Asynchronous Curriculum Experience Replay: A 
Deep Reinforcement Learning Approach for UAV 
Autonomous Motion Control in Unknown 
Dynamic Environments

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

Abstract—Unmanned aerial vehicles (UAVs) have been widely used in military warfare, and realizing safely autonomous motion control (AMC) in complex unknown environments is a challenge to face. In this article, we formulate the AMC problem as a Markov decision process (MDP) and propose an advanced deep reinforcement learning (DRL) method that allows UAVs to execute complex tasks in different environments. Aiming to overcome the limitations of the prioritized experience replay (PER), the proposed asynchronous curriculum experience replay (ACER) uses multithreads to asynchronously update the priorities and assigns the true priorities to increase the diversity of experiences. It also applies a temporary pool to enhance learning from new experiences and changes the fashion of experience pool to first-in-useless-out (FIUO) to make better use of old experiences. In addition, combined with curriculum learning (CL), a more reasonable training paradigm is designed for ACER to train UAV agents smoothly. By training in a large-scale dynamic environment constructed based on the parameters of a real UAV, ACER improves the convergence speed by 24.66% and the convergence result by 5.59% compared to the twin delayed deep deterministic policy gradient (TD3) algorithm. The testing experiments carried out in environments with different complexities further demonstrate the strong robustness and generalization ability of the ACER agents.

Index Terms—Autonomous motion control, curriculum learning, deep reinforcement learning, experience replay, UAV.

I. INTRODUCTION

With the rapid development of unmanned aerial vehicle (UAV) technology, UAVs have been widely used in military wars in recent years. The characteristics of UAVs, such as low cost, strong survivability and high operational efficiency, make them perform well in executing tasks such as intelligence, surveillance, and reconnaissance [1], [2], electronic countermeasures [3], [4], and ground attacks [5], [6]. To accomplish these tasks successfully, UAVs usually need to achieve autonomous motion control (AMC) in complex and changeable unknown environments. Taking the ground attack mission under the anti-access/area denial strategic concept as an example (Fig. 1), UAV needs to use terrain following-terrain avoidance technology to make full use of the earth curvature, terrain occlusion, and the blind spot of enemy integrated air-defense system as a cover, infiltrate the enemy area quickly and covertly for a surprise attack.

The results of UAV AMC will directly affect the results of the combat mission, so a long-term goal of UAV applications is to develop a technique that can enable UAVs to fly safely and accurately without human intervention. As a popular research topic, UAV AMC has aroused widespread interest, and a series of methods have been proposed to address it. These methods can be generally classified into two groups: nonlearning-based and learning-based [7]. Many studies [8], [9] used nonlearning-based methods such as the A* algorithm for UAV route planning and performed well when information for the entire environment is known. To adapt to unknown or partially known environments, another class of nonlearning-based methods [10], [11] resorts to simultaneous localization and mapping to control UAVs. Since

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply. 
such methods need to model the environment based on observational information, once the environment changes, modeling will lead to unaffordable computational costs [12]. Therefore, it is necessary to find more efficient methods to control UAVs in highly dynamic unknown environments.

Reinforcement learning (RL), an online learning-based method, has been introduced by many studies [13], [14], [15] to address the AMC problem. Li et al. [13] presented a novel route planning algorithm based on Q-learning [16], which improves the convergence speed by using prior knowledge to guide the UAV in selections. Hung et al. [14] applied Q-learning to flocking so that UAVs can learn how to flock in a simulated nonstationary stochastic environment. As the complexity of the environment continues to increase, deep reinforcement learning (DRL), a new area of intense interest that combines the perceived capabilities of deep learning and the decision-making capabilities of RL, has been proven to be a more effective approach to solve the problem. Asheralieva et al. [17] combined deep Q-learning (DQN) [18] with the Bayesian deep learning to propose an unsupervised algorithm to achieve more efficient UAV control with incomplete information. Based on double DQN (DDQN) [19], Liu et al. [20] introduced a quality-of-service-based action selection policy to ensure the UAV with limited energy dynamically plans its trajectory as a mobile edge server to serve the mobile terminal users. To achieve a better utility of controlling UAV, Chen et al. [21] developed an online DRL scheme that adopts two separate DDQN to approximate the Q-factor and the post-decision Q-factor, respectively. However, the action space of UAVs in the battlefield environment is continuous, which requires policy-based DRL methods to control the UAV. Based on the deep deterministic policy gradient (DDPG) [22] algorithm, Ding et al. [23] proposed an algorithm named energy-efficient fair communication with trajectory designs and band allocation to implement UAV control in auxiliary communication tasks. For the UAV ground target tracking task, Li et al. [24] introduced an improved DDPG algorithm that uses long short-term memory networks to approximate the state of environments and improves the approximation accuracy and the efficiency of data utilization.

New challenges arise when UAVs perform AMC in realistic large-scale environments: the limited number of experiences. In military warfare, the preparation time for UAVs to perform each combat mission is usually limited. Limited training time results in limited experience. Therefore, it is necessary to design an algorithm to use limited experience to train the best UAV. To solve this problem, usually one of two approaches is chosen: 1) Create more reliable experiences [25], [26]. Ensuring the accuracy and reliability of the generated experiences in this approach, however, is difficult. 2) Use limited experiences more efficiently. This method is the focus of this article, and it is also a research hotspot of DRL: experience replay (ER).

To mitigate the pressure on training caused by the lack of reliable experiences, Schaul et al. [27] first introduced a prioritized experience replay (PER) approach that greatly improves the convergence speed of DRL algorithms by replaying important experiences more frequently. Although PER has shown advantages in many scenarios of UAV control [28], [29], it still has the following limitations: 1) Updating the temporal difference error (TD error) too slowly that affects the quality of the sampled experiences. 2) Using the clip operation weakens the difference between experiences. 3) Maximum priority cannot ensure the priorities of the newest experiences. 4) Sampling according to the TD error alone may not be the best method.

Many studies have made efforts to solve these problems: Ren et al. [30] proposed a deep curriculum reinforcement learning (DCRL) method that combines DRL with curriculum learning (CL) [31] to adaptively select appropriate experiences based on the complexity of each experience. Our previous work [12] considered the relevance between the current state and experiences and presented a relevant experience learning (REL) method to use more indicators to replay experiences. These novel methods only solve the fourth limitation of PER to some extent, while this article attempts to fundamentally overcome all of the PER limitations and makes the following contributions:

1) The AMC problem of UAV in unknown complex environments is modeled as a Markov decision process (MDP). The state space, action space, and reward function are well designed for the MDP so that the problem can be addressed by different DRL algorithms.

2) Based on CL, a DRL framework for controlling UAVs in large-scale environments is developed for the first time. Under this framework, algorithms can change the training paradigm of agent to improve the robustness of the DRL algorithms.

3) An efficient ER algorithm, the asynchronous curriculum experience replay (ACER), is proposed to overcome all the limitations of PER. ACER uses a subthread to asynchronously update the priorities, assigns the true priorities and makes full use of the newest experiences and the stored experiences to improve the performance of DRL algorithms.

4) By combining with the state-of-the-art twin delayed deep deterministic policy gradient (TD3) algorithm [32], many experiments were conducted on the environments described in [12]. The training experiments show that ACER improves the convergence speed by 24.66% and the convergence result by 5.59% compared with vanilla TD3. Testing experiments in environments with different complexities prove the better robustness and generality of the ACER algorithm. In addition, the motion trajectories of the ACER agent show the practicality of the proposed algorithm for future use on real UAVs.

The remainder of this article is organized as follows. In Section II, some background knowledge of TD3 and PER is introduced. Section III formulates the UAV AMC problem as an MDP. The proposed algorithm is presented in Section IV. In Section V, training and testing experiments are presented, and the experimental results are discussed. Section VI concludes this article and envisages some future work.

II. BACKGROUND

In this section, we first describe the TD3 algorithm and its origin in detail. Then the classical ER algorithm PER and some
related improvements are introduced. Finally, the principle and advantages of CL are briefly presented.

A. Actor-Critic, DDPG, and TD3

Value-based methods such as DQN, have performed well in addressing problems with discrete action space. In continuous spaces, finding a greedy policy requires optimization at every timestep, but this optimization is too slow to be practical with large, unconstrained function approximators and is nontrivial [22]. Unlike value-based methods, policy-based methods approximate the policy directly so that they can perform well in searching action in continuous spaces.

