*sensors*

Article

# Deep Reinforcement Learning Approach with Multiple Experience Pools for UAV's Autonomous Motion Planning in Complex Unknown Environments

**Zijian Hu, Kaifang Wan *, Xiaoguang Gao, Yiwei Zhai and Qianglong Wang**

School of Electronic and Information, Northwestern Polytechnical University, Xi'an 710129, China; huzijian@mail.nwpu.edu.cn (Z.H.); cxg2012@nwpu.edu.cn (X.G.); zyw@mail.nwpu.edu.cn (Y.Z.); wql1995@mail.nwpu.edu.cn (Q.W.)

* Correspondence: wankaifang@nwpu.edu.cn

Received: 21 February 2020; Accepted: 28 March 2020; Published: 29 March 2020

**Abstract:** Autonomous motion planning (AMP) of unmanned aerial vehicles (UAVs) is aimed at enabling a UAV to safely fly to the target without human intervention. Recently, several emerging deep reinforcement learning (DRL) methods have been employed to address the AMP problem in some simplified environments, and these methods have yielded good results. This paper proposes a multiple experience pools (MEPs) framework leveraging human expert experiences for DRL to speed up the learning process. Based on the deep deterministic policy gradient (DDPG) algorithm, a MEP–DDPG algorithm was designed using model predictive control and simulated annealing to generate expert experiences. On applying this algorithm to a complex unknown simulation environment constructed based on the parameters of the real UAV, the training experiment results showed that the novel DRL algorithm resulted in a performance improvement exceeding 20% as compared with the state-of-the-art DDPG. The results of the experimental testing indicate that UAVs trained using MEP–DDPG can stably complete a variety of tasks in complex, unknown environments.

**Keywords:** UAV; motion planning; deep reinforcement learning; multiple experience pools

## 1. Introduction

The number of applications for unmanned aerial vehicles (UAVs) is widely increasing in the civil arena such as surveillance [1,2], delivery of goods [3,4], power line inspection [5,6], and mapping [7,8]. In the majority of these applications, it is necessary for UAVs to plan their motion such that they can perform their tasks while avoiding threats in complex, unknown environments.

Many traditional path planning algorithms, such as A* algorithm, visibility graph algorithm, and free space algorithm, are used to solve the motion planning problem of UAV, but these methods can usually only achieve good results when the environment or map is known. Under real circumstances, the task environment of UAV is often unknown or not completely known. Simultaneous localization and mapping (SLAM) maps the unknown environment according to position and sensor information of the UAV during its movement in the environment so as to implement the automatic motion planning of the UAV according to the drawn map. Simultaneous localization and mapping make up for the limitations of traditional path planning methods in unknown environments, but these kinds of methods also have a large disadvantage: it is difficult to solve dynamic environment problems. Once the environment changes, the map needs to be redrawn which will seriously affect the mission efficiency of the UAV.

With the rise of artificial intelligence, reinforcement learning (RL) has become an effective method to solve the above problems. Through continuous interaction between the UAV and the environment, the UAV trained by RL can efficiently complete tasks in complex environments. With the increasing complexity of the environment, deep reinforcement learning (DRL) with neural networks can be applied to environments that are closer to reality. Many studies used DRL to solve the autonomous motion planning (AMP) problem of UAV and achieved good results, but these studies still have some shortcomings: (1) the models of the UAV and environment can be more complex and realistic; (2) the convergence speed and convergence results of the algorithm can be improved.

To address these problems, explorations and experiments were conducted in this study. The main contributions of this study are as follows:

1. Construction of a complex 3D simulation environment for AMP.

A universal environment based on the parameters of the real UAV was built. The UAV, obstacles, etc., in this environment were decided in equal proportion according to actual situations. The complete kinematics model of the UAV enables the environment to better simulate real situations. By changing the environment, simulations of many different tasks can be implemented to test the performance of the algorithm.

2. A multiple experience pools DRL (MEP-DRL) framework was designed to solve the AMP problem in complex, unknown environments.

"Experience" usually refers to a set of data containing state, action, reward, and other information in RL. The agent optimizes its action strategy through continuous learning of experiences. The quality of the experiences determines the effectiveness of the training. Therefore, in order to improve the quality of the experiences learned by the agent, multiple experience pools are used to store different kinds of experiences in this paper. In the learning process, dynamically adjusting the proportion of the learning experiences obtained from each experience pool not only solves the AMP problem but also effectively improves the convergence efficiency and results.

3. A method combining model predictive control (MPC) and simulated annealing (SA) to generate expert experiences was introduced.

The AMP problem is a very complicated decision-making problem. In real situations, expert experiences can greatly improve UAV control. An MPC-SA architecture is designed, in which the MPC predicts the future, and then the SA selects the best outcome from the predicted results, thereby achieving a good simulation of expert experiences.

The remainder of this paper is structured as follows: The related work is discussed in Section 2. The AMP problem of a UAV is formulated in Section 3. Section 4 presents the proposed MEP–DDPG algorithm. The simulation environment is introduced in Section 5. The training and testing experiment results are presented and discussed in Section 6. Finally, Section 7 concludes this paper and envisages some future work.

## 2. Related Work

A multitude of methods ranging from offline to online methods have been proposed to address UAVs' AMP problems. Offline-based methods do not learn during the planning process and maintain fixed rules or strategies from beginning to end. Many well-known path planning algorithms fall into this category, the most typical of which is the A* algorithm. Yang [9] described a UAV fast path planning method that uses an improved sparse A* algorithm to significantly improve the search efficiency. Wang [10] proposed using an enhanced sparse A* search method based on Dubins path for UAV path planning that can determine the optimal path with less planning time. Ren [11] proposed a three-dimensional (3D) UAV path planning algorithm based on using an A* algorithm to solve the path planning problem in a complex environment. These methods have achieved good results in specific environments, but they do not perform well when planning in unknown environments.

In comparison, simultaneous localization and mapping (SLAM), which is one of the online-based methods, can successfully solve the AMP problem even when the environment is unknown [12–15]. Wang [12] presented a complete navigation system for an indoor UAV that was based on SLAM, and this system was able to ensure that the UAV could fly along the internal walls of a room without collisions. Cui [13] combined SLAM with an online path planning module to allow UAVs to navigate autonomously through a foliage environment. However, since SLAM needs to model the environment online first, if the environment changes, it could perform poorly. In addition, the efficiency of such methods depends on the complexity of the environment. The more complex the environment, the more time it takes to map.

To overcome the shortcomings of the methods described above, other studies [16–19] have proposed using reinforcement learning (RL), an online learning method, for UAV motion planning. Reinforcement learning can solve the UAV's AMP problem with navigating in an unknown environment, but as the environment becomes more complex and closer to reality, the "dimension curse" problem limits its development.

In 2013, DeepMind innovatively combined deep learning (DL) with RL to form a new hotspot in the field of artificial intelligence which is known as DRL [20]. By leveraging the decision-making capabilities of RL and the perceived capabilities of DL, DRL has been proven to be efficient at controlling UAV [21–31]. Zhu [21] proposed a framework for target driven visual navigation, this framework addressed some of the limitations that prevent DRL algorithms from being applied to realistic settings. Zhang [22] presented a coarse-to-fine approach with RL to address the aspect ratio variation issue for target tracking in UAV. Bøhn Eivind [23] proposed a DRL controller to handle the nonlinear attitude control problem, enabling extended flight envelopes for fixed-wing UAVs. Tai [24] designed a mapless motion planner with DRL which can navigate the nonholonomic mobile robot to the desired targets without colliding with any obstacles. Pfeiffer Mark [25] presented and analyzed an approach that combines the advantages of both imitation learning and DRL for target-driven map-less navigation. Han [26] introduced a double deep Q-Network (Double DQN) [32] that utilized a priority sample replay method, and this demonstrated better results than DQN [20] and Double DQN when UAVs navigated through a 3D obstacle avoidance environment. Kersandt [27] used DQN, Double DQN, and Dueling DQN [33] in the same UAV control mission and compared each of these methods. Singla [28] designed a deep recurrent Q-Network [34] with temporal attention that exhibited significant improvements over DQN and D3QN [32] for UAV motion planning in a cluttered and unseen environment. For the autonomous landing task of UAV, Polvara R [29] introduced a sequential DQN which is comparable with DQN and human pilots while being quantitatively better in noisy conditions. Wang [30] proposed a fast recurrent deterministic policy gradient algorithm to address the UAV's autonomous navigation problem in a large-scale complex environment. Rodriguez-Ramos Alejandro [31] addressed the continuous UAV landing maneuver on a moving platform by means of the DDPG algorithm, trained in simulation, and tested in real flights.

These studies demonstrate that DRL has already successfully addressed several AMP problems, but as the unknown environment becomes increasing complexity, there are still some challenges that must be overcome:

1. The models of the UAV and environment are simplified.

Previous studies have focused on UAV navigation. Therefore, the UAV model has been simplified. For example, in real situations, the motion space of a UAV is continuous, whereas in some studies [26–28], the motion space of a UAV is considered discrete. Furthermore, the flight speed of a UAV is variable within a certain range, where Reference [30] regards UAV speed as a fixed value. In addition to these issues, the environment should also be as realistic as possible. Otherwise, these simplifications of the UAV model will make the learning process simpler; however, they may lead to poor practicability.

2. The convergence speed and results can be improved.

It is well known that DRL, which is the approach based on artificial neural networks, involves many hyperparameters, and its performance is often inextricably linked to the setting of these hyperparameters. In addition, other factors such as different algorithm frameworks and reward functions can also have a significant impact on the results. Therefore, it is essential to use appropriate strategies to optimize these factors and improve the convergence speed as well as the results.

