JMLR: Workshop and Conference Proceedings 101:64–79, 2019 ACML 2019

A Continuous Actor-Critic Reinforcement Learning Approach to Flocking with Fixed-Wing UAVs

Chang Wang WANGCHANG07@NUDT.EDU.CN
Chao Yan YANCHAO17@NUDT.EDU.CN
Xiaojia Xiang * XIANGXIAOJIA@NUDT.EDU.CN
Han Zhou ZHOUHAN@NUDT.EDU.CN
National University of Defense Technology, Changsha, China

Editors: Wee Sun Lee and Taiji Suzuki

Abstract

Controlling a squad of fixed-wing UAVs is challenging due to the kinematics complexity and the environmental dynamics. In this paper, we develop a novel actor-critic reinforcement learning approach to solve the leader-follower flocking problem in continuous state and action spaces. Specifically, we propose a CACER algorithm that uses multilayer perceptron to represent both the actor and the critic, which has a deeper structure and provides a better function approximator than the original continuous actor-critic learning automation (CACLA) algorithm. Besides, we propose a double prioritized experience replay (DPER) mechanism to further improve the training efficiency. Specifically, the state transition samples are saved into two different experience replay buffers for updating the actor and the critic separately, based on the calculation of sample priority using the temporal difference errors. We have not only compared CACER with CACLA and a benchmark deep reinforcement learning algorithm DDPG in numerical simulation, but also demonstrated the performance of CACER in semi-physical simulation by transferring the learned policy in the numerical simulation without parameter tuning.

Keywords: unmanned aerial vehicle (UAV), flocking, reinforcement learning, actor-critic, experience replay

1. Introduction

UAVs (unmanned aerial vehicles) have been widely used for survivor search, coaster-border patrol, and anti-terrorist operations. These tasks typically require the collaboration of a squad of UAVs supervised by human operators. However, controlling the UAV squad remains a challenge because it requires tremendous human efforts to monitor each UAV's state which introduces heavy workload. In this paper, we consider the leader-follower problem that only the leader is controlled manually while the followers flock with the leader autonomously.

In the literature, the consensus theory (Sahu and Subudhi (2018); Wang et al. (2014)) has been used to solve the UAV flocking problem. However, it requires a precise kinematics model which is complex, time-varying and non-linear, therefore is hard to obtain in real-world environments. As an alternative approach, reinforcement learning (RL) methods have

* Corresponding author

attracted attention for developing the flocking behavior. In particular, a Q-learning based framework was developed for particle-based agents to learn how to flock in a self-organized way to avoid predators (Koichiro et al. (2006)). Similarly, a low-level flocking controller was integrated with a high-level RL module for multiple robots to learn how to avoid predators collaboratively (La et al. (2015)).

These methods have been proved effective for ground robots or quad-rotor UAVs in simulation, but flocking with fix-wing UAVs in real-world environments is much more difficult. The reason is that the policy learned by the RL agent is difficult to converge due to environmental dynamics such as airspeed and side wind. Not much work has been done about fixed-wing UAVs flocking in dynamic environments. The most related work (Hung et al. (2015)) proposed the Dyna-Q(λ) algorithm to learn a control policy for the leader-follower problem. The Dyna RL method learned an environment model and used the model to plan actions that sped up the learning process. Then, the authors proposed a Q(λ) algorithm with adaptive learning rates (Hung and Givigi (2017)). The algorithm was proved effective for fixed-wing UAVs flocking in numerical simulation. However, the authors simplified the problem by discretizing the state and action spaces. This is not appropriate for controlling UAVs in real-world environments.

In order to solve high-dimensional and continuous RL problems, deep neural networks (DNNs) have been used as function approximators to generalize over the state and action spaces. The experience replay technique (Lin (1992)) has been proved effective for the training of deep Q-network (DQN) (Volodymyr et al. (2015)) and deep deterministic policy gradient (DDPG) (Lillicrap et al. (2015)) by reusing the state transition samples. Instead of sampling from the experience replay buffer uniformly, prioritized experience replay (PER) (Schaul et al. (2015)) calculates the priority of samples to further improve the training efficiency. For example, Hou et al. (2017) proposed a new DDPG method with PER. Alternatively, a multi-critic DDPG method used the technique of double experience replay (Wu et al. (2018)). However, there is no guarantee that the performance of an experience replay mechanism is better than others for every RL algorithm. In other words, it is necessary to optimize the experience replay mechanism for a chosen RL algorithm.