Actor-critic [33], a category of policy-based methods for addressing MDPs [34], [35], seeks to learn the optimal policy \( a \sim \pi_\theta(s) \) by applying a stochastic policy gradient in the parameter space:

\[
\nabla_\theta J (\pi_\theta) = \mathbb{E}_{s \sim \rho, a \sim_\rho \pi_\theta} \left[ \nabla_\theta \log \pi_\theta(s, a) Q^\pi(s, a) \right],
\]

where \( \rho, a \) is the space of the sampled states and \( \pi_\theta \) is the action space. The stochastic policy gradient method needs many experiences to sample the whole action space, while the deterministic policy gradient methods select the action \( a = \mu_\theta(s) \) with the highest probability at every state, which reduces the amount of experience sampling and improves the efficiency of the algorithm. The gradient of the deterministic policy methods is as follows:

\[
\nabla_\theta J (\mu_\theta) = \mathbb{E}_{s \sim \rho^\mu} \left[ \nabla_\theta \mu_\theta(s) \nabla_\alpha Q^\mu(s, a) \right]_{a=\mu_\theta(s)},
\]

where \( \rho^\mu \) is the space of the sampled states. To optimize the training process, Lillicrap et al. [22] proposed an actor-critic method DDPG, which shows good results in solving MDPs with continuous action spaces. Derived from the techniques of DQN, both the actor and critic networks of DDPG contain two artificial neural networks with the same structure, called eval net and target net. The parameters of eval nets are updated more frequently than those of target nets to make the algorithm more stable [18]. In addition, DDPG adds independent noise \( \mathcal{N} \) to increase the randomness of the agent’s exploration:

\[
\alpha_t = \mu_t \left( s_t \mid \theta^\mu \right) + \mathcal{N}_t.
\]

The actor network uses policy gradient \( \nabla_\theta J \) to approximate the parameters of eval net while the critic network updates its eval net by minimizing the loss function \( L (\theta^Q) \):

\[
\nabla_\theta J \approx \frac{1}{N} \sum_i \nabla_\alpha Q(s, a \mid \theta^Q) \big|_{s=s_i, a=\mu(s_i)} \nabla_\theta \mu(s \mid \theta^\mu) \big|_{s},
\]

\[
L (\theta^Q) = \frac{1}{N} \sum_i \left( y_i - Q(s_i, a_i \mid \theta^Q) \right)^2,
\]

\[
y_i = r(s_i, a_i) + \gamma Q'(s'_{i}, \mu'(s'_{i} \mid \theta^{\mu'}), \theta^Q),
\]

where \( \theta^\mu, \theta^{\mu'}, \theta^Q \), and \( \theta^{\mu'} \) represent the parameters of the eval net in the actor network, the target net in the actor network, the eval net in the critic network, and the target net in the critic network, respectively, and \( N \) is the number of sampled experiences.

Based on the DDPG algorithm, Fujimoto et al. [32] introduced a novel policy-based method named TD3, which has been proven to be one of the state-of-the-art DRL algorithms. As shown in Fig. 2, the TD3 algorithm also uses an experience pool for storing and replaying old experiences. The actor network is used to determine the probability of choosing an action while the critic networks are used to evaluate the action selected by the agent based on the environmental state.

TD3 makes the following enhancements to improve the performance: 1) Double networks for overestimation: Like the DQN algorithm, TD3 proposes to use two sets of critic networks to calculate \( Q(s, a) \) and chooses the smaller set as the target. Therefore, (5) and (6) become:

\[
L (\theta^{Q_1}) = \min_{j=1,2} \frac{1}{N} \sum_i \left( y_i - Q_j(s_i, a_i \mid \theta^{Q_j}) \right)^2,
\]

\[
y_i = r(s_i, a_i) + \gamma \min_{j=1,2} Q_j(s', \mu(s' \mid \theta^{\mu'}), \theta^{Q_j}).
\]

2) Actor update delay for stability: Different from the synchronous updates of DDPG, TD3 allows critic networks to update more frequently than actor networks to make the training of actor more stable. 3) Action noise for smooth target policy: TD3 adds action noise \( \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -1, 1) \) to make an action random in a certain range when calculating \( Q \) to make the policy smooth and stable.

The parameter update method of TD3 is the same soft-updated method as that in the DDPG algorithm:

\[
\theta' = \tau \theta + (1 - \tau) \theta',
\]

where \( \tau \in [0, 1] \) determines the update degree of the network parameters.

B. PER

ER, a process of storing past sequential experiences and sampling them to reuse for updating policies, was first explored by Lin [36] in 1992 to speed up the training process. More recently, ER has seen widespread adoption, since it has been shown to be instrumental in the breakthrough success of DRL [18]. This new
area of intense research has attracted many studies to explore how ER can influence the performance of off-policy DRL algorithms [37, 38, 39]. ER is always a fixed-size experience pool that holds the most recent experiences collected by the agent. This first-in-first-out (FIFO) buffer brings two advantages: 1) uniform sampling breaks the correlation between experiences to improve the stability of the policy; 2) the large-capacity buffer ensures the possibility of learning from long-term experiences, thereby avoiding ‘catastrophic forgetting’ [40].

In the ER of DQN, the uniform sampling policy results in all experiences having the same probability of being sampled, which ignores the importance of each experience. PER [27] differs from this and assigns the priority of the experience \( e_i \) in the experience pool according to its TD error:

\[
\delta_i = r_i + \gamma \max_{a'_i} Q' \left( s'_i, a'_i \right) - Q \left( s, a \right),
\]

(10)

\[
p_i = |\delta_i| + \varepsilon.
\]

(11)

The sampling probability of \( e_i \) is defined as:

\[
P \left( e_i \right) = \frac{p_i^\alpha}{\sum_{j=1}^{D} p_j^\alpha},
\]

(12)

where \( D \) is the capacity of the experience pool and \( \alpha \in [0, 1] \) is a constant to control the size of \( p_j^\alpha \). These ensure that the greater the TD error is, the higher the priority of the experience is and the greater the probability is that the experience can be sampled. To reduce the computational complexity of searching for experiences with higher priorities, PER adopts a ‘sum-tree’ structure to improve search efficiency and ensure the priority and diversity of the sampled experiences at the same time [41]. Following the innovation of PER, Brittain et al. [42] believed that the previous experiences leading to the important experiences with higher TD error should also be assigned higher priorities and proposed the prioritized sequence experience replay algorithm.

In addition, some studies try to improve the PER algorithm from different perspectives. Some of them believed that TD error should not be the only criterion for measuring the importance of experience, and other indicators, such as reward [43, 44], difficulty [30] and the relevance between experiences and current state [12], should also be considered. Ren et al. [30] first introduced CL into ER and proposed DCRL to use complexity and coverage penalties to control the experiences learned by the agent. These methods improved the convergence speed of PER to a certain extent, but they did not fundamentally overcome the limitations of the PER algorithm.

C. CL

Learning from simple to difficult is the universal order of human learning because simple knowledge always lays the foundation for more difficult future learning. Motivated by this learning process, Bengio et al. [31] proposed CL in 2009, which advocates letting the model learn from easier samples and gradually increasing the complexities of the samples. Generally, CL assigns different weights to different training samples according to the difficulty of the samples. In the initial stage of training, the weights of simple samples are the highest. As the training process progresses, the weights of difficult samples will be gradually increased. This process of dynamically assigning weights to the samples is called the curriculum. The definition of the curriculum is not fixed, and different evaluation criteria of sample difficulty can be set for different problems. Due to the two benefits of less training time and better model generalization ability brought by CL, CL has been widely used in computer vision [45], natural language processing [46], DRL [47, 48] and other fields.

All these applications demonstrate that as a flexible plug-and-play submodule independent of the original training, CL is easy to use to speed up the learning process, especially for deep neural networks [49]. In this article, CL is introduced for the first time to prescribe the order of the experiences learned by the UAV to better solve the AMC problem by using DRL methods.

III. PROBLEM FORMULATION

In this section, we formulate the UAV AMC problem in complex unknown environments. The UAV model is introduced first. Then, the details of state space, action space, and reward function are presented to model the MDP.

A. UAV Model

UAV is usually equipped with autopilot system to provide low-level flight control, that is, the autopilot system controls the thrust system and wings to generate the required dynamic parameters, so as to realize the control of altitude, forward and vertical speeds, and the pitch attitude, and finally drive the aircraft to fly stably or track the desired path [50]. Because RL algorithm usually provides the waypoints needed by UAV to achieve AMC, this article only considered the high-level flight control model of UAV and used the load factor \( n_0 \) as the input to control the UAV. According to reference [51, 52, 53], assuming that \( N \) is the sum of all external forces acting on the UAV except gravity, the definition of \( n_0 \) is as follows:

\[
n_0 = \frac{N}{G},
\]

(13)

where \( G \) is the dimensionless value of gravity \( G \). The centroid acceleration \( a_0 \) of UAV can be expressed as:

\[
a_0 = \frac{N + G}{m_u},
\]

(14)

where \( m_u \) is the mass of UAV. And \( a_0 \) can be represented by \( n_0 \):

\[
a_0 = n_0 \cdot g + g.
\]

(15)

where \( g \) is the dimensionless value of gravitational acceleration \( g \). Then the velocity \( v_u \) and position \( p_u \) of the UAV can be further calculated:

\[
v_u = v_0 + \int_t a_0 dt,
\]

(16)

\[
p_u = p_0 + \int_t v_u dt.
\]

(17)

where \( v_0 \) and \( p_0 \) is the velocity and position at the previous timestep, respectively.
Assuming that the velocity direction of the UAV is always the same as the UAV axis (shown as the red arrow in Fig. 3(a)), the pitch angle $\theta_u$ and yaw angle $\varphi_u$ of the UAV at the current timestep can be calculated according to the velocity direction:

$$\theta_u = \arctan \left( \frac{v_{uz}}{\sqrt{v_{ux}^2 + v_{uy}^2}} \right),$$

(18)

$$\varphi_u = \arctan \left( \frac{v_{uy}}{v_{ux}} \right).$$

(19)

### B. MDP Modeling

An MDP is always represented as a 4 tuple $(S, A, P, R)$: $S$ is a set of all states that the agent can get in the environment; $A$ represents the set of all actions that can be chosen by the agent in the environment; $P$ is the probability of executing an action $a$ from state $s$ to $s'$ (where $a \in A$ and $s, s' \in S$); $R$ represents the reward of performing action $a$ at state $s$ [34]. The UAV AMC problem is a typical sequential decision-making problem that can be modeled as MDP.