The above problems are discussed and solved in detail in the following sections of this study.

## 3. Problem Formulation

### 3.1. UAV's AMP

As mentioned earlier, in actual applications of UAVs, AMP is an essential technology that determines if the UAV can successfully complete the task. If the UAV cannot appropriately control its own posture and movements, it may result in a failure of the mission and may also cause damage to surrounding objects. For example, when a UAV is delivering goods, it may cause injury to people or vehicles if it is flying too low or too fast. Figure 1 is a schematic diagram of an AMP problem from different perspectives.

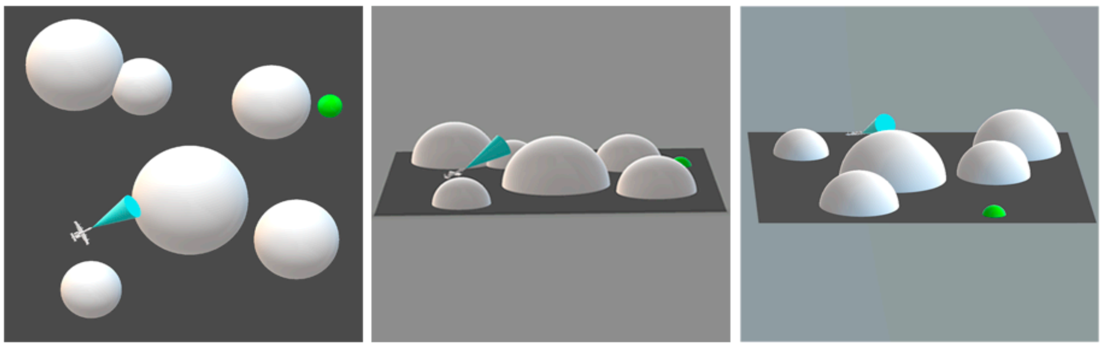

**Figure 1.** Different perspectives of an AMP problem. **Left**: top view; **Middle and Right**: side views from different perspectives.

The blue cone in the figure represents the range that the UAV can sense through the sensor. The white hemispheres represent dangerous areas such as buildings in ground search tasks or an adversary's radar monitoring range. The green hemisphere represents the final destination that the UAV needs to reach. In short, the AMP problem is how to make the UAV automatically use the limited information obtained inside the range of the blue cone to adjust its posture and reach the position of the green hemisphere efficiently without entering the white hemispheres areas.

#### 3.1.1. Motion Planning Framework for UAVs

In the real world, UAV motion planning can be divided into three modules: navigation, guidance, and control.

As shown in Figure 2, the three modules of navigation, guidance, and control complete the motion planning for UAV. The navigation module is primarily responsible for sensing which includes perceiving environment and estimating the state of the UAV and passing the perceived information to other modules. The perceived information usually includes the UAV's position, speed, flying height, wind speed, target position, etc. The guidance module plays a planning role throughout the closed loop. It analyzes the information obtained about the UAV and environment, plans the motion of the UAV in combination with the scheduled flight mission, and outputs the required load factor to the other modules. When the control module receives the required load factor, the required load factor will be converted into different actuator commands and then passed to the corresponding actuators. These actuators, such as engines and rudders, can execute the corresponding commands and achieve stable control of the UAV.

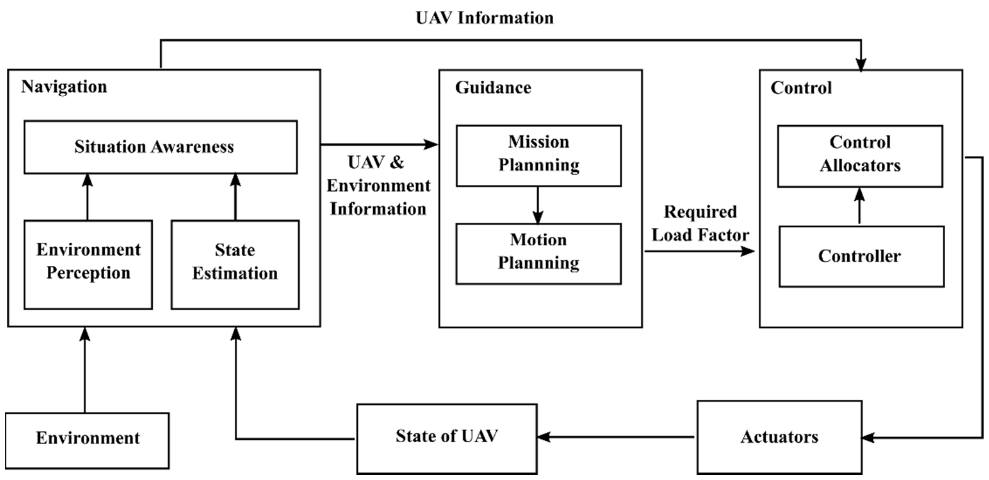

**Figure 2.** Unmanned aerial vehicle (UAV) motion planning framework for real-world applications.

In the simulation environment, the navigation, guidance, and control modules do not operate independently. As shown in Figure 3, the motion planning module directly replaces the navigation and guidance modules. In addition, the control module and the UAV model are integrated into the simulation module. The main work of the motion planning module is integrating the current information of environment and the UAV and using an offline or the online-based method to plan the motion of the UAV according to the UAV's mission. In addition, based on the action to be performed by the UAV, the motion planning module will calculate the required load factor of this action and transfer it to the simulation module. The simulation module calculates the UAV's acceleration, velocity, and position based on this load factor and then adjusts the attitude of the UAV model and displays it on the simulation interactive interface.

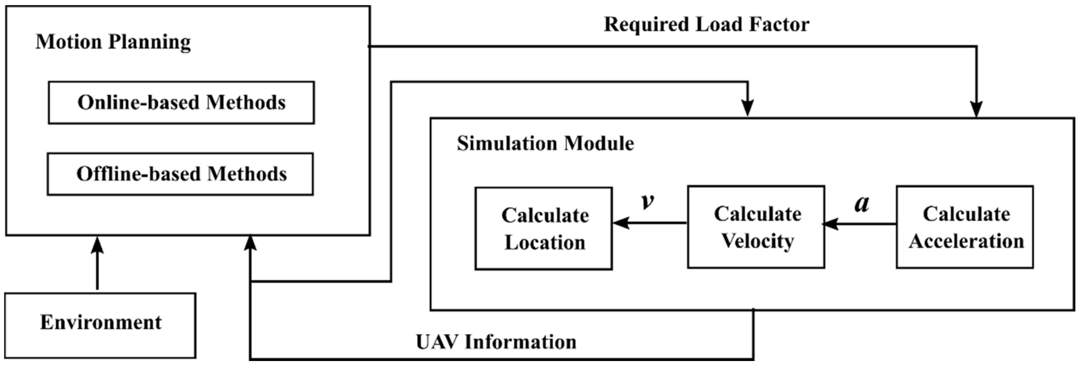

**Figure 3.** UAV motion planning framework in the simulation environment.

### 3.1.2. UAV Model

Previous studies focused more on the motion planning module and simplified the simulation module, while the UAV model used in this study was as realistic as possible. Here, this paper assumed that the velocity direction of the UAV's velocity was consistent with the direction of the UAV's axis. When the motion planning module outputs the required load factor $n_u$ at the current moment, according to the reference [35], the acceleration $a_u$ at the current moment can be calculated as follows:

$$a_u = \begin{bmatrix} a_{ux} \\ a_{uy} \\ a_{uz} \end{bmatrix} = n_u \cdot g = \begin{bmatrix} n_{ux} \\ n_{uy} \\ n_{uz} \end{bmatrix} g \tag{1}$$

Defining $\Delta t$ as the simulation step, the velocity $v_u$ at the current moment can be obtained from the velocity $v_0$ at the previous moment:

$$v_u = v_0 + a_u \cdot \Delta t = \begin{bmatrix} v_{0x} + a_{ux} \cdot \Delta t \\ v_{0y} + a_{uy} \cdot \Delta t \\ v_{0z} + a_{uz} \cdot \Delta t \end{bmatrix} \tag{2}$$

Similarly, the current UAV position $p_u$ can be calculated from the position $p_0$ at the previous moment:

$$p_u = \begin{bmatrix} p_{ux} \\ p_{uy} \\ p_{uz} \end{bmatrix} = p_0 + v_u \cdot \Delta t = \begin{bmatrix} p_{0x} + v_{ux} \cdot \Delta t \\ p_{0y} + v_{uy} \cdot \Delta t \\ p_{0z} + v_{uz} \cdot \Delta t \end{bmatrix} \tag{3}$$

At the same time, the yaw angle $\varphi_u$ and pitch angle $\alpha_u$ of the UAV at the current moment can be calculated according to the direction of the velocity (Figure 4):

$$\varphi_u = atan\left(\frac{v_{uy}}{v_{ux}}\right) \tag{4}$$

$$\alpha_u = atan\left(\frac{v_{uz}}{\sqrt{{v_{ux}}^2 + {v_{uy}}^2}}\right) \tag{5}$$

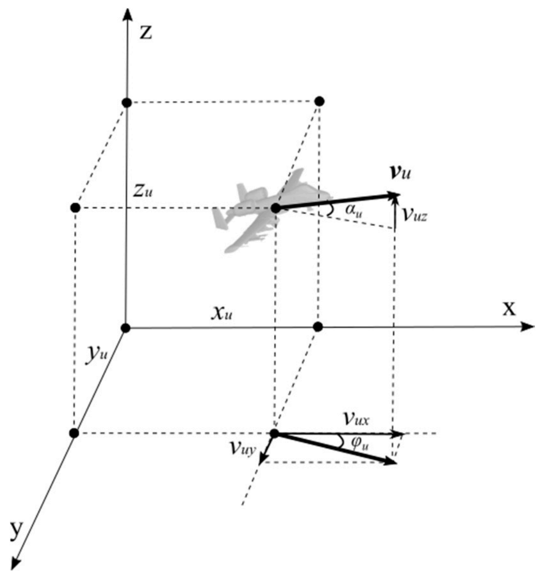

**Figure 4.** UAV attitude angles calculated in a 3D environment.

So far, all the required UAV information has been obtained. In this paper, the vector $\xi_u = [p_{ux}, p_{uy}, p_{uz}, \varphi_u, \alpha_u, \| \mathbf{v}_u \|]^T$ represents the UAV's state for calculation and description.