In this paper, we solve the fixed-wing UAVs flocking problem within the RL framework in continuous spaces. Specifically, we choose continuous actor-critic learning automation (CACLA) (Van Hasselt and Wiering (2007, 2009)) as the baseline RL algorithm due to its ease of implementation and good performance for continuous RL problems. We replace the single-layer actor and critic networks with multilayer networks, and we propose a new experience replay mechanism accordingly. The main contributions are as follows:

- We have proposed a novel RL algorithm for fixed- wing UAVs flocking in continuous spaces.
- We have designed a double prioritized experience replay mechanism to speed up the learning.
- We have designed a semi-physical system to test the proposed algorithm towards real-world flocking.

The paper is organized as follows. Section 2 formulates the flocking problem. Section 3 describes the proposed CACER algorithm, followed by experiments and results in Section 4. Finally, Section 5 concludes the paper and outlines our plans for future work.

2. Problem Statement

In this paper, we deal with the challenge of flocking with fixed-wing UAVs through reinforcement learning (RL). In accordance with Hung and Givigi (2017), we also assume that there is only one leader and several followers that they fly at a constant average speed and fixed different heights to simplify the collision problem. The leader has its own task-specific control policy such as searching a given area or tracking a ground vehicle. It shares its state information with the followers by broadcasting through a wireless communication channel, including its position and pose. A follower has to control its roll angel in order to keep a certain distance from the leader ($d_1 < \rho < d_2$) in the top view (see Figure 1). As the followers are collision-free, they are allowed to use the same control policy so that the aggregate behavior emulates flocking in a leader-follower topology. Each follower maneuvers by selecting a roll angle setpoint regulated by an autopilot using a PID controller, and the roll angle setpoint is updated once every second.

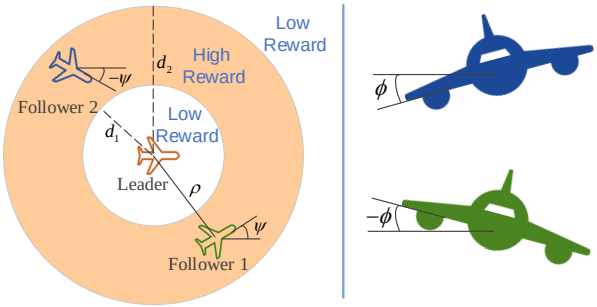

Figure 1: **Left**: Top view of the leader-follower topology. **Right**: Front view of the UAVs.

Without any prior knowledge of the leader's state transition model or control policy, a follower has to learn how to flock with the leader given the current state. In other words, the follower RL agent learns a strategy that describes the best roll angle setpoint for a given state, maximizing the expected accumulated rewards. Different with the previous work (Hung and Givigi (2017)) that discretizes the state and action spaces, we solve a more challenging problem in continuous state and action spaces towards developing intelligent agents that can learn to flock in the physical world.

2.1. Fixed-wing UAV Kinematics Model

In this paper, we assume that the state transition model is not available for RL agents, so we take the model-free RL approach. However, such RL methods typically require a long time of trial-and-error training, which is not feasible in the physical world or simulation

environment. Therefore, we simplify the UAV kinematics model and use simulated experience data to improve the policy learning efficiency. Then, we test the learned policy in a semi-physical environment that uses a real autopilot to control UAVs.

In the physical world, UAV kinematics are usually described by a six degree of freedom (DoF) aircraft model. Based on the assumptions of constant speed and fixed altitudes, we can simplify the model to 4 DoF. In order to compensate for the loss of unmolded dynamics, stochastic environmental disturbances are introduced in the roll, airspeed, and each of the substates of the model as follows:

$$\dot{\xi} = \frac{d}{dt}\begin{pmatrix} x \\ y \\ \psi \\ \phi \end{pmatrix} = \begin{pmatrix} s\cos\psi + \eta_x \\ s\sin\psi + \eta_y \\ -\left(\alpha_g/s\right)\tan\phi + \eta_\psi \\ f(\phi, r) \end{pmatrix} \tag{1}$$

where (x, y) is the planar position of the UAV, ψ is the heading, ϕ is the roll angle, α_g is the acceleration due to gravity, s is the airspeed of the UAV drawn from a normal distribution $\mathbb{N}\left(\overline{s}, \sigma_s^2\right)$, and the disturbance terms $(\eta_x, \eta_y, \eta_\psi)$ are drawn from the normal distributions $\mathbb{N}\left(\overline{\eta}_x, \sigma_x^2\right), \mathbb{N}\left(\overline{\eta}_y, \sigma_y^2\right)$ and $\mathbb{N}\left(\overline{\eta}_\psi, \sigma_\psi^2\right)$, respectively. The function $f(\phi, r)$ defines the relations between the desired roll angel r and the response roll angle ϕ.