1) State Space and Action Space Specification: For autonomous control, the UAV shall at least be capable of collecting information from three sources, i.e., the information of its own state, the information observed from the environment, and the information of the target.

In a real combat mission, the position, velocity, and attitude of the UAV can be provided by onboard GPS, sensors, and gyro-scope devices. In this article, the state of the UAV is defined as a dimensional vector state $s$ in 3D environments; thus, the action $a$ of the MDP can be designed as $a = [n_{ux}, n_{uy}, n_{uz}]^T$.

2) Reward Design: Reward acts as the only criterion for evaluating how good the action chosen by the agent at a certain state is. The design of the reward function has a huge impact on the convergent motion policy of the agent. A reasonable reward function can speed up the convergence, and an unreasonable reward function may cause the algorithm to not converge. When controlling UAV in large-scale environments, due to the initial policy of DRL is randomly generated, algorithms would take an extremely long time to converge if a sparse reward function is designed [7]. In this article, a nonsparse reward function that incorporates domain knowledge about the AMC problem is introduced. This nonsparse reward function consists of five parts, namely, position reward, angle reward, height reward, obstacle penalty, and velocity reward.

The mission of the UAV is to approach the target, which means that the actions that make the UAV’s position closer to the target should be rewarded. Inspired by the references [7], [23], the position reward is designed as:

$$r_p = \frac{D_{pre} - D_{cur}}{k_p},$$

(20)

where $D_{pre}$ and $D_{cur}$ are the distances between the UAV and the target at the previous timestep and the current timestep, respectively. $k_p$ is a constant to normalize $r_p$.

To get close to the target as quickly as possible, an angle reward is necessary to ensure that the UAV flies in the direction of the target at all times [55]:

$$r_a = -\left( \frac{\varphi_{at} + \theta_{at}}{\kappa_a} \right),$$

(21)

where $\varphi_{at}$ and $\theta_{at}$ are the yaw and pitch angles between the UAV and the target, respectively, which are shown in Fig. 3(c) and (d), and $\kappa_a$ is a constant to normalize $r_a$.

The existence of a multipath effect makes the probability of radar finding low-altitude targets low or even zero. In addition, terrain occlusion such as mountains can also effectively reduce the risk of UAV being detected by enemy radar. The actions that make the height of the UAV lower should be rewarded:

$$r_h = 1 - \frac{p_{uz}}{H_{env}},$$

(22)

where $H_{env}$ is the maximum height of the environment.

In practice, colliding with obstacles could be catastrophic for UAVs [7]. As shown in Fig. 3(b) and (c), the distance $d_l$ between UAV and obstacles detected by the TAR is used in this article as an indicator to evaluate the danger degree of the current state. Those actions that make the average detected relative distance $d_{ave}$ of the radar rays smaller should be punished more:

$$r_d = d_{ave} - \sum_{i=1}^{N_t} \frac{d_i}{d_{ml}}/N_t,$$

(23)

where $d_{ml}$ is the maximum detection distance of the TAR.
When performing tasks in adversarial dynamic battlefield environment, the longer the flight time, the greater the risk the UAV will bear. UAVs usually need to complete the task as soon as possible under the premise of ensuring their own safety [7], [56]. This article sets a velocity upper limit $v_{\text{max}}$ to ensure that the flight states of the UAV conform to the actual situation, and gives greater rewards to those actions that make the UAV faster, so as to improve the execution efficiency of combat tasks:

$$ r_v = \frac{\|u_d\|}{v_{\text{max}}} $$  \hspace{1cm} (24)

In addition to the abovementioned influencing rewards, the UAV will also obtain a reward $r_a$ when the mission succeeds (reaching the target area) or receive a penalty $r_f$ when the mission fails (colliding). To avoid the UAV flying too far away from the target according to its nonconvergent policy, being out of the setting range is also considered as mission failure. In summary, the complete reward function is designed as follows:

$$ r(s, a) = \begin{cases} 
\begin{align*}
& r_a & \text{succeeds} \\
& r_f & \text{fails} \\
& \lambda_1 r_p + \lambda_2 r_a + \lambda_3 r_h + \lambda_4 r_d + \lambda_5 r_v & \text{every step}
\end{align*}
\end{cases} $$  \hspace{1cm} (25)

where $\lambda_1, \lambda_2, \lambda_3, \lambda_4, $ and $\lambda_5$ are constants used to tune the contribution rates of different rewards. Among these five rewards, $r_d$ directly determines whether the UAV can avoid the obstacles, so it should be assigned a larger coefficient $\lambda_4$ to improve the safety of flight. In fact, $r_h$ and $r_v$ are somehow included in $r_p$. To make the UAV’s motion trajectory more in line with mission requirements, $r_h$ and $r_v$ are still considered but assigned smaller coefficients $\lambda_3$ and $\lambda_5$, respectively. The detailed settings for these parameters are introduced in later sections.

IV. ASYNCHRONOUS CURRICULUM EXPERIENCE REPLAY

In this section, asynchronous experience replay (AER) is first introduced to overcome the limitations of the PER algorithm. Then, curriculum experience replay (CER) is designed to integrate CL with DRL and address a key problem of the proposed AER. Subsequently, the implementation of the integrated ACER algorithm is presented. Finally, the time complexity of the ACER is discussed.

A. Asynchronous Experience Replay

1) Asynchronous TD Error Updating: The PER algorithm uses the TD errors of different experiences to give them different priorities and then samples and learns according to the priorities, thereby accelerating the convergence speed of the DRL algorithms. In PER, the priorities of the stored experiences are updated through the learning process; that is, an experience will be given a new priority only when it is sampled; otherwise, its priority will remain unchanged forever. This sampled-updated priority updating method caused only the priorities of $N$ experiences to be determined according to the importance of the experience to the current network when sampled.

For a stored experience $e_i$ whose priority has not been updated, its current sampling probability is:

$$ P(e_i) = \frac{p_i}{\sum_{j=1}^D p_j} = \frac{p_i}{\sum_{j=1}^{i-1} p_j^a + p_i^a + \sum_{j=i+1}^D p_j^a} $$  \hspace{1cm} (26)

If the priority of $e_i$ is updated to the real priority $\hat{p}_i$, its real sampling probability is:

$$ \hat{P}(e_i) = \frac{\hat{p}_i}{\sum_{j=1}^{i-1} p_j^a + \hat{p}_i + \sum_{j=i+1}^D p_j^a} $$  \hspace{1cm} (27)

The difference between the real sampling probability $\hat{P}(e_i)$ and the current sampling probability $P(e_i)$ can be calculated:

$$ \hat{P}(e_i) - P(e_i) = \frac{(\hat{p}_i - p_i) \left( \sum_{j=1}^{i-1} p_j^a + \sum_{j=i+1}^D p_j^a \right)}{\left( \sum_{j=1}^{i-1} p_j^a + \hat{p}_i + \sum_{j=i+1}^D p_j^a \right) \left( \sum_{j=1}^{i-1} p_j^a + \hat{p}_i + \sum_{j=i+1}^D p_j^a \right)} $$  \hspace{1cm} (28)

Since $p_j > 0$ and $1 \geq \alpha \geq 0$, when $\hat{p}_i > p_i$, $\hat{P}(e_i) > P(e_i)$; when $\hat{p}_i < p_i$, $\hat{P}(e_i) < P(e_i)$. This reveals how the priority distribution of the stored experiences in PER is unreasonable: some of the non-updated experiences should have a larger sampling probability but are ignored due to their past-given low priorities, while some other experiences with high priorities should have a smaller sampling probability and are sampled even if they have less effect on the current network.

To solve this problem, this article proposes adopting multithreading technology and opening a subthread to update the priorities of the stored experiences in order. The main thread and the subthread correspond to a network for avoiding resource access conflicts. These two networks have the same structure and initialization mode. As shown in Fig. 4, the proposed AER algorithm separates the updating process from the learning process. The main thread is responsible for the learning process to update the policy, while the subthread is responsible for the updating process to make the priority distribution of all stored experiences meet the current network requirements as much as possible. After each learning, the parameters of the main thread’s network are copied to the subthread’s network to ensure that the two networks remain consistent.
A simple experiment is conducted with ‘CartPole-v0’ and the DQN algorithm. The priority distribution of experiences in the entire experience pool at different training stages is generated to show the effectiveness of AER. As shown in Fig. 5, we use \( p_{\text{real}} \), the real priority distribution of all stored experiences at the current timestep, as the standard to measure the accuracy of the priority distributions. At different stages of training, there is a small gap between the priority distributions \( p_{\text{AER}} \) and \( p_{\text{real}} \), and this gap decreases as the networks gradually converge. In comparison, the gap between \( p_{\text{PER}} \) and \( p_{\text{real}} \) is obviously larger from beginning to end. This asynchronous TD error updating method of AER allows all experiences to be updated in a short time to obtain the latest priorities. In this way, the gap between the current priority distribution and the \( p_{\text{real}} \) can be narrowed, and the advantages of importance sampling can be given full play to improve the convergence speed of the algorithm.