## 3.2. *RL for UAV's AMP*

The UAV's AMP problem is a typical agent decision-making problem [26], so RL-based approaches can be used to address it. In 1957, Bellman [36] proposed a model for sequential decisions known as the Markov decision process (MDP). Almost all RL problems can be modeled as an MDP and shown as a 5 tuple $(S, A, P, R, \gamma)$, where $S$ is the state space, which is the set of all states that an agent can reach in a specific environment; $A$ represents the action space in the environment; $P$ is the probability of executing an action $a$ from state $s$ to state $s'$; $R$ represents the reward received by the agent after executing action $a$ and transiting from state $s$ to state $s'$ (where $a \in A$ and $s, s' \in S$); $\gamma \in [0,1]$ is the discount factor which determines the importance of current or future rewards [37].

The interaction process between the UAV and the environment can be constructed using MDP: At each discrete time step $t$, the state of the UAV is $s_t$, and the action $a_t$ is performed based on this state. After this time step, the UAV receives the reward $r_t$ from the environment and reaches the next state $s_{t+1}$. The action selected by the UAV at each time step is determined by $\pi$, which is the collection of policies. For example, an element $\pi(a|s)$ from $\pi$ represents the probability that the UAV takes an action $a$ at a certain state $s$. The goal of RL is to find the optimal policy $\pi^*$, and to obtain the largest cumulative reward $R_t$ if policy $\pi^*$ is followed:

$$R_t = \sum_{t\prime=t}^{T} \gamma^{t\prime-t} r(s_{t\prime}, a_{t\prime}) \tag{6}$$

where $T$ represents the end time step of an episode.

The state-action value function $Q^{\pi}(s,a)$ represents the expected episode cumulative reward obtained by the UAV when performing the action $a$ in accordance with the policy $\pi$ in the current state $s$:

$$Q^{\pi}(s,a) = E[R_t|s_t = s, a_t = a, \pi] \tag{7}$$

When the optimal policy $\pi^*$ is followed:

$$Q^*(s,a) = \underset{\pi}{max} E[R_t|s_t = s, a_t = a, \pi] \tag{8}$$

From the Bellman equation [36]:

$$Q^*(s,a) = E\left[ r_{t+1} + \gamma \underset{a_{t+1}}{max} Q^{\pi}(s_{t+1}, a_{t+1}) \middle| s_t = s, a_t = a \right] \tag{9}$$

Constantly iterating Formula (9) will eventually converge the state-action value function to obtain the optimal policy [38]:

$$\pi^* = \underset{a \in A}{argmax} Q^*(s,a) \tag{10}$$

The state-action value function can only be used to address the discrete action space problem, because in continuous spaces finding the greedy policy requires an optimization of $a_t$ at every timestep; this optimization is too slow to be practical with large, unconstrained function approximators and nontrivial [39]. While in the real world, the action space of a UAV is continuous, which requires the addition of some policy-based RL methods. These are explained in Section 4.

## 4. MEP–DDPG for Motion Planning

### *4.1. DDPG*

RL methods are mainly classified into two types of value-based methods and policy-based methods. Value-based methods output the state-action values and select the action with the highest value. Common methods in this type include Q-learning [40] and DQN. Policy-based methods directly output the probability of the next action, and select the action based on the probability. This type of method is suitable for non-continuous and continuous motion problems. A typical method of the policy-based methods is policy gradient [41].

Lillicrap [39] used the idea of extending DQN based on Q-learning to transform the deterministic policy gradient (DPG) [42] and proposed a DDPG algorithm based on the actor–critic framework. In the actor–critic framework, the actor network chooses action based on probability. The critic network evaluates the action selected by the actor network. The actor network then modifies the probability of action selection based on the evaluation of the critic network. Because of this framework, DDPG could handle problems with both high-dimensional continuous state space and high-dimensional continuous action space [43]. In DDPG, the double-network structure cuts off the correlation and improves the learning efficiency of the neural networks. Consequently, there are target-net and eval-net in both the actor network and the critic network.

The critic network updates the network parameters by minimizing the loss function $L\left(\theta^{Q}\right)$:

$$\begin{cases} L\left(\theta^{Q}\right) = \frac{1}{N}\sum_{i}\left(y_i - Q\left(s_i, a_i\middle|\theta^{Q}\right)\right)^2 \\ y_i = r(s_i, a_i) + \gamma Q'\left(s_{i+1}, \mu'\left(s_{i+1}\middle|\theta^{\mu'}\right)\middle|\theta^{Q'}\right) \end{cases} \tag{11}$$

where $\theta^{Q}$ and $\theta^{Q'}$ represent the parameters of eval-net and target-net in the critic network respectively, $\theta^{\mu'}$ means the parameters of target-net in actor network, $N$ is the number of experiences used to learn.

The actor network uses the Monte Carlo sampling method to approximate the expected value and approximates its network parameters using the chain rule:

$$\nabla_{\theta^{\mu}} J \approx \frac{1}{N}\sum_{i} \nabla_a Q\left(s, a|\theta^{Q}\right)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{s_i} \tag{12}$$

where $\theta^{\mu}$ represents the parameters of the eval-net actor network.

During the training process, only the eval-net parameters are required, and the parameters of the target-net are soft-updated by eval-net at regular intervals:

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta' \tag{13}$$

where the range of $\tau$ is $[0, 1]$.

This soft-updated method makes the problem more similar to supervised learning, which improves its stability.

The DDPG also uses the experiences playback mechanism [20] to store the experience $(s_t, a_t, r_t, s_{t+1})$ generated by the interaction between the actor network and the environment in the experience pool. During network training, samples are randomly selected from the experience pool each time there is a network parameter update. However, the experiences in the experience pool are not all useful for learning. In particular, during the early stages of training, a majority of the samples in the experience pool are not useful. Improving the quality of each sampling experience will have a significant impact on the efficiency of the DDPG algorithm.

## *4.2. MEP–DDPG Framework*

As mentioned in Section 3, UAV motion planning represents a complicated problem. In practice, UAVs are often controlled by operators on the ground using remote control equipment. When using RL to realize UAV's AMP, if we can add human expert experiences to the learning process, the UAV may learn better strategies by combining expert experiences with the experiences it has explored. An MEP–DDPG framework is designed to incorporate expert experiences into the DRL process.

As shown in Figure 5, multiple experience pools included in MEP–DDPG are used to store different types of experiences. At each time step $t$, the UAV performs the action $a_t$, and through interaction with the environment, an experience $(s_t, a_t, r_t, s_{t+1})$ is generated. According to the method of generating this action (whether generated under the guidance of experts or not under the guidance of experts), this experience is put into different experience pools. It is assumed that expert experiences are generated under the guidance of different human experts. However, different experts may have different UAV motion planning policies. For example, Expert 1 requires the UAV to always maintain a slow speed and fly cautiously, whereas Expert 2 requires UAV to fly at the maximum speed when the threats are not detected, etc. Under the framework of MEP–DDPG, the experiences generated by a UAV interacting with the environment under the guidance of Expert 1 is stored in experience pool 1, the experiences generated under the guidance of Expert 2 is stored in experience pool 2, and so on. The expert experiences generated under the guidance of $X-1$ different experts are stored in $X-1$ different experience pools, and the experiences gained by the UAV is stored in experience pool $X$. During the learning time, a total of $N$ experiences are sampled from $X$ different experience pools to form a batch according to a certain sampling strategy. The parameters of the critic network are updated by minimizing the loss function $L\left(\theta^{Q}\right)$ (Formula (11)). The strategy gradient method is used to update the parameters of the actor network (Formula (12)). Then, the parameters of the target-net in both actor network and critic network are updated by the soft update method. The above process is performed at each time step, and the cycle is repeated until the networks reach convergence.