We use a second-order system to simulate the initial condition response of the roll dynamics (Hung and Givigi (2017)), and stochastic terms are introduced to make the response more realistic. The undamped natural frequency ω_n and the damping ratio ζ are based on the autopilot parameters of the UAV. In this paper, they are drawn from the normal distributions $\mathbb{N}\left(\overline{\omega}_n, \sigma_\omega^2\right)$ and $\mathbb{N}\left(\overline{\zeta}, \sigma_\zeta^2\right)$.

2.2. MDP model of Flocking

We formulate the problem of UAV flocking using a Markov Decision Process (MDP) model. The main elements of the MDP model are described as follows.

2.2.1. State Representation

Denote by $\xi_l := (x_l, y_l, \psi_l, \phi_l)$ the leader's state and $\xi_f := (x_f, y_f, \psi_f, \phi_f)$ the follower's state. Then, the system state $z := (z_1, z_2, z_3, z_4, z_5, z_6)$ is defined as

$$\begin{cases} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} \cos\psi_l & \sin\psi_l \\ -\sin\psi_l & \cos\psi_l \end{bmatrix} \begin{bmatrix} x_f - x_l \\ y_f - y_l \end{bmatrix} \\ z_3 = \psi_f - \psi_l \\ z_4 = \phi_f \\ z_5 = \phi_l \\ z_6 = r_l \end{cases} \tag{2}$$

where (z_1, z_2) denotes the planar position of the follower relative to the leader, z_3 denotes the difference in the heading between the leader and the follower, r_l denotes the leader's roll command. In accordance with Hung and Givigi (2017), the leader's roll command is assumed randomly given rather than predefined, which introduces additional stochasticity to the problem. However, we do not discretize the state space. This makes the problem much more difficult than that Hung and Givigi (2017) deals with.

2.2.2. Action Space

As mentioned, the roll command is the only way for the UAV to change its state. The roll command is updated every one second, during which the autopilot carries out the closed-loop control. To avoid sharp changes in the roll, we define the roll command $a_r \in A$ in a continuous space where:

$$A := [-15^\circ, +15^\circ] \tag{3}$$

If the follower's current roll angle is ϕ, then the next roll angle setpoint r is defined by:

$$r = \begin{cases} r_{\text{bd}} & \text{if } \phi + a_r > r_{\text{bd}} \\ -r_{\text{bd}} & \text{if } \phi + a_r < -r_{\text{bd}} \\ \phi + a_r & \text{otherwise} \end{cases} \tag{4}$$

where $[-r_{\text{bd}}, r_{\text{bd}}]$ is the allowed setpoint range of the roll angle.

2.2.3. Reward Function

The reward function is defined in accordance with Hung and Givigi (2017):

$$\begin{cases} g = -cost \\ cost = \max\left\{d, \dfrac{d_1 \left|z_3\right|}{\pi(1+\omega d)}\right\} \\ d = \max\left\{d_1 - \rho, 0, \rho - d_2\right\} \\ \rho = \sqrt{z_1^2 + z_2^2} \end{cases} \tag{5}$$

where g stands for the immediate reward, ρ is the distance between the leader and the follower (see Figure 1), and ω is a tuning parameter. We note that maximizing the reward means minimizing the cost.

3. Continuous Actor-Critic with Experience Replay (CACER)

In this paper, we propose a continuous actor-critic reinforcement learning algorithm with experience replay (CACER) based on CACLA (Van Hasselt and Wiering (2007, 2009)). CACER consists of an actor and a critic. Different with CACLA that uses a single-layer feed-forward neural network, we use a multilayer perceptron (MLP) to represent the actor that maps the state space to the action space, i.e., $Act^* : S \to A$, where $Act^*(s)$ denotes the optimal action in the state s. We also use an MLP to represent the critic that approximates the state value function: $V : S \to \mathbb{R}$. Denote by (s_k, a_k, r_k, s_{k+1}) the tuple of state, action, and reward at the time step k and $k+1$, respectively. The critic is updated as follows:

$$V_{k+1}\left(s_k|\theta^V\right) = V_k\left(s_k|\theta^V\right) + \beta\delta_k \tag{6}$$

in which δ_k is the Temporal Difference (TD) error defined by:

$$\delta_k = r_k + \gamma \cdot V_k\left(s_{k+1}|\theta^V\right) - V_k\left(s_k|\theta^V\right) \tag{7}$$

where $0 \leq \gamma \leq 1$ is the discount factor, $0 \leq \beta \leq 1$ is the learning rate of the critic. The parameters θ^V of the critic network are adjusted by performing gradient descent with the loss of $\|\delta_k\|^2$.