2) True Priority Needs to be Considered: A good ER algorithm can achieve a balance between the priority and diversity of the sampled experiences [41]. For stability reasons, the PER algorithm clipped the TD errors to fall within \([-1, 1]\) [27]. This clip operation ensures the stability of the algorithm and prevents excessive priority from affecting the diversity of sampling. However, it has a great impact on sampling, that is, all priorities greater than 1 will be clipped to 1, no matter how large it was. It can be seen from Fig. 5 that there are many experiences with priority \( p = 1 \) in each period of training, and the probability of these experiences being sampled is completely equal; that is, sampling in these experiences regresses to uniform sampling. There is no doubt that abandoning the true priority will have a huge impact on the quality of the sampled samples, thereby slowing the convergence speed of the algorithm.

The above problem can be solved in two ways: 1) Set different clip ranges for different environments, although doing so will introduce some task-dependent hyperparameters that need careful tuning. 2) Use other methods to replace the clip operation. To maximize the priority while ensuring diversity, this article abandons the clip operation, assigns experience to the true priority based on its TD error, and proposes a method combined with CL to avoid the appearance of outliers. This method will be introduced in detail in later sections.

3) Newest is Useful: Novel experiences, the experiences that have not been learned, are quite important, because the actual process of DRL is to optimize the policy through learning the stored experiences, thereby generating novel experiences for further learning, and to continuously loop this process until the optimal policy is found. Novel experiences can be divided into two categories: 1) a novel state with any action and 2) an old state with a novel action. For the first case, the agent has never reached these states, so it has not learned what motion policy should be taken at these states. Learning from these novel experiences can improve the motion policy and promote the agent to explore a wider range of the environment. The generation of this kind of novel experience mainly comes after the second kind of novel experience: the agent executes novel actions at the old state to reach novel states.

Assuming that at any timestep during the training process, for a state \( s \), \( a_1 \) is the optimal action. The agent should perform action \( a_1 \) at state \( s \) and generate the old experience \( (s, a_1, r_1, s_1) \). However, due to the existence of \( \varepsilon \)-greedy and exploration noise, the agent chooses a novel action \( a_2 \) and gains novel experience \( (s, a_2, r_2, s_2) \). Taking the DQN algorithm as an example, the TD errors of these two experiences are:

\[
\delta_1 = r_1 + \gamma \max_{a_1'} Q'(s_1, a_1') - Q(s, a_1),
\]

\[
\delta_2 = r_2 + \gamma \max_{a_2} Q'(s_2, a_2) - Q(s, a_2).
\]

For the problem of high-dimensional and large-scale state space such as UAV AMC, the change of state \( s \) by an action \( a \) in a timestep can be ignored, so we have \( \max_{a_1'} Q'(s', a_1') \approx \max_a Q'(s, a) \). Assuming \( \delta_1 > 0 \), we can get

\[
\delta_2 - \delta_1 \approx r_2 - r_1 + Q(s, a_1) - Q(s, a_2).
\]

1) When \( a_2 \) is a better action, that is, \( r_1 < r_2 \): Since \( a_1 \) is the optimal action until now and the novel experience \( (s, a_2, r_2, s_2) \) has not been learned, we have

\[
Q(s, a_1) = \max_a Q(s, a) > Q(s, a_2),
\]

and \( \delta_2 > \delta_1 \).

2) When \( a_2 \) is a worse action, that is, \( r_1 > r_2 \):
The DQN algorithm uses a neural network to update the Q value, where each update of $Q(s, a)$ can still be expressed by

$$Q(s, a) = Q(s, a) + \alpha \left[ r + \gamma \max_{a'} Q'(s', a') - Q(s, a) \right]. \quad (32)$$

However, due to the parameter update of the neural network, the update of a Q value will have a certain impact on other Q values. For a novel experience $(s, a_2, r_2, s_2)$ that has not been learned before, its Q value is also constantly changing in the past learning process.

Here, we make a reasonable assumption that before the training starts, the probability of all actions being selected in any state $s$ is equal, that is, $Q_0(s, a_1) = Q_0(s, a_2)$. Suppose that by the time the novel experience appears, the old experience $(s, a_1, r_1, s_1)$ has been learned $n$ times; then, $Q(s, a_1) - Q(s, a_2)$ can be estimated by (33) shown at the bottom of this page.

Thus, the difference in TD errors of the novel and the old experience is

$$\delta_2 - \delta_1 \approx r_2 - r_1 + Q(s, a_1) - Q(s, a_2)$$

$$= -(1 - \alpha)^n (r_1 - r_2). \quad (34)$$

Since $0 < \alpha < 1$, we can get

$$-(1 - \alpha)^n < 0,$$

and $\delta_2 < \delta_1$.

In summary, when the novel action is a better action, the novel experience is more worth learning than the old experience.

New experiences are generated under the guidance of the latest policy, and they are more likely to contain novel experiences. Combined experience replay [38] replays the newest experience every learning time to improve the performance of DQN with a large experience pool. PER [27] assigns the newest experiences with the greatest priority among all stored experiences. However, combined experience replay only replays the newest experience and PER cannot guarantee that the newest experiences can be learned at least once; neither of these two advanced ER algorithms can take full advantage of novel experiences.

To remedy this situation, this article constructed a small FIFO experience pool named the temporary pool to temporarily store the serial number of the newest experiences (Fig. 6). Every new experience will be stored in the experience pool and its serial number will be stored in the temporary pool. When sampling, the $N_{tp}$ experiences copied according to the serial numbers stored in the temporary pool and the $N_{ep}$ experiences sampled from the remaining experiences in the experience pool together form a minibatch of size $N$ for updating networks. This temporary pool ensures that the same experience will not be sampled multiple times during one sampling, thereby effectively avoiding overfitting and making full use of the newest experiences. A deep survey of the influence of different $N_{tp}$ on the algorithm will be presented in Section V.

4) Oldest is Not the Worst: The experience pool in most DRL algorithms adopts FIFO fashion because it is generally believed that the older the experience, the less learning value it contains. This may not be the case, because old experiences are not necessarily the worst.

Chen et al. [57] proposed a locality-sensitive experience replay (LSER) to replace experiences with smaller rewards when the experience pool is full. LSER proves that appropriately increasing the exploitation of agents without changing the exploration can effectively improve the performance of the algorithm. Inspired by LSER, this article believes that even if some of the older experiences do not have much effect on updating the current networks, they may still play a key role at some point in the future. Compared with some older experiences, new experiences with less value for the current network update are more necessary to be replaced. In AER, when a new experience

$$Q(s, a_1) - Q(s, a_2)$$

$$= Q_n(s, a_1) - Q_n(s, a_2)$$

$$\approx \left\{ Q_{n-1}(s, a_1) + \alpha \left[ r_1 + \gamma \max_{a'_1} Q'_{n-1}(s_1, a'_1) - Q_{n-1}(s, a_1) \right] \right\} - \left\{ Q_{n-1}(s, a_2) + \alpha \left[ r_2 + \gamma \max_{a'_2} Q'_{n-1}(s_2, a'_2) - Q_{n-1}(s, a_2) \right] \right\}$$

$$= (1 - \alpha) \left[ Q_{n-1}(s, a_1) - Q_{n-1}(s, a_2) \right] + \alpha (r_1 - r_2)$$

$$= (1 - \alpha) \left( (1 - \alpha) \left[ Q_{n-2}(s, a_1) - Q_{n-2}(s, a_2) \right] + \alpha (r_1 - r_2) \right) + \alpha (r_1 - r_2)$$

$$= (1 - \alpha)^2 \left[ Q_{n-2}(s, a_1) - Q_{n-2}(s, a_2) \right] + (\alpha + \alpha (1 - \alpha)) (r_1 - r_2)$$

$$= (1 - \alpha)^n [Q_0(s, a_1) - Q_0(s, a_2)] + [\alpha + \alpha(1 - \alpha) + \cdots + \alpha(1 - \alpha)^{n-1}] (r_1 - r_2)$$

$$= [1 - (1 - \alpha)^n] (r_1 - r_2)$$

\[33\]
needs to be stored, if the experience pool is full, the experience with smaller priority will be replaced.

We improved the ‘sum-tree’ of the PER algorithm to form a first-in-useless-out (FIUO) buffer named ‘double sum-tree’ to reduce the computational complexity of finding more useless experience. Due to the asynchronous TD error updating subthread, the priorities of all stored experiences are constantly being updated, so it can be ensured that the replaced experience is more useless in a short time.

As shown in Fig. 7, unlike ‘sum-tree’, which only stores the sampling factor $p^*_j$ of experiences $e_i$ in the leaf node, ‘double sum-tree’ also stores the replacing factor $\frac{1}{p^*_j}$. The parent node also has two data points $\sum p^*_j$ and $\sum \frac{1}{p^*_j}$, which are the sum of the sampling factors and replacing factors of its child nodes, respectively. The probability of replacing experience $e_i$ is

$$P_{rep}(e_i) = \frac{\frac{1}{p^*_i}}{\sum_{j=1}^{D} \frac{1}{p^*_j}}. \quad (36)$$

After using a specific method introduced in later sections to limit the size of priorities, this FIUO buffer can make each experience have a probability of being replaced, and the lower-priority experience has a greater probability of being replaced, thus achieving a trade-off between exploration and exploitation, and reducing the possibility of convergence to sub-optimal policy.