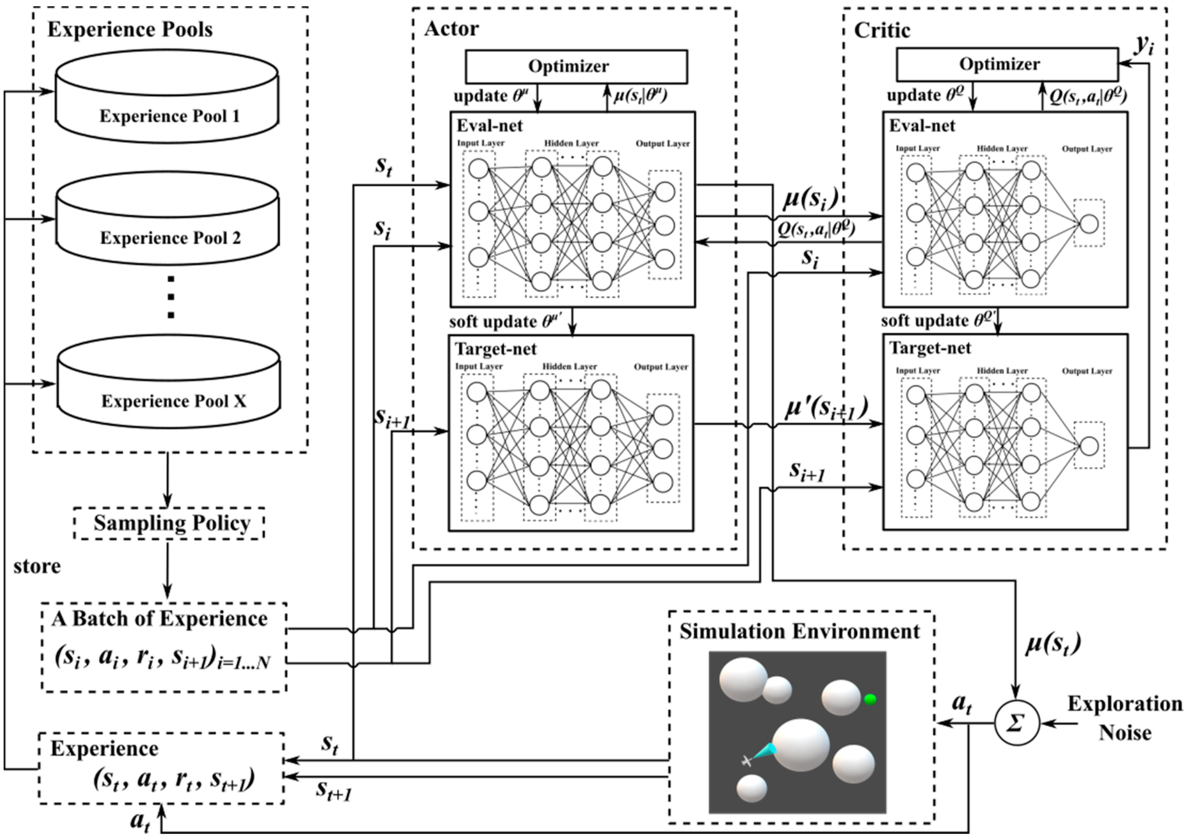

**Figure 5.** Framework of MEP–DDPG (experience pools are used to store different kinds of experiences; the actor network is responsible for selecting actions; the critic network is responsible for evaluating selected actions).

In the early stages of training, because the UAV knows nothing about the environment, most of the experiences gained by randomly taking actions to interact with the environment is not useful. Therefore, this MEP–DDPG framework uses expert guidance to generate expert experiences and stores them into the corresponding experience pool based on a certain proportion $\eta$. As the training process progresses, $\eta$ gradually decreases to 0 while the learning method gradually returns to the initial DDPG in the second half of the training.

Because the UAV learns higher-quality experiences (expert experiences) in the early stages of training, this means that the parameters of the neural network are quickly adjusted to reach a good state. This method not only avoids falling into a local optimal solution in the early stages of training, but also greatly improves the convergence speed of the neural networks.

### *4.3. MPC-SA for Expert Experiences*

During each training process, the UAV interacts with the environment millions of times. If a real expert guides the motion of the UAV to provide expert experiences, it will require a significant amount of manpower and time. Therefore, an MPC-SA framework is proposed, which can simulate human experts by using limited information and provide expert guidance for the UAV's training.

#### 4.3.1. Model Predictive Control

According to references [44–46], the main idea of MPC is rolling optimization. Using the UAV's AMP as an example and assuming that UAV's actions are discrete, only three actions can be selected at each moment: deflecting to the left by a certain angle, deflecting to the right by a certain angle, or maintaining the current direction.

As shown in Figure 6, assuming the current time is $t = 0$, all possible action sequences from $t = 0$ to $t = n - 1$ can be enumerated (only the possible action sequences for $n = 3$ are shown). In other words, at each time step, all possible action sequences for the next 3 time steps are enumerated. From these, it is easy to select the optimal action sequence which is shown in red in Figure 6. By executing the red action sequence, the UAV was able to approach the target as quickly as possible without colliding with the obstacle. However, only let the UAV perform the first action in this sequence, which was to deflect to the right by a certain angle at time $t = 0$. Consequently, it reached the position shown at time $t = 1$ in the figure. Then, predict the action sequences of the next 3 time steps and let the UAV execute only the first action again. This process was repeated until the UAV finds the target and completes its mission.

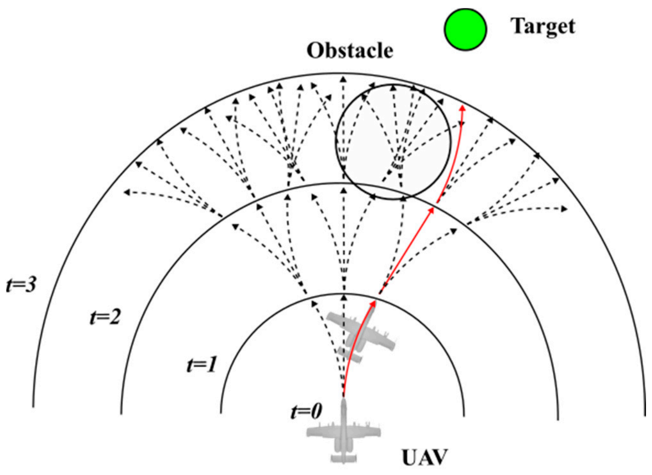

**Figure 6.** MPC for a UAV's motion planning (black arrows represent optional action sequences; red arrow represents the optimal action sequence).

In this way, the MPC rolling optimization method was used to plan the UAV's motion. This not only helps the UAV select the optimal action at each time step but also avoids failure to complete the task due to the changes in the environment. However, in practice, the action space of the UAV at each time step is continuous, which means that there are countless action sequences that can be selected. It is impossible to list all the action sequences one by one, so it is necessary to use some optimization methods to choose a better action sequence.

#### 4.3.2. Simulated Annealing

Simulated annealing is an optimization algorithm proposed by Kirkpatrick [47] in 1983 that includes two steps. First, a solution is randomly selected to start the process, and then a random perturbation is generated in order to obtain a new solution. Second, the objective function is used to judge whether the new solution is good or bad, and each new solution is selected to be accepted or rejected according to the following Formula (14):

$$p = \begin{cases} 1 & if\ E(x_{new}) < E(x_{old}) \\ exp\left(-\frac{E(x_{new})-E(x_{old})}{T}\right) & if\ E(x_{new}) \geq E(x_{old}) \end{cases} \tag{14}$$

where $p$ is the probability of choosing the new solution to take the place of the old one, while $E(x)$ is the objective function value of the solution $x$. By simulating the process of object cooling, the process described above is continuously repeated until the cycle ends and the final solution is obtained.

Because random perturbations are added into the solution process, the SA can move out of the local optimum to determine the global optimal solution. The SA algorithm is used to select among the action sequences predicted at each time step. Figure 7 shows some of the selection processes of the UAV in different states:

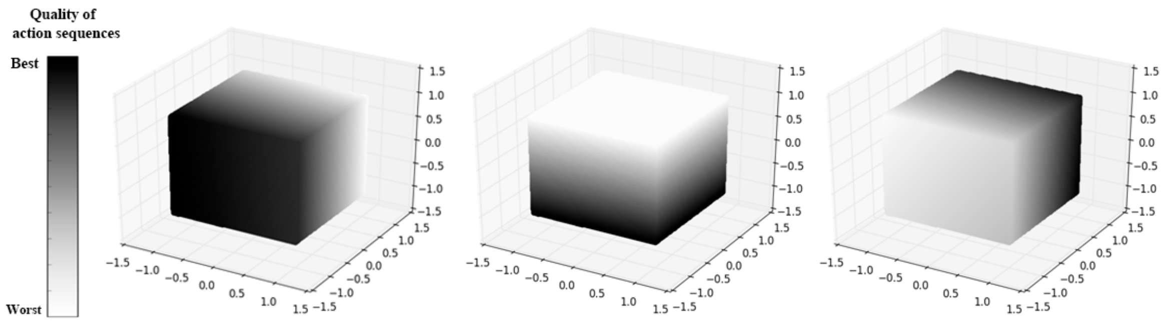

**Figure 7.** The UAV's decision-making processes at different states. **Left**: the position of the UAV is (300, 400, 80) and the position of the target is (0, 0, 0); **Middle**: the position of the UAV is (150, 350, 70) and the position of the target is (120, 360, 0); **Right**: the position of the UAV is (0, 0, 20) and the position of the target is (10, 250, 0);.

Since the dimension of the action space was too high when $n \geq 2$ in the MPC, here this paper set $n = 1$ to visualize the decision process which means that the UAV chose the optimal one among all possible action sequences at the current time step and executed it. At each time step, the required load factor of the UAV was a three-dimensional vector whose value range in each direction was $[-1, 1]$. As shown in Figure 7, the action space of the UAV at each moment was a cube, in which there were countless actions that could be selected. Each action sequence was evaluated according to the objective function in the SA algorithm and assigned a color; the darker the color, the better the action sequence. The role of SA was to choose the action sequence with the highest quality (the darkest color) in this continuous action space as much as possible.