In contrast to the critic, the actor is updated only when the TD-error is positive, which means the current state is better than expected. Therefore, the probability of selecting the previous action should be increased. The update rule of the actor is defined as follows:

$$Act_{k+1}\left(s_k|\theta^A\right) = \begin{cases} Act_k\left(s_k|\theta^A\right) + \alpha\left(a_k - Act_k\left(s_k|\theta^A\right)\right) & \text{if } \delta_k > 0 \\ Act_k\left(s_k|\theta^A\right) & \text{otherwise} \end{cases} \tag{8}$$

where $0 \leq \alpha \leq 1$ is the learning rate of the actor. The parameters θ^A of the actor network are updated through gradient descent with the loss of $\left\|a_k - Act_k\left(s_k|\theta^A\right)\right\|^2$ when the TD-error is positive.

Gaussian exploration is used for selecting exploratory actions. In other words, the action is randomly selected from the Gaussian distribution $G(x, \mu, \sigma)$ centered at the current output of the actor $Act_k\left(s_k|\theta^A\right)$:

$$G\left(x, Act_k\left(s_k|\theta^A\right), \sigma\right) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\left(x - Act_k\left(s_k|\theta^A\right)\right)^2/2\sigma^2} \tag{9}$$

where σ is the Gaussian exploration parameter. In this paper, the action space is one-dimensional.

The control policy of the leader is defined with environmental dynamics, and we treat the leader as part of the environment for the followers (see Figure 2).

In every training episode, both a follower and a leader are randomly initialized. Then, the follower agent obtains the system state z by combining its own state and the leader's state. The agent selects an action a_r according to the Gaussian exploration (Eq. 11). Consequently, the UAV kinematics model gives the next system state z', and the immediate

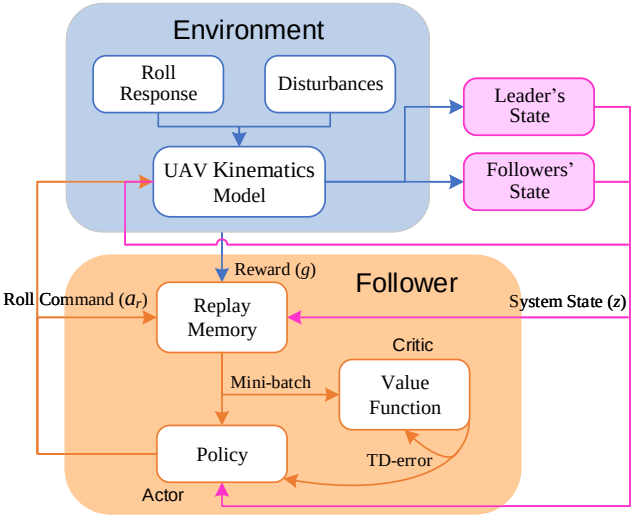

Figure 2: The interaction between a follower and a leader.

reward g is obtained. Compared with the original CACLA algorithm (Van Hasselt and Wiering (2009)), we use the technique of experience replay(Lin (1992)). In other words, every tuple of (z, a_r, g, z') is saved to the experience buffer rather than used only once as done in Van Hasselt and Wiering (2009). Then, a batch training is carried out by random sampling in the experience buffer. If the experience replay buffer is full, then the oldest data is replaced by the latest one. This mechanism alleviates the correlation between the sequential training data while increasing the data utilization.

We summarize the proposed CACER algorithm in Algorithm 1.

Algorithm 1 Continuous Actor-Critic with Experience Replay (CACER)

Input: N_s − maximum time steps; *max episodes* − maximum training episodes

1: Empty experience replay buffer D with capacity N as needed for Algorithm 2; empty experience replay buffer D_1 with capacity N and D_2 with capacity $N/2$ as needed for Algorithm 3
2: **repeat**
3: Initialize $z \leftarrow (\xi_l, \xi_f, r_l)$ randomly; $t = 1$
4: **while** $t \leq N_s$ **do**
5: Select the follower's roll command a_r according to Eq. 9
6: Calculate the follower's roll angle setpoint r_f using Eq. 4
7: Apply the selected r_f and the leader's roll angle setpoint r_l to the UAV dynamics model, respectively
8: Observe the subsequent state $\left(\xi_l', \xi_f'\right)$
9: Calculate the immediate reward g using Eq. 5
10: Choose the leader's roll command from A in Eq. 3 randomly
11: Calculate the leader's next roll angle setpoint r_l' using Eq. 4
12: Create the next system state $z' \leftarrow \left(\xi_l', \xi_f', r_l'\right)$ using Eq. 2
13: Experience replay and network update (see Algorithm 2 and see Algorithm 3)
14: $t \leftarrow t + 1; (\xi_l, \xi_f, r_l) \leftarrow \left(\xi_l', \xi_f', r_l'\right); z \leftarrow z'$
15: **end while**
16: **until** *max episodes*

As described in Line 4 of Algorithm 2, CACER randomly samples transition tuples from the experience replay buffer. This training approach is not as efficient as prioritized experience replay (PER, Schaul et al. (2015)) that considers the importance of samples to speed up the learning. In this paper, we propose a novel experience replay mechanism called double prioritized experience replay (DPER). Its combination with CACER is referred to as CACER-DP, see Algorithm 3.