B. Curriculum Experience Replay

The first task of combining CL and ER is to define a reasonable standard to measure the difficulty of different experiences. For example, the experiences of UAVs can be simply divided into three categories according to difficulty (Fig. 8): 1) steady forward flight (simple experiences); 2) steady flight toward the target (medium experiences); and 3) steady flight toward the target and avoid obstacles (difficult experiences). However, UAV AMC in the real world is a complex problem, which means that artificially assigning the difficulties of experiences may be inappropriate and more professional classification standards should be applied. Motivated by the PER algorithm, which assigns different experiences with different priorities $p$ according to the TD errors, we find that the TD error $\delta$ is a good standard for dividing the difficulties of experiences. In deep neural networks, transitions with large magnitudes of TD errors require a smaller step size to follow the curvature of the objective function [58]. The larger the $\delta$ is, the greater the impact of the experience on the current network, the more the experience should be learned, and, in a sense, the greater the difficulty of fully grasping the policy of the experience.

A priority function for measuring difficulty should meet the following conditions: 1) It should be a bounded function in $[0, +\infty)$ to avoid outliers and ensure that every experience has the probability of being sampled. 2) It should increase monotonically first and then decrease monotonically to ensure that the experience corresponding to the curriculum factor can have the greatest priority. 3) The slope should be easily adjusted to ensure that it can be adjusted to suit different environments. The first condition can ensure the diversity of the sampled experiences, while the second and third conditions can ensure the sampling priority and the universality of the priority function, respectively. After a certain amount of exploration, the priority function of CER is depicted as follows:

$$p(\delta, c) = \begin{cases} 
\exp (k_1 \cdot (|\delta| - c)) & |\delta| \leq c \\
\exp (k_2 \cdot (c - |\delta|)) & |\delta| > c 
\end{cases}, \quad (37)$$

where $c$ is the increasing curriculum factor that indicates the learning stages. $k_1$ and $k_2$ are constants used to adjust the slope of the priority function. Fig. 9 gives a sketch of the priority function with different $c$. By using this well-designed priority function whose value range is $(0,1)$, the clip problem introduced by the true TD error can be solved. In the whole training process, the initial value of $c$ is $c_{init}$, which is updated at regular intervals by $c = c + c_{incr}$. Unlike the DCRL algorithm updates the curriculum factor at
each learning step, we believe that an update frequency that is too high causes the agent to be unable to learn steadily because the difficulty of the experiences to be learned is constantly changing. The CER algorithm updates \( c \) every \( U_c \) episodes to ensure that the agent has a consistent learning standard in every \( U_c \) episodes. From Fig. 9, the smaller the \( k_1 \) (or \( k_2 \)), the slower the trend of the priority function, and the greater the diversity of the sampled experiences. As mentioned before, compared with the experiences with smaller TD errors (\(|\delta| < c\)), the experiences with larger TD errors (\(|\delta| > c\)) are more valuable to learn \[27\], \[42\] and needs to be learned more times \[58\]. After exploration, we found that setting \( k_1 > k_2 \) to make the experiences with larger TD errors have higher sampling priorities can have a positive impact on the convergence of the network.

To verify the impact of CER, experiments are conducted on ‘CartPole-v0’, and the results are shown in Fig. 10. It can be clearly seen that the CER algorithm effectively changes the priority distribution of experiences in the experience pool. As the maximum priority of all experiences is still assigned to the newest experiences, there are still many experiences with priority 1 when the CER is applied alone (Fig. 10(a)). When the AER is added, the subthread quickly updates the priorities of all stored experiences. The experiences with priority 1 will be given their true priorities. In addition, following the curriculum factor \( c \), the priority distribution of experience in the experience pool becomes the distribution we want, as shown in Fig. 10(b). Under this distribution, the agent can better choose the experience that suits its current state to learn and achieve rapid and stable convergence of the policy.

C. Implementation

CER mainly plays a role in the asynchronous update of AER’s subthread. When AER updates the priorities of the stored experiences in order, CER gives more appropriate priorities to the sampled experiences according to the curriculum factor and their true TD errors, so as to ensure the rationality of the priorities of the stored experiences. In this way, AER can accelerate the formation of a curriculum priority distribution of experiences and the conversion between curriculum priority distributions of different difficulties to ensure that the agent can learn stably according to the difficulty of the experiences. CER can enable AER to give true priority to experiences to make full use of the advantages of importance sampling, thereby accelerating the speed of agent learning. Here, the ACER algorithm is finally formed and applied to the TD3 algorithm to show its performance. The pseudocode of the ACER-TD3 algorithm is presented as Algorithm 1.

D. Analysis of the Algorithm Complexity

The ACER algorithm uses a subthread to asynchronously update the TD errors of \( A \) experiences costs \( O(A) \) each timestep. The operation of assigning true priority of the new experience and storing it in the temporary pool both increase the time complexity of \( O(1) \). In addition, finding the experience to be replaced in the FIUO buffer takes \( O(\log D) \). In summary, the time complexity of ACER increased by \( O(A) \) compared to the PER algorithm at each timestep. Since the asynchronous update operation is completed by the subthread, the running time of the ACER algorithm will not increase significantly.

V. Experiments

A. Settings

To simulate the realistic UAV state as much as possible, a UAV model based on the parameters of the ‘Wing Loong II’ UAV (Fig. 11), an identify and destroy integrated UAV developed by the Chinese Chengdu Aircraft Design and Research Institute, is constructed. The maximum flight speed and the maximum load factor of the UAV are set to 103m/s and 15, respectively. For the TAR of the UAV, \( N_r \) is 32 to return the environmental state, and the detection distance is set to 5km.

For the simulation of the battlefield, a large-scale dynamic 3D environment ranging from \( 120 \times 90 \times 10 \)km\(^3\) is designed to ensure UAVs perform different missions. As shown in Fig. 12, the simulation environment mainly contains three modules: the

![Chinese ‘Wing Loong II’ UAV.](Fig. 11)

![The Simulation Environment.](Fig. 12)
Algorithm 1: ACER-TD3.

**Initialization:** experience pool $R$ with the capacity $D$, temporary pool $R_{tp}$ with the capacity $N_{tp}$, minibatch $N$, replay period $K$, actor update delay $d$, training episode $M$, the number of asynchronous TD error update experiences $A$, update period $U_c$, curriculum factor $c$ and exponents $\alpha$ and $\beta$.

**Initialize:** Q-values $Q(s, a | \theta_1)$, $Q(s, a | \theta_2)$, and actor $\mu(s | \theta^a)$ with weights $\theta_1$, $\theta_2$, and $\theta^a$.

**Initialize:** target nets $Q'_1$, $Q'_2$, and $\mu'$ with weights $\theta'_1$, $\theta'_2$, and $\theta'^a$.

**Initialize:** subthread’s networks by copying main thread’s.

**for** episodes $= 1$, $M$ **do**

Select action with exploration noise $a_t = \mu(s_t | \theta^a) + \epsilon, \epsilon \sim N(0, \sigma)$ and observe the reward $r_t$ and new state $s_{t+1}$.

Store the experience $(s_t, a_t, r_t, s_{t+1})$ in the temporary pool $R_{tp}$ and the experience pool $R$ with maximal priority $p_t = \max p_i$.

**if** $t \equiv 0 \mod K$ **then**

Sample $N_{ep}$ experiences from $R$ according to the sample probability $i \sim P(i) = p_t^i / \sum j p_j^i$ combined with the $N_{tp}$ temporary experiences of $R_{tp}$ to form the minibatch $(s_i, a_i, r_i, s_i)$ for $i = 1 \ldots N$.

Compute the importance-sampling weight:

$$\omega_i = (D \cdot P(i))^{-\beta} / \max_j \omega_j.$$  

Set:

$$y_i = r(s_i, a_i) + \gamma \min_{j = 1 \ldots 2} Q(s_i', a_i' | \theta^2) + \epsilon | Q(s_i', a_i' | \theta^1),$$  

$$\epsilon \sim \text{clip}(N(0, \sigma), -L, L).$$

Update critic by minimizing the loss:

$$L(\theta^Q) = \min_{j = 1 \ldots 2} \frac{1}{N} \sum_i \omega_i \left( y_i - Q_j(s_i, a_i | \theta^Q) \right)^2.$$  

**if** $t \equiv 0 \mod d$ **then**

Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^{a}} \mathbb{E} = \frac{1}{N} \sum_i \nabla_{\theta^{a}} Q(s_i, a_i | \theta^Q) |_{a_i = \mu(s_i, \cdot | \theta^a)} \nabla_{\theta^{a}} \mu(s_i, \cdot | \theta^a).$$  

**end if**

Update the target networks: $\theta'_i \leftarrow \theta_i + (1 - \tau) \theta_i'$.

Copy network parameters from main thread to subthread.