## 4.4. MEP–DDPG Algorithm

The experiences generated by the MPC-SA can provide a good approximation for the experiences of human experts. These experiences were placed in different experience pools, and a certain sampling strategy was used for sampling. After applying the DDPG algorithm to train the UAV, the MEP–DDPG algorithm could finally formed (Algorithm 1):

**Algorithm 1: MEP–DDPG Algorithm**

```
Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$
Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize experience pools $R_1, R_2 \cdots R_X$
for episode = $1, M$ do
  Initialize a random process $\mathcal{N}$ for action exploration
Receive initial observation state $s_1$
for $t = 1, T$ do
  According to probability $\eta$, select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ based on current policy and exploration noise or select action $a_t$ by MPC-SA
  Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
  Store experience $(s_t, a_t, r_t, s_{t+1})$ in specific $R$ depending on the source of $a_t$.
  Follow a specific sampling policy, sample a random minibatch of $N$ experiences $(s_i, a_i, r_i, s_{i+1})$ from $R_1, R_2 \cdots R_X$
  Set
      $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
  Update critic by minimizing the loss:
      $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
  Update the actor policy using the sampled policy gradient:
      $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i,a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$
  Update the target networks:
      $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
      $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
  end for
end for
```

At each time step $t$, the UAV chose an action $a_t$ based on the existing strategy or according to the MPC-SA based on a certain probability $\eta$. Subsequently, the UAV executed this action $a_t$ at the current state of the environment $s_t$ and obtained the environmental feedback reward $r_t$ and the next state $s_{t+1}$. Next, according to the source of the action $a_t$, it stored the experience $(s_t, a_t, r_t, s_{t+1})$ in a corresponding experience pool $R$ for the learning process. During the learning time, a total of $N$ experiences were sampled from all experience pools to form a minibatch following a certain policy. Finally, the parameters of each network were updated according to the DDPG's soft update method.

## 5. Training and Testing Environment

To achieve conditions similar to real-world conditions, this study designed a 3D simulation environment based on the parameters of the real UAV. At present, countries around the world have made progress in the research of UAVs and created advanced UAVs. These advanced UAVs can be used in many different tasks such as: disaster monitoring, anti-smuggling, environmental protection, meteorological observation, forest fire prevention and geological survey, etc. The ranges of some main parameter of these advanced UAVs are shown in the Table 1:

**Table 1.** The ranges of some main parameter of some advanced UAVs.

| Parameter | Range |
|---|---|
| maximum flying height | 7000–12,000 m |
| maximum flight speed | 220–370 km/h |
| maximum flight time | 20–40 h |
| maximum load factor | 20–30 |
| maximum take-off weight | 3–7 t |

Based on these parameters, a $120 \times 90 \times 10$ km$^3$ simulation environment in equal proportions was constructed. The programming language used to build the platform was Python 2, and the display interface was implemented by using the visualization toolkit. Figure 8 shows the simulation environment from different perspectives:

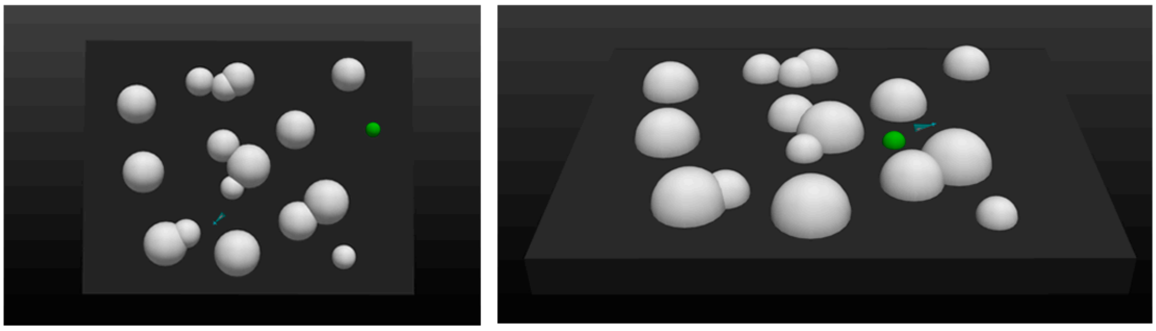

**Figure 8.** The simulation environment as shown from different perspectives. **Left**: top view; **Right**: main view.

In this environment, the white hemispheres represent the danger areas and the green hemisphere represents the target. The mission of UAV was to fly as low as possible and reach the target area as quickly as possible without entering the danger areas. The UAV's maximum flight speed was 103 m/s and the maximum load factor was 25. In addition, the acceleration of gravity in the environment was set at a fixed value of 9.8 m/s$^2$. The radiuses of the threat areas ranged from 5–10 km while the radius of the target area was 3 km.

For detection, $N_r$ line segments were used to simulate lidar. As shown in Figure 9, $N_r$ blue line segments were used to sense the information in environment. Each line segment can calculate the distance $d$ from the UAV to the obstacles or boundaries. In addition, when an object is detected by a line segment, the blue color will change to red as an alert. The monitoring distance of lidar is set to 50 km in the experiments.

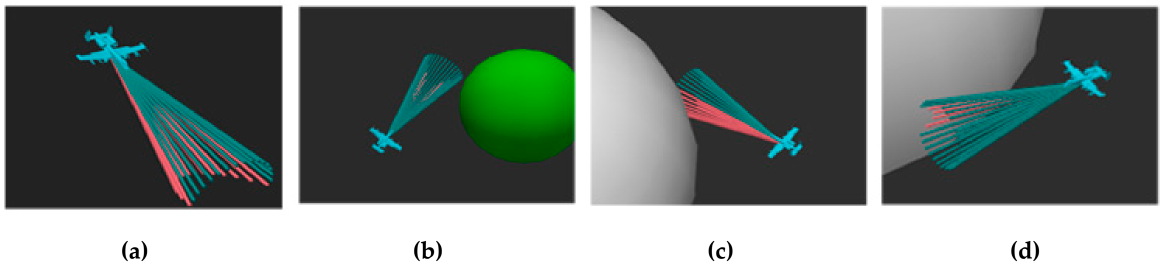

**Figure 9.** Blue line segments show the sensor range of the UAV. (**a**,**b**): ground (boundary) detected; (**c**,**d**): obstacle detected.

At each time step $t$, an environment state vector $\xi_e = [d_1, d_2, \cdots, d_{Nr}]^T$ was calculated and used to form the state $s_t$ together with $\xi_u$ and $\xi_T$, where $\xi_T = \left[p_{Tx}, p_{Ty}, p_{Tz}\right]^T$, and $p_{Tx}$, $p_{Ty}$, $p_{Tz}$ represent the position of the target. Therefore, the system state $s = \left[p_{Tx} - p_{ux}, p_{Ty} - p_{uy}, p_{Tz} - p_{uz}, \varphi_u, \alpha_u, \| v_u \|, d_1, d_2, \cdots, d_{Nr}\right]$ with a total of $N_r + 6$ elements was fed into the critic network, and an action $a_t = \left[n_{ux}, n_{uy}, n_{uz}\right]$ was the output.

To avoid the sparse reward problem in RL, the reward function was set as follows:

$$r(s,a) = \begin{cases} r_a & if\ arrive\ the\ target\ area \\ r_c & if\ collied\ or\ out\ of\ range \\ \lambda_1\left(D_{pre} - D_{cur}\right) + \lambda_2\left(-\frac{\Delta\varphi+\Delta\mu}{8}\right) + \lambda_3\left(1 - \frac{p_{uz}}{H_{env}}\right) + \lambda_4 d_{ave} + \lambda_5 \| v_u \| & every\ step \end{cases} \tag{15}$$

where $D_{pre}$ and $D_{cur}$ are the previous and current relative distances between UAV and the target, respectively; $\Delta\varphi$ and $\Delta\mu$ represent the yaw angle and the pitch angle between UAV's direction and target, respectively; $H_{env}$ is the maximum height of the simulation environment; $d_{ave}$ represents the average of $\xi_e$. These 5 sub items in Formula (15) represent the contributions of distance, angle, height, threat, and velocity to the rewards, respectively, and the contribution rates can be tuned by $\lambda_1, \lambda_2 \cdots \lambda_5$.

## 6. Experiments and Tests

### 6.1. Training in Static Environments

During the entire training process, the hyper parameters were set as follows:

1. DDPG algorithm:

$N_r$ was 32, which indicates that there were 32 line segments used to simulate lidar, and the input layer of the actor network had 38 nodes. The actor network had two hidden layers with 100 nodes for each and one output layer with three nodes, while the critic network had two hidden layers with 100 nodes for each and one output layer with 1 node. The learning rates of the actor and the critic networks were 0.0001 and 0.001, respectively. The soft update rate $\tau$ of the actor network was 0.1 and the $\tau$ of the critic network was 0.2. The discount factor $\gamma$ was 0.9. The optimizer was Adam. The size of experience pool was 50,000 and the batch size was 256. The parameters in the reward function (Formula (15)) were: $r_a = 100$, $r_c = -200$, $\lambda_1 = 20$, $\lambda_2 = 20$, $\lambda_3 = 10$, $\lambda_4 = 40$, and $\lambda_5 = 10$.

2. MEP–DDPG algorithm:

For the SA part, the objective function was set according to Formula (16):

$$f_{SA} = \rho_1\left(D_{pre} - D_{cur}\right) + \rho_2 D_{obs} \tag{16}$$

where $D_{obs}$ represents the distance between the UAV and the nearest obstacle that can be detected, and the values of $\rho_1$ and $\rho_2$ are 100 and 200, respectively. The number of extra experience pools for storing expert experiences was 1, and the capacity was 50,000. The sampling policy in multiple experience pools was as follows: $k$ was the proportion of the experiences sampled from the expert experience pools, and its initial value was 100%. During the training process, the value of $k$ decreased by 0.002% at each learning time until it was reduced to 10%. This ensured that at the beginning of the training, a majority of the learning experiences came from the expert experience pools, and only 10% of the experiences came from the expert experience pools in the second half of the training process. All other hyperparameters were the same as those used in the DDPG algorithm.