Considering the actor and the critic of CACER are updated under different conditions, we use two separate experience replay buffer (D_1 for the actor and D_2 for the critic) to save the transition tuples. Specifically, every transition tuple is saved to D_2 for updating the critic network, while D_1 only saves the transition tuples whose TD-errors are positive for updating the actor network. Similar to (Schaul et al. (2015); Hou et al. (2017)), CACER-DP samples a mini-batch of transition tuples from D_2 with the sampling probability $P(i)$

Algorithm 2 Experience Replay and Network Update

Input: z, a_r, g, z' – a state transition sample; N_b – training batch size

1: Delete the oldest tuple in D if $\|D\| = N$
2: Save (z, a_r, g, z') to D
3: Empty the temporal buffer $D' = \varnothing$
4: Sample a mini-batch of N_b tuples $(z^j, a_r^j, g^j, z^{j+1})$ from D randomly
5: **for** each tuple $(z^j, a_r^j, g^j, z^{j+1})$ **do**
6: Calculate the TD-error δ^j using Eq. 7
7: Save the tuple $(z^j, a_r^j, g^j, z^{j+1})$ to D' if $\delta^j > 0$
8: **end for**
9: Update the actor using Eq. 8 with loss of $\frac{1}{\|D'\|}\sum_{j'}\left\|a_r^{j'} - Act\left(z^{j'}|\theta^A\right)\right\|^2$
10: Update the critic using Eq. 6 with loss of $\frac{1}{N_b}\sum_j \left\|\delta^j\right\|^2$

when training the critic:

$$P(i) = \frac{p_i^\lambda}{\sum_n p_n^\lambda} \tag{10}$$

in which the exponent λ determines the degree of prioritization, and p_i is the priority value of the corresponding transition tuple i:

$$p_i = |\delta_i| + \varepsilon \tag{11}$$

where $|\delta_i|$ is the absolute TD-error, and ε is a small positive constant to avoid zero probability.
The loss function for updating the parameters of the critic network can be written as $\mathbb{E}\left(\sum_i w_i \left\|\delta_i\right\|^2\right)$, where w_i is the importance sampling weights defined by:

$$w_i = \frac{(N \cdot P(i))^{-\eta}}{\max_n w_n} \tag{12}$$

where N is the size of D_2, and the exponent η determines the amount of importance sampling correction.

The same with CACLA, the actor of CACER is updated only when the TD-error is positive. It is intuitive to design a different experience replay mechanism for updating the actor. Based on CALCA+Var(Van Hasselt and Wiering (2007)), we define the priority of a transition tuple $j \in D_1$ as follows:

$$p_j = \max\left(\delta_j / \sqrt{\text{var}_j}, \varepsilon\right) \tag{13}$$

where var is the averaged running variance of the TD-errors, which measures whether an action is an outlier. This variance can be tracked over time as follows:

$$\text{var}_{t+1} = (1-\tau)\,\text{var}_t + \tau\delta_t^2 \tag{14}$$

We note that the loss function combined with the importance sampling weights can be rewritten as $\mathbb{E}\left(\sum_j w_j \left\|a_r^j - Act\left(z^j|\theta^A\right)\right\|^2\right)$ when updating the parameters of the actor network.

Algorithm 3 Double Prioritized Experience Replay and Network Update (CACER-DP)