**end if**

Subthread samples $A$ experiences from $R$. Compute the TD error:

$$\delta_i = y_i - Q_j(s_i, a_i | \theta^Q).$$

Update the experience priority:

$$p_t \leftarrow p(\delta_i, c) = \begin{cases} \exp(k_1 \cdot (|\delta_i| - c)) & |\delta_i| \leq c \\ \exp(k_2 \cdot (c - |\delta_i|)) & |\delta_i| > c. \end{cases}$$

**end for**

Update curriculum factor $c = c + c_{\text{incr}}$.

**end if**

**end for**

UAV, the target, and the obstacles. The target, represented by the green hemisphere, is set with a fixed radius of 3km. Since the airborne electronic support measures (ESM) can effectively sense whether the UAV enters the enemy monitoring area, the air-defense weapon threats can be abstracted into the same area as the terrain threats, which are uniformly represented by the white hemispheres whose radii range from 5-10km. The initial positions of the target and UAV are randomly generated, and the distance between them is greater than 50km to increase the exploration of every episode. The direction of movement of each target is also random. To more easily calculate the state information of the UAV, the acceleration of gravity is a fixed value of 9.8m/s². In addition, after much exploration and experimentation, the reward function is set as follows: the rewards $r_t$ and $r_f$ are 100 and -200, respectively; the contribution rate factors are $\lambda_1 = 20$, $\lambda_2 = 20$, $\lambda_3 = 10$, $\lambda_4 = 40$, and $\lambda_5 = 10$. More information about the simulation environment can be found in our previous work [12].

In addition to the proposed ACER algorithm, two baseline algorithms of TD3 [32] and PER [27] and two novel ER algorithms REL [12] and DCRL [30] are trained for comparison. For the hyperparameters of the TD3 algorithm, both the actor network and the two critic networks contain two hidden layers with 100 nodes each. The Adam optimizer is employed to update the network parameters with learning rates of 0.0001 and 0.001 for the actor and critic, respectively. The discount factor $\gamma$ is 0.9. The soft update rates of the actor and critic are set to 0.1 and 0.2, respectively. Both the action exploration noise and the target policy smoothing noise satisfy a Gaussian distribution $N(0, 0.1)$. The actor update delay $d$ is 2. For the ER of the training process, the capacity $D$ of the experience pool is 50,000. The size of the minibatch $N$ is 256. The replay period $K$ is 20. The other hyperparameters of the ER part of the PER, REL, and DCRL algorithms are the same as those in [12], [27], and [30]. For the proposed ACER algorithms, the exponent $c$ is 0.6 and the importance sample variable $\beta$ is linearly annealed from 0.4 to 1. The number of asynchronous TD error updating experiences $A$ is 256. The capacity of the temporary experience pool $N_{tp}$ is 5. The initial value of curriculum factor $c$ is 10, and the increment of curriculum factor $c_{\text{incr}}$ is 1. The curriculum factor update period $U_c$ is 100. The $k_1$ and $k_2$ of the priority function are 0.001 and 0.005, respectively. In addition, every agent is warmed up by an initial 200 episodes without training. The total number of episodes for each training is 5,000, and the maximum timestep of each episode is 3,000. The simulation timestep is 0.01s. Each training takes about 10 hours. All the experiments are carried out on a computer with an Nvidia RTX 2080Ti GPU, Ubuntu 16.04 LTS, and Python.

B. Experimental Results

To validate the efficiency of the proposed ACER algorithm, an environment containing 20 random obstacles with a speed of 5m/s is set to train the above five algorithms. The hit rate, the probability of the UAV successfully hitting the target in the last 500 episodes, is defined as a main evaluation indicator. In addition, the following indicators are defined from different perspectives to further compare the pros and cons of the algorithms: 1) Training Peak (TP): The peak hit rate of the trained agent during the entire 5,000 episodes of training. 2) Convergence Time (CT): The number of episodes where the hit rate first reached 70%. 3) Stability after Convergence (SC): The standard deviation of the hit rate in the last 1,500 episodes.
Convergence Result (CR): The average hit rate in the last 1,500 episodes. Each algorithm is trained 10 times under different random seeds to obtain the average value to avoid the influence of random numbers. The complete experimental results are shown in Fig. 13 and Table I below:

From the experimental results, we can see that the convergence speed of vanilla TD3 is the slowest due to the uniform experience replay. Other algorithms with different ER mechanisms all have higher convergence speeds. This also further shows the great influence of ER in the DRL algorithms. The proposed ACER algorithm only needs 2,600 episodes to converge, which is an improvement of 24.66% compared with the TD3 algorithm (which needs 3,451 episodes). However, compared with two of the state-of-the-art ER algorithms, REL and DCRL, the ACER algorithm can still improve by 15.23% and 13.94%, respectively.

For the convergence results, the hit rate of the ACER algorithm is the highest (79.52%) among all algorithms. In addition, it is worth mentioning that the hit rate of ACER is always the highest throughout the convergence process. Compared with the TD3 algorithm (75.31%), the convergence result of ACER is improved by 5.59%. Compared with other advanced ER algorithms, the hit rate of ACER still shows some improvement. An interesting phenomenon is that the convergence speed of DCRL is second only to ACER among all algorithms, but its convergence result is indeed the worst. There are two reasons why DCRL has a fast convergence rate but poor convergence results: 1) The curriculum factor that changes too quickly prevents the agent from fully learning the experience of various difficulties. 2) The adoption of a coverage penalty introduces errors to the priority.

The stability after convergence is another important indicator to evaluate the performance of the algorithms. The smaller the SC is, the better the initial convergence of the algorithm, which means that the update of the network parameters in the later training stage will not have a greater impact on the performance of the algorithm. Among all algorithms, ACER (1.14) has the smallest SC, followed by PER (1.17). The SCs of REL, DCRL, and TD3 are larger, and the values are 1.68, 1.71, and 1.73, respectively. This means that compared to other algorithms, the ACER algorithm is more likely to converge to the optimal solution directly.

C. Testing in Different Environments

There are two main factors that affect the complexity of the environment: the velocity of obstacles and the number of obstacles. The greater the velocity and the number of obstacles are, the greater the requirement placed on the accuracy or effectiveness of UAV AMC will be. In this section, environments of different levels of complexities are set to explore the generalization capabilities of agents trained by different algorithms. In addition, by drawing the UAV motion trajectories, the performance of the agents in different environments is analyzed and discussed.

1) Environments With Different Velocities of Obstacles: Experiments are conducted in a complex environment with 20 obstacles, and the experimental results are shown in Fig. 14. Obviously, it can be seen that as the speed of obstacles continues to increase, the hit rates of the agents generally show a downward trend. Among these agents, ACER’s hit rate dropped the slowest, while DCRL and TD3’s hit rates dropped the fastest. When the obstacles’ movement velocity reaches 10m/s, the hit rate...
of different agents is significantly different: From high to low, they are ACER (78.20%), REL (74.10%), PER (69.50%), DCRL (63.70%), and TD3 (60.70%). When the obstacle’s movement velocity reaches 15m/s, the hit rate gap between the agents is more obvious. ACER still has a hit rate of 70.60%, while TD3 and DCRL drop to 52.30% and 46.90%, respectively.

Fig. 15 shows the motion trajectories of all well-trained agents in three environments with different velocities of obstacles. The proposed ACER agent exhibits better generalization ability because it accomplishes the task excellently in all three environments (Fig. 15(c), (j), and (p)). Another agent trained by the advanced REL algorithm performs well in ENV 2 and ENV 3 but takes more timesteps 2,183 (Fig. 15(l)) and 2,347 (Fig. 15(r)), respectively. In test environment ENV 1, two agents (TD3 and PER) also complete the task, which costs 1,494 (Fig. 15(e)) and 2,082 (Fig. 15(f)) timesteps, respectively. However, the DCRL agent is the only other agent that reaches the target point in ENV 2, taking 1,945 timesteps.

2) Environments With Different Numbers of Obstacles: The increase in the number of obstacles will lead to a decrease in the density of safe areas in the environment, which will greatly increase the difficulty for the UAV to complete the task. To verify the generalization ability of the algorithms in environments with different obstacle densities, experiments are conducted in an environment in which the velocity of obstacles is 10m/s. Experimental results (Fig. 16) demonstrate that an increase in the number of obstacles will gradually reduce the hit rates of the agents. As the number of obstacles increased from 10 to 30, the DCRL agent’s hit rate dropped the most, from 76.20% to 44.20%. The TD3 agent’s hit rate dropped from 75.50% to 50.80%. Compared with these agents, the ACER agent only has a drop of 17.30% (from 84.60% to 67.30%), which demonstrates excellent generalization ability.

Three environments with different numbers of obstacles (Fig. 17(a), (g), and (m)) are set up to show the motion trajectories of the agents trained by all algorithms. In ENV 4 (lowest obstacle density), the ACER, REL, and TD3 agents spend 1,580 (Fig. 17(d)), 1,676 (Fig. 17(e)), and 1,780 (Fig. 17(f)) timesteps flying to the target point and finally succeeding. Only one well-trained agent (ACER) completes the task in ENV 5 (Fig. 17(l)). Except for the ACER agent, which only takes 1,840 timesteps (Fig. 17(q)) to reach the target in the most complex ENV 6, the DCRL agent also completes the task using 1,905 timesteps (Fig. 17(r)). However, the agent trained by the PER algorithm failed in all three environments.