During the training process, the position of the UAV and the target were randomly generated. All the algorithms were implemented with Tensorflow and models were trained on Ubuntu 16.04 with a GeForce RTX 2070 GPU for 5000 episodes. The maximum number of steps per episode was set to 3000. It took 10–14 h to complete a training with 5000 episodes. Experiments were performed in simulation environments with different levels of environmental complexity, and the average success rate (the probability of the UAV successfully reaching the target area in last 500 episodes) and episode average reward (the average rewards of all the steps in one episode) were used to evaluate the performance of the algorithms. The experimental results are shown in Figures 10–12:

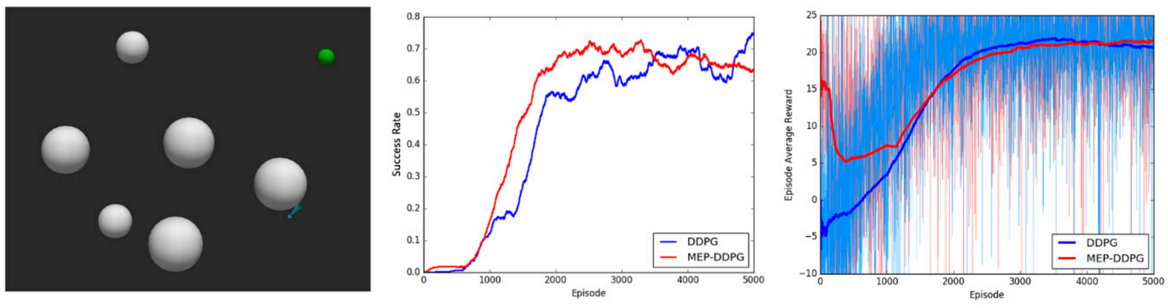

**Figure 10.** Experimental results in a low complexity environment. **Left**: the simulation environment with low complexity (6 threats); **Middle**: the success rate of DDPG and MEP–DDPG; **Right**: the episode average reward of DDPG and MEP–DDPG.

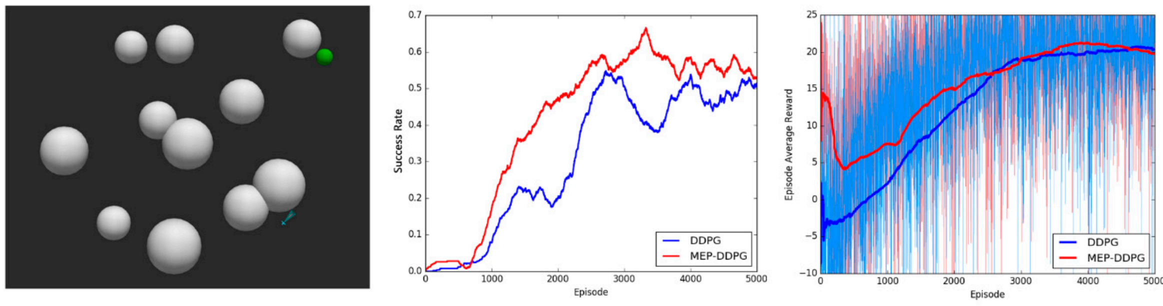

**Figure 11.** Experimental results in a medium complexity environment. **Left**: the simulation environment with medium complexity (11 threats); **Middle**: the success rate of DDPG and MEP–DDPG; **Right**: the episode average reward of DDPG and MEP–DDPG.

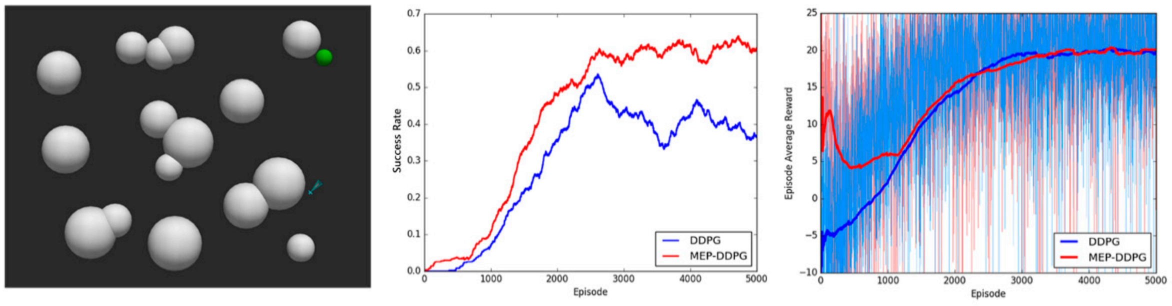

**Figure 12.** Experimental results in a high complexity environment. **Left**: the simulation environment with high complexity (16 threats); **Middle**: the success rate of DDPG and MEP–DDPG; **Right**: the episode average reward of DDPG and MEP–DDPG.

The experimental results demonstrated that the convergence speed and convergence results were better for the MEP–DDPG algorithm than DDPG in different environments. This is because the UAV learns from higher quality experiences (expert experiences) at the beginning of training, which means that the parameters of the neural network are quickly adjusted to an optimal state. However, it is easy to find that the episode average reward of the two algorithms was not significantly different, and sometimes the average reward for the DDPG was higher than that of the MEP–DDPG. This shows that the action selected by the DDPG was no worse than the action selected by the MEP–DDPG in each step, and sometimes it is even better than the MEP–DDPG selected actions. However, at the end of an episode, the UAV trained by DDPG was not able to reach the target position. The models trained by the two algorithms were run 1000 times each, and the average speed of every episode was recorded (Figure 13):

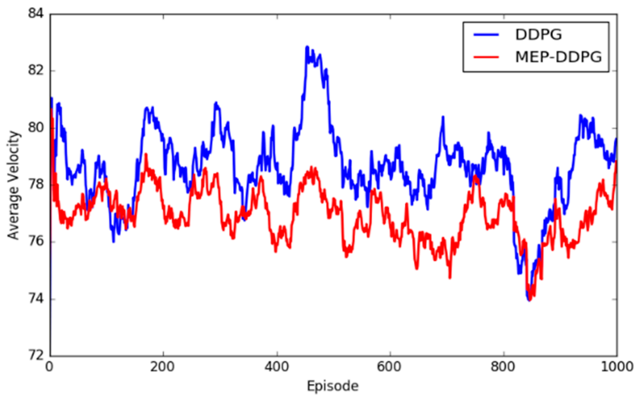

**Figure 13.** Average velocity of UAVs trained by DPPG and MEP–DDPG.

Obviously, the UAV trained by DDPG had a higher velocity than that trained by MEP–DDPG, which leads to the conclusion that in some cases, the UAV trained by DDPG cannot maneuver as safely as the UAV trained by MEP–DDPG when facing the same threats. This is demonstrated in Figures 14 and 15:

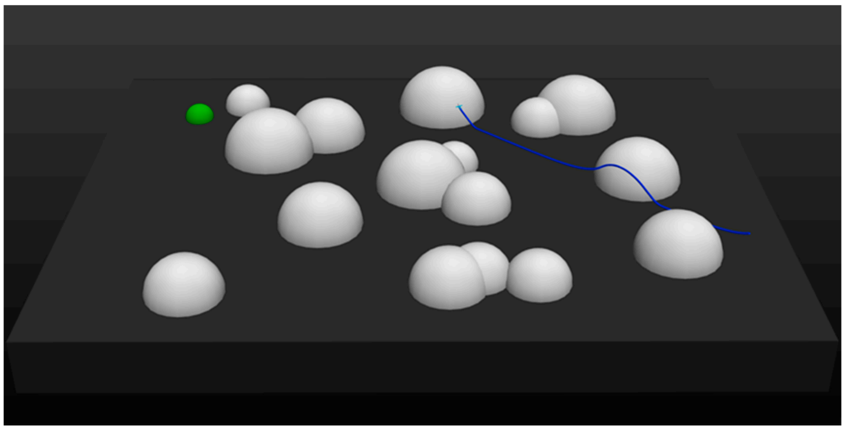

**Figure 14.** The trajectory of a UAV trained by DDPG.

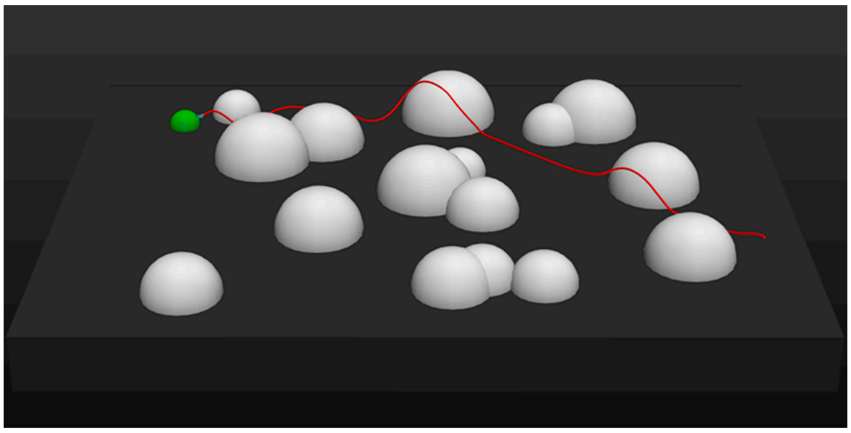

**Figure 15.** The trajectory of a UAV trained by MEP–DDPG.

It is evident that for the same task, the flight trajectories of UAVs trained by DDPG and MEP–DDPG in the first half are almost identical. However, due to the faster flight speed, the UAV trained by DDPG was unable to slow down and climb in a short period of time when facing threats, which resulted in a failure of the mission. In comparison, the UAV trained by the MEP–DDPG algorithm was able to easily climb to avoid threats and successfully complete tasks. These results demonstrate that DDPG is stuck in a local optimal solution, which leads to the excessively fast flight speed of the UAV. In the real world, flying at a higher velocity is good, because the UAV can complete the task faster; however, the success of the task and the safety of the UAV are more important. Therefore, the more cautious UAV trained by the MEP–DDPG algorithm may have a higher reference value.