Input: z, a_r, g, z' − a state transition sample; N_b − training batch size

Delete the oldest tuple in D_1 if $\|D_1\| = N/2$
Save (z, a_r, g, z') to D_1 with maximal priority $p_t = \max_{i<t} p_i$ if $\delta > 0$
Delete the oldest tuple in D_2 if $\|D_2\| = N$
Save (z, a_r, g, z') to D_2 with maximal priority $p_t = \max_{i<t} p_i$
Empty the temporal buffer $D' = \varnothing$
Sample N_b tuples $(z^j, a_r^j, g^j, z^{j+1})$ from D_1 with probability $P(j)$ (Eq. 10)
for each tuple $(z^j, a_r^j, g^j, z^{j+1})$ sampled from D_1 **do**
    Compute the importanc sampling weight w^j using Eq. 12
    Calculate the TD-error δ^j using Eq. 7
    Update the running variance of the TD-error using Eq. 14
    Update transition priority according to Eq. 13
    Save the tuple $(z^j, a_r^j, g^j, z^{j+1})$ to D' if $\delta^j > 0$
end for
Update the actor using Eq. 8 with loss of $\frac{1}{\|D'\|} \sum_{j'} w^{j'} \left\| a_r^{j'} - Act\left(z^{j'} | \theta^A\right) \right\|^2$
Sample N_b tuples $(z^i, a_r^i, g^i, z^{i+1})$ from D_2 with probability $P(i)$ (Eq. 10)
for each tuple $(z^i, a_r^i, g^i, z^{i+1})$ sampled from D_2 **do**
    Compute the importance sampling weight w^i using Eq. 12
    Calculate the TD-error δ^i using Eq. 7
    Update transition priority according to Eq. 11
end for
Update the critic using Eq. 6 with loss of $\frac{1}{N_b} \sum_i w^i \left\| \delta^i \right\|^2$

4. Experiments and Results

We compare the proposed algorithms with a range of methods in numerical simulation, and then transfer the learned policy to semi-physical simulation.

4.1. Experiment Settings

In this paper, both the actor and the critic use the same structure of MLP (see Figure 3). Specifically, each MLP consisted of 3 hidden layers. Both the first hidden layer and the second hidden layer had 256 nodes, followed by the third hidden layer with 128 nodes. We note that each hidden layer was followed by a Rectified Linear Unit (ReLU) (Nair and Hinton (2010)) activation function layer. The output layer of the critic used the linear activation function, while the output payer of the actor used the tanh activation function. This guaranteed the output of the actor was within the range of $[-1, +1]$, which coule be linearly magnified to the action space $[-15, +15]$, as mentioned in Eq. 3. We used TensorFlow [1] and Keras [2] for implementation.

The desired number of training episodes (*max episodes*) was set to 50000, in which each episode had a maximal number of 30 time steps ($N_s = 30$), i.e. 30 seconds. The network

1. https://www.tensorflow.org/
2. https://keras.io/

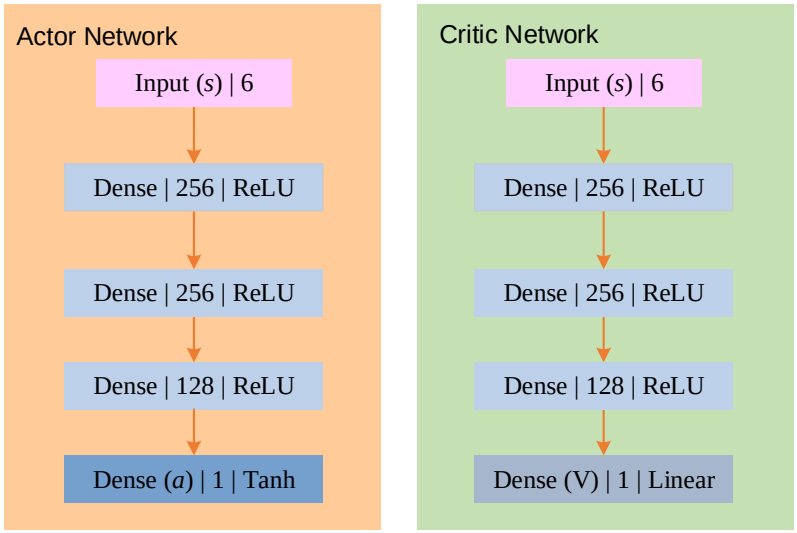

Figure 3: The network structure for CACER. Each layer is featured by its type, dimension and activation function.

parameters of the actor and the critic were both trained using the Adam optimizer (Kingma and Ba (2014)) with the MSE loss function. The exploration parameter and the learning rates (σ, α, β) were annealed exponential from the initial values (0.25, 0.01, 0.001) to the minimum values (0.025, 0.001, 0.0001), respectively. In other words, these parameters were multiplied by 0.99 every 100 episodes. According to Schaul et al. (2015), the exponent λ in Eq. 10 and the small positive constant ε in Eq. 11 were set to 0.6 and 0.01, respectively. The exponent η in Eq. 12 was annealed linearly from 0.4 to 1 over 6×10^5 time steps. Additionally, when tracking the variance of TD-errors in Eq. 14, we used $\text{var}_0 = 0$ and $\tau = 0.001$ (Van Hasselt and Wiering (2007)). The empirical values of the parameters are listed in Table 1, which are in accordance with Hung et al. (2015); Hung and Givigi (2017); Lillicrap et al. (2015).