The superiority of the proposed ACER over other algorithms can be demonstrated by the above testing experiments in different environments. The results show the following: 1) Stronger generalization ability: As the complexity of the environment increases, ACER can always maintain the highest hit rate. In addition, the ACER agent is the only agent that successfully completes tasks in six different environments. 2) Higher security: The longer the UAV stays in the enemy area, the more likely it is to be discovered. Compared to other agents, the ACER agent can always take shorter timesteps to complete the task, which ensures that it has higher security. 3) More decisive decision-making ability: An interesting phenomenon that can be easily seen is that TD3 (Fig. 17(f)), PER (Fig. 15(f)), DCRL (Fig. 17(r)), and REL (Fig. 15(l) and (r)) agents sometimes hesitate near the target. This situation usually occurs when the agent detects both the target and the obstacles. The agent wants to approach the target to obtain a larger reward but is afraid of collision with obstacles around the target. This is due to the insufficient learning of the reward function in the training process. The agent is too entangled in the reward of the next few timesteps and does not know that if it flies decisively to the target, it can obtain the reward of success. In a real battlefield environment, UAVs need to make decisive decisions, because once the opportunity is not seized, the UAV may have to wait a long time for the next opportunity. ENV 3 is a good example: Although the REL and TD3 agents also reached the target point early, the target was covered by moving obstacles due to their hesitation. TD3 hovered and finally collided with an obstacle (Fig. 15(q)), while REL took 738 timesteps to wait for the next opportunity (Fig. 15(r)). For UAVs, more decisive decision-making capabilities can not only ensure their own safety, but also increase the success rate of missions. In summary, the excellent performance of the ACER agent shows that the ACER algorithm is a more valuable DRL algorithm when training real UAVs.

D. Additional Exploratory Experiments of Hyperparameters

The novel ACER algorithm adds some new hyperparameters to the vanilla TD3, and it is necessary to study the influence of the value of these hyperparameters on the performance of the algorithm. In this section, each group of experiments was performed 5 times under different random seeds to calculate the average. We first conducted some groups of experiments in the same training environment with different values of $N_{tp}$, and the experimental results are shown in Table II.

It can be clearly seen that the introduction of the temporary pool can effectively accelerate the convergence of the TD3 algorithm. Different $N_{tp}$ has different effects on the results of TD3. If $N_{tp}$ is too small (group No. 2), the new experience cannot be fully utilized. If $N_{tp}$ is too large (group No. 5), too many continuous experiences will be learned and affect the convergence result.
Fig. 15. Motion trajectories in environments with different velocities of obstacles. (a) ENV 1. start: (330, 550, 10), end: (−410, −280, 0), velocity of obstacles: 5 m/s. (b) Timestep = 948 (REL failed). (c) Timestep = 1,352 (ACER succeeded). (d) Timestep = 1,384 (DCRL failed). (e) Timestep = 1,494 (TD3 succeeded). (f) Timestep = 2,082 (PER succeeded). (g) ENV 2. start: (−100, 530, 10), end: (50, −560, 0), velocity of obstacles: 10 m/s. (h) Timestep = 916 (TD3 failed). (i) Timestep = 1,114 (PER failed). (j) Timestep = 1,787 (ACER succeeded). (k) Timestep = 1,945 (DCRL succeeded). (l) Timestep = 2,183 (REL succeeded). (m) ENV 3. start: (−180, 510, 10), end: (120, −460, 0), velocity of obstacles: 15 m/s. (n) Timestep = 910 (DCRL failed). (o) Timestep = 1,003 (PER failed). (p) Timestep = 1,609 (ACER succeeded). (q) Timestep = 1,828 (TD3 failed). (r) Timestep = 2,347 (REL succeeded).

Fig. 16. The hit rates of all algorithms with different numbers of obstacles. The number of obstacles increases gradually from 10 to 30. Each piece of data is calculated by the UAV agent trained by the corresponding algorithm tested in the environment for 1,000 episodes.

TABLE II

| No. | Ntp | TP↑ | CT↓ | SC↓ | CR↑ |
|-----|-----|-----|-----|-----|-----|
| 1   | 0   | 78.80% | 3,451 | 1.73 | 73.31% |
| 2   | 1   | 78.80% | 3,380 | 1.98 | 76.42% |
| 3   | 5   | 79.36% | 3,152 | 1.72 | 77.05% |
| 4   | 10  | 81.68% | 3,335 | 2.04 | 77.79% |
| 5   | 20  | 76.72% | 3,462 | 1.90 | 74.31% |

The arrows are attached to point to the better performance and the best results are marked in bold.

TABLE III

| No. | A | TP↑ | CT↓ | SC↓ | CR↑ |
|-----|---|-----|-----|-----|-----|
| 1   | 0 | 76.72% | 3,228 | 2.12 | 73.14% |
| 2   | 64 | 77.68% | 3,038 | 1.04 | 75.13% |
| 3   | 128 | 77.60% | 2,951 | 1.50 | 75.03% |
| 4   | 256 | 81.92% | 2,600 | 1.13 | 79.52% |
| 5   | 512 | 82.32% | 2,507 | 1.95 | 79.33% |

The arrows are attached to point to the better performance and the best results are marked in bold.

Experiments have found that the values of Ntp from 5 to 10 can produce better results for the algorithm. The number of experiences whose priorities are asynchronously updated by the subthread should also be studied. From Table III, as A increases, the convergence speed and convergence result of the ACER algorithm gradually increase. As A increases past 256, the performance of the algorithm is no longer significantly improved.

The introduction of CL also adds some hyperparameters, such as the initial value of the curriculum factor cinit, the increment of the curriculum factor cincr, and constants k1, and k2. From the experimental results shown in Table IV, increasing or decreasing the value of cinit will have an adverse effect on the performance of the agent, which indicates that the optimal value of cinit is approximately 10.0. Experiments also show that appropriately reducing cincr can achieve better results (group No. 6). In addition, as the value of k1/k2 gradually increases, the performance of the agent worsens.

TABLE IV

| No. | cinit | cincr | k1 | k2 | TP↑ | CT↓ | SC↓ | CR↑ |
|-----|-------|-------|----|----|-----|-----|-----|-----|
| 1   | 0.01  | 0.005 | 0.005 | 81.92% | 2,600 | 1.14 | 79.52% |
| 2   | 1.0   | 0.003 | 0.002 | 79.26% | 2,877 | 1.23 | 77.71% |
| 3   | 10.0  | 0.005 | 0.002 | 77.43% | 3,133 | 1.15 | 76.82% |
| 4   | 5.0   | 0.01  | 0.005 | 78.67% | 3,094 | 1.31 | 77.33% |
| 5   | 15.0  | 0.01  | 0.005 | 76.13% | 3,242 | 1.62 | 75.24% |
| 6   | 10.0  | 1.0   | 0.005 | 82.36% | 2,713 | 1.09 | 79.66% |
| 7   | 10.0  | 1.5   | 0.005 | 72.92% | 3,285 | 2.04 | 69.43% |

Group No. 1 is the control group. The arrows are attached to point to the better performance and the best results are marked in bold.
VI. CONCLUSION

In this work, we design a DRL framework for controlling UAVs in complex unknown dynamic environments. The UAV AMC problem is formulated as an MDP and a novel DRL algorithm, ACER, is proposed to address it. In ACER, the priorities update of off-line experiences is accelerated by multithreads, and the true priorities are assigned without clip. A small temporary pool is designed for storing new experiences, and an FIU experience pool is designed to ensure that more effective experiences can be learned. Moreover, by integrating CL, ACER changes the random training process of DRL into a smoother training process that proceeds from simple to difficult while ensuring the stability. The experimental results demonstrate the success of ACER in comparison to some state-of-the-art DRL algorithms. In addition, the superiority of ACER has also been presented by generalizing the well-trained agent for different large-scale dynamic 3D environments.

Although ACER exhibits superior performance in addressing AMC problem, it still has some shortcomings that limit its practical application. For example, the combination with CL introduces several parameters that need to be carefully tuned. In the future, we will conduct more in-depth theoretical research on CL to build a more concise and universal DRL framework. In addition, introducing CL into multiagent DRL to achieve more efficient UAV cluster control will also be a focus of our research.

ACKNOWLEDGMENT

The authors thank Letian Zhang for the literature collection and linguistic assistance.

REFERENCES

[1] Y. Liu, Z. Luo, Z. Liu, J. Shi, and G. Cheng, “Cooperative routing problem for ground vehicle and unmanned aerial vehicle: The application on intelligence, surveillance, and reconnaissance missions,” *IEEE Access*, vol. 7, pp. 63504–63518, 2019.

[2] D. Shen, G. Chen, J. B. Cruz, and E. Blasch, “A game theoretic data fusion aided path planning approach for cooperative UAV ISR,” in *Proc. IEEE Aerosp. Conf.*, 2008, pp. 1–9.

[3] Y. Duan, X. Ji, M. Li, and Y. Li, “Route planning method design for UAV under radar ECM scenario,” in *Proc. IEEE 12th Int. Conf. Signal. Process.*, 2014, pp. 108–114.