### *6.2. Testing for Different Tasks*

To test the adaptability of the proposed algorithm, two tasks to simulate what might happen in a real unknown environment were designed.

#### 6.2.1. Testing for Tasks with Sudden Threats

In a real environment, changes in the dynamic environment can suddenly show up. For example, when a UAV is delivering an item, a car could suddenly pass by. Therefore, carrying out tasks in the presence of sudden threats was simulated:

As shown in Figure 16, the initial positions of the UAV and the target were (500, 10, 30), (−430, −5, 0). After the UAV bypassed the first threat at step = 250, a new threat suddenly appeared at (200, −5, 0) (the yellow hemisphere shown on the left in Figure 16b). When the step increased to 900, another threat suddenly appeared at (−310, 13, 0) (the yellow hemisphere shown on the right in Figure 16d). Judging from the trajectory of the UAV (red lines) in the figure, the UAV was able to make a good judgment and pass through the dangerous area safely when faced with sudden threats.

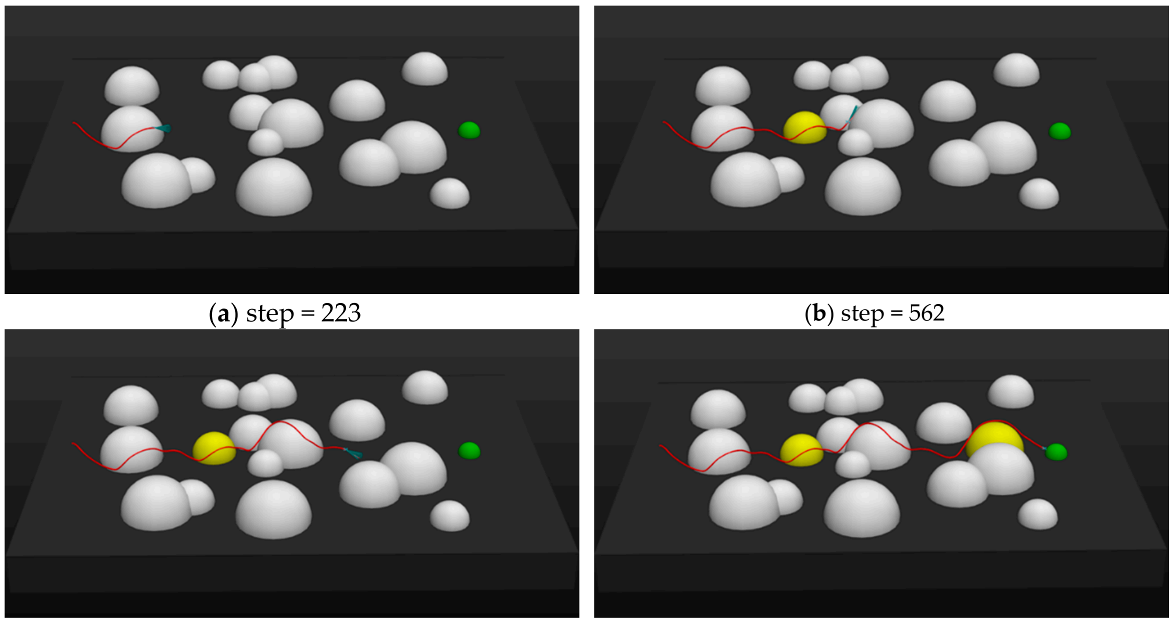
(**a**) step = 223 (**b**) step = 562
(**c**) step = 891 (**d**) step = 1125

**Figure 16.** The trajectory of the UAV at different time steps when performing a task with sudden threats.

#### 6.2.2. Testing for Tasks with Moving Target

This test was used to simulate the moving target problem in UAV's AMP. In many application scenarios, the target area to be reached can suddenly changes. Changes of the target area may cause the failure of the UAV's task. The following test can simulate this situation. Figure 17 presents the testing process:

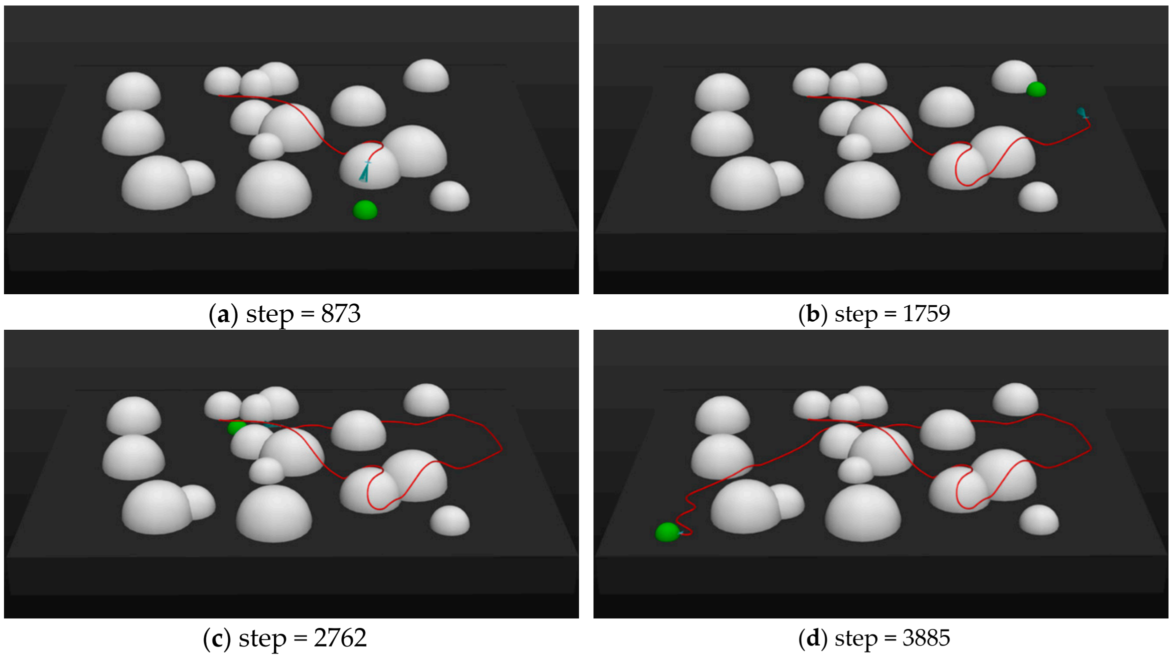
(**a**) step = 873 (**b**) step = 1759 (**c**) step = 2762 (**d**) step = 3885

**Figure 17.** The trajectory of the UAV at different time step when performing task with moving target.

As shown in the figures above, the initial positions of the UAV and the target were (200, 200, 30), (−170, −410, 0). In Figure 17a, at step = 873, the UAV was very close to the target, but then the target position suddenly changed to (−450, 300, 0) when step = 900. The UAV quickly turned to adjust the attitude and fly to the new target position (Figure 17b). When step = 1800, the position of target changed again to (150, 200, 0) (Figure 17c). When step = 2700, the position of the target changed for the third time to (500, −400, 0) (Figure 17d). The UAV's entire flight trajectory (red lines) demonstrated that, even though the position of the target changed several times, the UAV was still able to reach the target position safely and accurately. Therefore, the moving target problem was addressed successfully.

## 7. Conclusions and Future Work

This paper proposed a MEP–DDPG algorithm to address UAV's AMP problem. By using an MPC-SA framework to generate expert experiences and multiple experience pools to store expert experiences, MEP–DDPG can go beyond the local optimum generated during the early stages of training. Experimental results show that the convergence speed and results of the MEP–DDPG algorithm are better than those of the DDPG algorithm in several different complex unknown environments. In the testing process, MEP–DDPG performed well in some common tasks that may occur in real situations.

In the future, we intend to add additional complex tasks such as multi-target tracking to our 3D simulation environment. In addition, we plan to make the location and scope of the threat change over time to more realistically simulate some application environments.

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

**Funding:** This research was funded by (1) the National Natural Science Foundation of China, grant number 61573285; (2) Science and Technology on Avionics Integration Laboratory and Aeronautical Science Foundation of China (ASFC), grant number 20175553027.