Table 1: Parameter settings

Name	Value	Name	Value
d_1	40	d_2	65
ω	0.05	α_g	9.8
$\overline{s}$	10	σ_s	0.8
$\overline{\omega}_n$	6.3	σ_ω	0.1
$\overline{\zeta}$	0.5561	σ_ζ	0.01
$(\overline{\eta}_x, \overline{\eta}_y, \overline{\eta}_\phi)$	0	$(\sigma_x, \sigma_y, \sigma_\phi)$	1
N	100000	γ	0.95
N_b	32	r_{bd}	30°

4.2. Results of Numerical Simulation

We have compared CACER-DP (Algorithm 1 + Algorithm 3) with several other methods including the original CACLA, a state-of-the-art deep reinforcement learning algorithm DDPG, CACER with random experience replay (Algorithm 1 + Algorithm 2), and CACER with the original PER (Schaul et al. (2015); Hou et al. (2017)) (called CACER-P). The above algorithms used the same deep network structure and the same parameters except for CACLA. We note that CACLA only used a single-layer feed-forward neural network with 32 nodes, as recommended in Van Hasselt and Wiering (2007). We have compared the above algorithms in the training stage as well as in the testing stage.

4.2.1. Results of Policy Learning

In the training process, we calculated the averaged reward G_{Ave} obtained within a certain number of N_e episodes:

$$G_{\text{Ave}} = \frac{1}{N_e N_s} \sum_{k=1}^{N_e} \sum_{i=1}^{N_s} g_i^k \tag{15}$$

where g_i^k denoted the immediate reward decided by Eq. 5, and $N_e = 100$.

As shown in Figure 4, CACLA did not perform as well as the other approaches, which demonstrated the advantage of DNNs as function approximators and the better training efficiency of the experience replay technique. The reward curves of CACER, CACER-P and CACER-DP grew slightly slower than DDPG at the beginning, but they all achieved higher rewards than DDPG after about 1000 episodes of training and became stable afterwards. This meant that the proposed CACER based algorithms were slightly better than DDPG

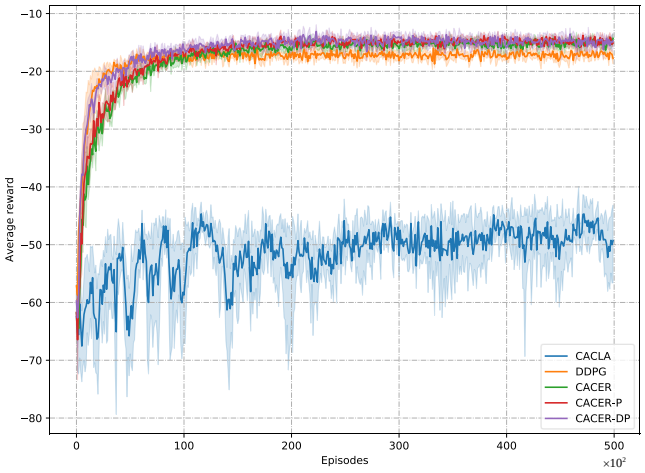

Figure 4: Averaged reward comparison of CACLA, DDPG, CACER, CACER-P and CACER-DP in the training stage. We note that $N_e = 100$, i.e., we calculated G_{Ave} every 100 episodes.

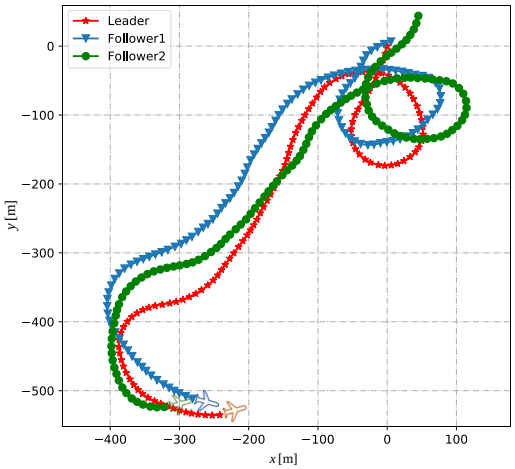

Figure 5: The trajectory results of the learned CACER policy in the numerical simulation.