[4] K. Hartmann and K. Giles, “UAV exploitation: A new domain for cyber power,” in *Proc. IEEE 8th Int. Conf. Cyber Conflict.*, 2016, pp. 205–221.

[5] M. Suresh and D. Ghose, “UAV grouping and coordination tactics for ground attack missions,” *IEEE Trans. Aerosp. Electron. Syst.*, vol. 48, no. 1, pp. 673–692, Jan. 2012.

[6] Y. Hou, X. Liang, L. He, and J. Zhang, “Time-coordinated control for unmanned aerial vehicle swarm cooperative attack on ground-moving target,” *IEEE Access*, vol. 7, pp. 106931–106940, 2019.

[7] C. Wang, J. Wang, Y. Shen, and X. Zhan, “Autonomous navigation of UAVs in large-scale complex environments: A deep reinforcement learning approach,” *IEEE Trans. Veh. Technol.*, vol. 68, no. 3, pp. 2124–2136, Mar. 2019.

[8] X. Yang, M.-Y. Ding, and C.-P. Zhou, “Fast marine route planning for UAV using improved sparse A* algorithm,” in *Proc. IEEE Int. Conf. Genet. Evol. Comput.*, 2010, pp. 190–193.

[9] R. Tianzhu, Z. Rui, X. Jie, and D. Zhuoning, “Three-dimensional path planning of UAV based on an improved A* algorithm,” in *Proc. IEEE Chin. Guid., Navigation Control Conf.*, 2017, pp. 140–145.

[10] T. Shimohara and T. Namenikawa, “SLAM for a small UAV with compensation for unordinary observations and convergence analysis,” in *Proc. IEEE 55th Annu. Conf. Soc. Instrum. Control Eng.*, 2016, pp. 1252–1257.

[11] A. Annaiyan, M. A. Olivares-Mendez, and H. Voos, “Real-time graph-based SLAM in unknown environments using a small UAV,” in *Proc. IEEE Int. Conf. Unmanned Aircr. Syst.*, 2017, pp. 1118–1123.

[12] Z. Hu, K. Wen, X. Gao, Y. Zhai, and Q. Wang, “Reliable experience learning: A deep reinforcement learning method for UAV autonomous motion planning in complex unknown environments,” *Chin. J. Aeronaut.*, vol. 34, no. 12, pp. 187–204, 2021.

[13] R. Li, L. Fu, L. Wang, and X. Hu, “Improved Q-learning based route planning method for UAVs in unknown environment,” in *Proc. IEEE Int. Conf. Control Autom.*, 2019, pp. 118–123.

[14] S.-M. Hung and S.-X. Givigi, “A Q-learning approach to flocking with UAVs in a stochastic environment,” *IEEE Trans. Cybern.*, vol. 47, no. 1, pp. 186–197, Jan. 2017.

[15] A. Asheralieva and D. Niyato, “Hierarchical game-theoretic and reinforcement learning framework for computational offloading in UAV-enabled mobile edge computing networks with multiple service providers,” *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8753–8769, Oct. 2019.

[16] J. C. H. Christopher and P. Dayan, “Q-learning,” *Mach. Learn.*, vol. 8, pp. 279–292, 1992.
[17] A. Asheralieva and D. Niyato, “Distributed dynamic resource management and pricing in the IoT systems with blockchain-as-a-service and UAV-enabled mobile edge computing,” IEEE Internet Things J., vol. 7, no. 3, pp. 1974–1993, Mar. 2020.

[18] V. Miculicic, “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, pp. 529–533, 2015.

[19] H. van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double Q-learning,” in Proc. AAAI Conf. Artif. Intell., 2016, pp. 2094–2100.

[20] Q. Liu, L. Shi, L. Sun, J. Li, M. Ding, and F. Shu, “Path planning for UAV-mounted mobile edge computing with deep reinforcement learning,” IEEE Trans. Veh. Technol., vol. 69, no. 5, pp. 5722–5728, May 2020.

[21] X. Chen et al., “Information freshness-aware task offloading in air-ground integrated edge computing systems,” IEEE J. Sel. Areas Commun., vol. 40, no. 1, pp. 243–258, Jan. 2022.

[22] T. P. Lillicrap et al., “Continuous control with deep reinforcement learning,” in Proc. Int. Conf. Mach. Learn., 2016. [Online]. Available: https://arxiv.org/abs/1509.02971

[23] R. Ding, F. Gao, and X. S. Shen, “3D UAV trajectory design and frequency band allocation for energy-efficient and fair communication: A deep reinforcement learning approach,” IEEE Trans. Wireless Commun., vol. 19, no. 12, pp. 7796–7809, Dec. 2020.

[24] B. Li and Y. Wu, “Path planning for UAV ground target tracking via deep reinforcement learning,” IEEE Access, vol. 8, pp. 29064–29074, 2020.

[25] Z. Hu, K. Wan, X. Gao, Y. Zhai, and Q. Wang, “Deep reinforcement learning approach with multiple experience pools for UAV’s autonomous motion planning in complex unknown environments,” Sensors, vol. 20, no. 7, 2020, Art. no. 1890.

[26] S. Yeo, S. Oh, and M. Lee, “Accelerating deep reinforcement learning using human demonstration data based on dual replay buffer management and online frame skipping,” in Proc. IEEE Int. Conf. Big Data Smart Comput., 2019, pp. 1–8.

[27] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” in Proc. Int. Conf. Learn. Representation, 2016. [Online]. Available: https://arxiv.org/abs/1511.05952

[28] L. Wang, K. Wang, C. Pan, W. Xu, N. Aslam, and A. Nallanathan, “Deep reinforcement learning based dynamic trajectory control for UAV-assisted mobile edge computing,” IEEE Trans. Mobile Comput., vol. 21, no. 10, pp. 3536–3550, Oct. 2022.

[29] Z. Hu, X. Gao, K. Wan, N. Evgeny, and J. Li, “Imaginary filtered hindsight experience replay for UAV tracking dynamic targets in large-scale unknown environments,” Chin. J. Aeronaut., vol. 36, no. 5, pp. 377–391, May 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S100903612200020X

[30] Z. Ren, D. Dong, H. Li, and C. Chen, “Self-paced prioritized curriculum learning with coverage penalty in deep reinforcement learning,” IEEE Trans. Neural Netw. Learn. Syst., vol. 29, no. 6, pp. 2216–2226, Jun. 2018.

[31] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, “Curriculum learning,” in Proc. Int. Conf. Mach. Learn., 2009, pp. 41–48.

[32] S. Fujimoto, H. Van Hoof, and D. Meger, “Addressing function approximation error in actor-critic methods,” in Proc. Int. Conf. Mach. Learn., 2018, pp. 1582–1591.

[33] V. R. Konda and J. N. Tsitsiklis, “Actor-critic algorithms,” in Proc. Adv. Neural Inf. Process. Syst., 1999, pp. 1008–1014.

[34] R. S. Sutton and A. G. Barto, Reinforcement Learning — An Introduction, Cambridge, MA, USA: MIT Press, 1998.

[35] R. Bellman, “A Markovian decision process,” Indiana Univ. Math. J., vol. 6, no. 4, 1957, Art. no. 15.

[36] L. J. Lin, “Self-improving reactive agents based on reinforcement learning, planning and teaching,” Mach. Learn., vol. 8, pp. 293–321, 1992.

[37] T. D. Brunin, J. Koher, K. Tyss, and R. Babiskin, “The importance of experience replay database composition in deep reinforcement learning,” in Proc. Deep Reinforcement Learn. Workshop, 2015. [Online]. Available: https://tril.berkeley.edu/deeprlworkshop/

[38] S. Zhang and R. S. Sutton, “A deeper look at experience replay,” 2017, arXiv:1712.01275.

[39] W. Fedus et al., “Revisiting fundamentals of experience replay,” in Proc. Int. Conf. Mach. Learn., 2020, pp. 3061–3071.

[40] Q. Wei, H. Ma, C. Chen, and D. Dong, “Deep reinforcement learning with quantum-inspired experience replay,” IEEE Trans. Cybern., vol. 52, no. 9, pp. 9326–9358, Sep. 2022.

[41] S. Lee, J. Lee, and I. Hasuo, “Predictive PER: Balancing priority and diversity towards stable deep reinforcement learning,” in Proc. IEEE Int. Joint Conf. Neural Netw., 2021, pp. 1–10.
Kaifang Wan was born in 1987. He received the B.E. degree in detection guidance and control technology from Northwestern Polytechnical University (NWPU), Xi’an, China, in 2010 and the Ph.D. degree in systems engineering in 2016. He is currently an Assistant Researcher with the School of Electronics and Information, NWPU. His current research interests include multiagent theory, approximate dynamic programming, and reinforcement learning. He was awarded with an admission from B.E. to Ph.D. directly in 2010.

Qianglong Wang received the B.E. degree in systems engineering from Northwestern Polytechnical University (NWPU), Xi’an, China, in 2017. He is currently working toward the Ph.D. degree in control science and engineering with the School of Electronics and Information, NWPU. His research interests include deep learning, computer vision, and sensitivity analysis.

Yiwei Zhai was born in 1997. She received the M.E. degree in systems engineering from the School of Electronics and Information, Northwestern Polytechnical University, Xi’an, China, in 2021. Her research interests include path planning, reinforcement learning, and multiagent systems.