**Acknowledgments:** The authors are grateful to WAN and GAO for discussions.

**Conflicts of Interest:** The authors declare that they have no conflicts of interest.

## References

1. Stevens, R.C.; Sadjadi, F.A.; Braegelmann, J.R.; Cordes, A.M.; Nelson, R.L. Small unmanned aerial vehicle (UAV) real-time intelligence, surveillance, and reconnaissance (ISR) using onboard pre-processing. *SPIE Def. Secur. Symp.* **2008**, *6967*, 1–8.
2. Liu, Y.; Luo, Z.; Liu, Z.; Shi, J.; Cheng, G. Cooperative Routing Problem for Ground Vehicle and Unmanned Aerial Vehicle: The Application on Intelligence, Surveillance, and Reconnaissance Missions. *IEEE Access* **2019**, *7*, 63504–63518. [CrossRef]
3. Kuru, K.; Ansell, D.; Khan, W.; Yetgin, H. Analysis and Optimization of Unmanned Aerial Vehicle Swarms in Logistics: An Intelligent Delivery Platform. *IEEE Access* **2019**, *7*, 15804–15831. [CrossRef]
4. Yang, N.K.; San, K.T.; Chang, Y.-S. A novel approach for real time monitoring system to manage UAV delivery. In Proceedings of the 5th IIAI-AAI, Kumamoto, Japan, 10–14 July 2016.
5. Zheng, H.; Su, X.; Lin, Y.; Zhou, Z.; Sun, H. A somatosensory-operation-based unmanned aerial vehicle for power line inspection in first person of view. In Proceedings of the 11th PowerCon, Guangzhou, China, 6–8 November 2018.
6. Nguyen, V.N.; Jenssen, R.; Roverso, D. Intelligent monitoring and inspection of power line components powered by UAVs and deep learning. *IEEE Power Energy Technol. Syst. J.* **2019**, *6*, 11–21. [CrossRef]
7. Aljehani, M.; Inoue, M. Comparison of UAV flight behaviors after autonomous mapping of an urban brea. In Proceedings of the 7th IEEE GCCE, Nara, Japan, 9–12 October 2018.
8. Rokhmana, C.; Andaru, R. Utilizing UAV-based mapping in post disaster volcano eruption. In Proceedings of the 6th InAES, Yogyakarta, Indonesia, 1–3 August 2016.
9. Yang, X.; Ding, M.-Y.; Zhou, C.-P. Fast marine route planning for UAV using improved sparse A* algorithm. In Proceedings of the 4th IEEE ICGEC, Taiwan/Xiamen, China, 29 August–1 September 2011.
10. Wang, Z.; Liu, L.; Long, T.; Yu, C.; Kou, J. Enhanced sparse A* search for UAV path planning using dubins path estimation. In Proceedings of the 33rd IEEE CCC, Nanjing, China, 28–30 July 2014.
11. Tianzhu, R.; Rui, Z.; Jie, X.; Zhuoning, D. Three-dimensional path planning of UAV based on an improved A* algorithm. In Proceedings of the IEEE CGNCC, Nanjing, China, 12–14 August 2016.
12. Fei, W.; Jin-Qiang, C.U.I.; Ben-Mei, C.H.E.N.; Tong, H.L. A comprehensive UAV indoor navigation dystem based on vision optical flow and laser FastSLAM. *Acta Autom. Sin.* **2013**, *39*, 1889–1990.
13. Cui, J.Q.; Lai, S.; Dong, X.; Liu, P.; Chen, B.M.; Lee, T.H. Autonomous navigation of UAV in forest. In Proceedings of the ICUAS, Orlando, FL, USA, 27–30 May 2014.
14. Thrun, S.; Montemerlo, M. The graph slam algorithm with applications to large-scale mapping of urban structures. *Int. J. Robot. Res.* **2006**, *25*, 403–429. [CrossRef]
15. Gee, T.; James, J.; Mark, W. Lidar guided stereo simultaneous localization and mapping (slam) for UAV outdoor 3D scene reconstruction. In Proceedings of the 2016 ICIVC, Palmerston North, New Zealand, 21–22 November 2016.
16. Santos, S.R.B.D.; Givigi, S.N.; Junior, C.L.N. An experimental validation of reinforcement learning applied to the position control of UAVs, Systems, Man, and Cybernetics (SMC). In Proceedings of the IEEE ICSMC, Seoul, Korea, 14–17 October 2012.
17. Yijing, Z.; Zheng, Z.; Xiaoyi, Z.; Yang, L. Q learning algorithm based UAV path learning and obstacle avoidence approach. In Proceedings of the 36th IEEE CCC, Dalian, China, 26–28 July 2017.
18. Pham, H.X.; La, H.M.; Feil-Seifer, D. Reinforcement learning for autonomous uav navigation using function approximation. In Proceedings of the IEEE International Symposium on Safety, Security, and Rescue Robotics SSRR, Philadelphia, PA, USA, 6–8 August 2018.
19. Lin, X.; Yu, Y.; Sun, C.Y. Supplementary reinforcement learning controller designed for quadrotor UAVs. *IEEE Access* **2019**, *7*, 26422–26431. [CrossRef]
20. 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]

21. Zhu, Y.; Mottaghi, R.; Kolve, E.; Lim, J.J.; Gupta, A.; Fei-Fei, L.; Farhadi, A. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Singapore, Singapore, 29 May–3 June 2017.
22. Zhang, W.; Song, K.; Rong, X.; Li, Y. Coarse-to-Fine UAV Target Tracking With Deep Reinforcement Learning. *IEEE Trans. Autom. Sci. Eng.* **2019**, *16*, 1522–1530. [CrossRef]
23. Bohn, E.; Coates, E.M.; Moe, S.; Johansen, T.A. Deep reinforcement learning attitude control of fixed-wing uavs using proximal policy optimization. In Proceedings of the IEEE International Conference on Unmanned Aircraft Systems (ICUAS), Atlanta, GA, USA, 11–14 June 2019.
24. Tai, L.; Paolo, G.; Liu, M. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 24–28 September 2017.
25. Pfeiffer, M.H.; Shukla, S.; Turchetta, M.; Cadena, C.; Krause, A.; Siegwart, R.; Nieto, J.; Lerma, C.C. Reinforced imitation: Sample efficient deep reinforcement learning for mapless navigation by leveraging prior demonstrations. *IEEE Robot. Autom. Lett.* **2018**, *3*, 4423–4430. [CrossRef]
26. Han, X.; Wang, J.; Xue, J.; Zhang, Q. Intelligent decision-making for 3-dimensional dynamic obstacle avoidance of UAV based on deep reinforcement learning. In Proceedings of the 11th WCSP, Xi'an, China, 23–25 October 2019.
27. Kjell, K. Deep Reinforcement Learning as Control Method for Autonomous UAV. Master's Thesis, Polytechnic University of Catalonia, Barcelona, Spain, 2018.
28. Singla, A.; Padakandla, S.; Bhatnagar, S. Memory-based deep reinforcement learning for obstacle avoidance in UAV with limited environment knowledge. *IEEE Trans. Intell. Transp. Syst.* **2019**. [CrossRef]
29. Polvara, R.; Patacchiola, M.; Hanheide, M.; Neumann, G. Sim-to-Real quadrotor landing via sequential deep Q-Networks and domain randomization. *Robotics* **2020**, *9*, 8. [CrossRef]
30. Wang, C.; Wang, J.; Shen, Y.; Zhang, X. Autonomous navigation of UAVs in large-scale complex environments: A deep reinforcement learning approach. *IEEE Trans. Veh. Technol.* **2019**, *68*, 2124–2136. [CrossRef]
31. Rodriguez-Ramos, A.; Sampedro, C.; Bavle, H.; De La Puente, P.; Campoy, P. A Deep Reinforcement Learning Strategy for UAV Autonomous Landing on a Moving Platform. *J. Intell. Robot. Syst.* **2018**, *93*, 351–366. [CrossRef]
32. Van, H.H.; Guez, A.; Silver, D. Deep reinforcement learning with double Q-learning. In Proceedings of the AAAI Conference on Artificial Intelligence, Phoenix, AZ, USA, 12–17 February 2016.
33. Singla, A.; Padakandla, S.; Bhatnagar, S. Dueling network architectures for deep reinforcement learning. *IEEE Trans. Intell. Transp. Syst.* **2016**, *48*, 1995–2003.
34. Hausknecht, M.; Stone, P. Deep recurrent q-learning for partially observable mdps. In Proceedings of the 2015 AAAI Fall Symposium Series, Arlington, VA, USA, 12–14 November 2015.
35. Gao, X. *Introduction to Aviation Military Vehicle*; Northwestern Polytechnical University Press: Xi'an, China, 2004.
36. Bellman, R. A Markovian decision process. *Indiana Univ. Math. J.* **1957**, *6*, 679–684. [CrossRef]
37. Sutton, S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 1998.
38. Liu, Q.; Zhai, J.-W.; Zhang, Z.-Z.; Zhang, S. A survey on deep reinforcement learning. *Chin. J. Comput.* **2018**, *41*, 1–27.
39. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. *arXiv* **2016**, arXiv:1509.02971v6.
40. Watkins, C.J.C.H.; Dayan, P. Q-learning. *Mach. Learn.* **1992**, *8*, 279–292. [CrossRef]
41. Sutton, R.S.; McAllester, D.A.; Singh, S.P.; Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*; MIT Press: Cambridge, MA, USA, 2000; pp. 1057–1063.
42. Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; Riedmiller, M. Deterministic policy gradient algorithms. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 7–12 December 2015; pp. 2944–2952.
43. Wan, K.; Gao, X.; Hu, Z.; Wu, G. Robust motion control for UAV in dynamic uncertain environments using deep reinforcement learning. *Remote Sens.* **2020**, *12*, 640. [CrossRef]
44. Wang, L. Model predictive control: Design and implementation using MATLAB (T-3). In Proceedings of the American Control Conference, St. Louis, MO, USA, 10–12 June 2009; pp. 25–26.

45. Camacho, E.F.; Bordons, C. *Model Predictive Control*; Springer: London, UK, 2004.
46. Allgöwer, F.; Zheng, A. *Nonlinear Model Predictive Control*; Birkhäuser: Basel, Switzerland, 2005; Volume 152, pp. 257–258.
47. Kirkpatrick, S.; Gelatt, C.D.; Vecchi, M.P. Optimization by simulated annealing. *Science* **1983**, *220*, 671–680. [CrossRef] [PubMed]