737. [CrossRef]
50. Li, B.; Li, Q.; Zeng, Y.; Rong, Y.; Zhang, R. 3D trajectory optimization for energy-efficient UAV communication: A control design perspective. *IEEE Trans. Wirel. Commun.* **2021**, *21*, 4579–4593. [CrossRef]
51. Arulkumaran, K.; Deisenroth, M.P.; Brundage, M.; Bharath, A.A. A brief survey of deep reinforcement learning. *arXiv* **2017**, arXiv:1708.05866.
52. Gupta, J.K.; Egorov, M.; Kochenderfer, M. Cooperative multi-agent control using deep reinforcement learning. In Proceedings of the Autonomous Agents and Multiagent Systems: AAMAS 2017 Workshops, Best Papers, São Paulo, Brazil, 8–12 May 2017; Revised Selected Papers 16; Springer: Berlin/Heidelberg, Germany, 2017; pp. 66–83.
53. Rasool, S.; Ullah, I.; Ali, A.; Ahmad, I. 3D UAV Trajectory Design for Fair and Energy-Efficient Communication: A Deep Reinforcement Learning Technique. *arXiv* **2023**, arXiv:2303.05465.
54. Fujimoto, S.; Hoof, H.; Meger, D. Addressing function approximation error in actor-critic methods. In Proceedings of the International Conference on Machine Learning (PMLR), Stockholm, Sweden, 10–15 July 2018; pp. 1587–1596.
55. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef] [PubMed]

56. Jia, J.; Xing, X.; Chang, D.E. GRU-Attention based TD3 Network for Mobile Robot Navigation. In Proceedings of the 2022 22nd International Conference on Control, Automation and Systems (ICCAS), Jeju, Republic of Korea, 27 November–1 December 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 1642–1647.
57. Jiang, H.; Esfahani, M.A.; Wu, K.; Wan, K.W.; Heng, K.K.; Wang, H.; Jiang, X. iTD3-CLN: Learn to navigate in dynamic scene through Deep Reinforcement Learning. *Neurocomputing* **2022**, *503*, 118–128. [CrossRef]
58. Dodge, M. *Classics in Cartography: Reflections on Influential Articles from Cartographica*; John Wiley & Sons: Hoboken, NJ, USA, 2011.
59. Henderson, P.; Islam, R.; Bachman, P.; Pineau, J.; Precup, D.; Meger, D. Deep reinforcement learning that matters. In Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; Volume 32.
60. Prometheus. 2021. Available online: https://github.com/amov-lab/Prometheus (accessed on 2 May 2021).