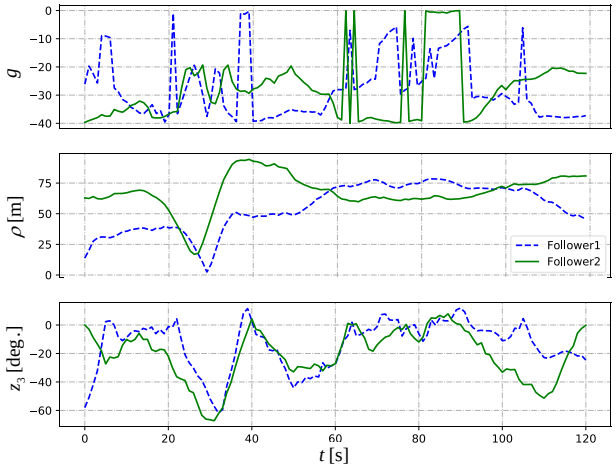

Figure 6: g, ρ, and z_3 results of the learned CACER policy in the numerical simulation.

for this RL task. Further results of policy testing are given in Table 2. In addition, the curves of CACER and CACER-P almost overlapped, indicating that the direct combination of the popular PER technique and CACER (i.e., CACER-P) did not work well. In contrast, the curve of CACER-DP grew much faster than CACER and CACER-P in the beginning, which proved the effectiveness of the proposed DPER mechanism. Besides, CACER-DP obtained a higher averaged reward than DDPG in most training episodes, meanwhile had the similar learning speed in the early episodes.

4.2.2. Results of Policy Testing

After the training stage, we first tested the learned CACER policy in a flocking task with two followers as shown in Figure 5.

In this experiment, the time steps $N_s = 120$, i.e. the testing took 2 minutes. The actions of the leader were randomly generated, and the followers used the learned CACER policy to follow the leader. The trajectory results illustrated that the followers succeeded to follow the leader most of the time. In addition, we recorded the distances between the leader and the followers (ρ), the difference in the heading (z_3) and the immediate reward (g) in Figure 6. The best performance of the following behavior was achieved between 50s and 90s, during which the heading of the leader did not change much.

We also compared the CACER-based algorithms with CACLA and DDPG as well as the imitation policy in the testing stage. The rewards were also averaged using Eq. 15 over 100 experiments. The imitation follower selected the same roll angle setpoint as the leader. Intuitively, the imitation policy seemed to be a good choice for the follower when the initial states of them were exactly the same. In each experiment, the initial states of the imitation follower were the same with the leader, while other followers' initial states were randomly

generated and taking the same value for comparison. This meant the task was easier for the imitation follower than the other followers.

Table 2 compares the results of the averaged reward and its variance Var. the averaged reward G_{Ave} obtained by the three CACER-based policies and the DDPG policy were much higher than the reward of the CACLA policy and the imitation policy. Also, the corresponding variances were also much lower. Overall, the CACER-based policies achieved similar performance with the DDPG policy. Specifically, the CACER-DP policy obtained a higher reward than the DDPG policy while having a lower variance. However, the CACER policy and the CACER-P policy were less stable with higher variances than DDPG. This illustrated the advantage of the proposed DPER mechanism in combination with the CACER algorithm.

Table 2: Policy testing results of CACLA, DDPG, CACER, CACER-P, CACER-DP, and imitation.

Method	G_{Ave}	Var
CACLA	-98.38	3949.60
Imitation	-65.61	2892.08
DDPG	-12.90	12.78
CACER	-12.05	26.70
CACER-P	**-11.37**	16.26
CACER-DP	-11.39	**12.08**

4.3. Results of Semi-Physical Simulation

We further tested the CACER algorithm in semi-physical simulation experiment. The high-fidelity semi-physical simulation system (Wang et al. (2018); Ma et al. (2017)) consisted of an X-Plane 10 Flight Simulator [3], two PX4[4] / Pixhawk[5] autopilots, two onboard computers and a SuperStation ground control station(see Figure 7). X-Plane 10 can simulate complex environmental conditions such as weather changes and wind disturbance. SuperStation can control multiple UAVs by switching control modes and planning path. The above modules were operated or connected with Robot Operating System (ROS) [6]. We chose the HilStar17F model in X-Plane for both the leader and the follower. Their control policies were running independently on their onboard computers that were connected through an RJ45 patch cable. The leader sent its state information to the follower via UDP socket.

In one experiment, we started the leader and the follower in the MANUAL mode using SuperStation. First, the MISSION mode was turned on and the UAVs kept a certain distance from each other by following predefined paths. Then, the OFFBOARD mode was turned on. The leader chose a random roll command at each time step, and the follower used the CACER policy learned in the previous numerical simulation experiment without

3. https://www.x-plane.com/manuals/desktop/
4. https://px4.io/
5. http://pixhawk.org/
6. http://www.ros.org/

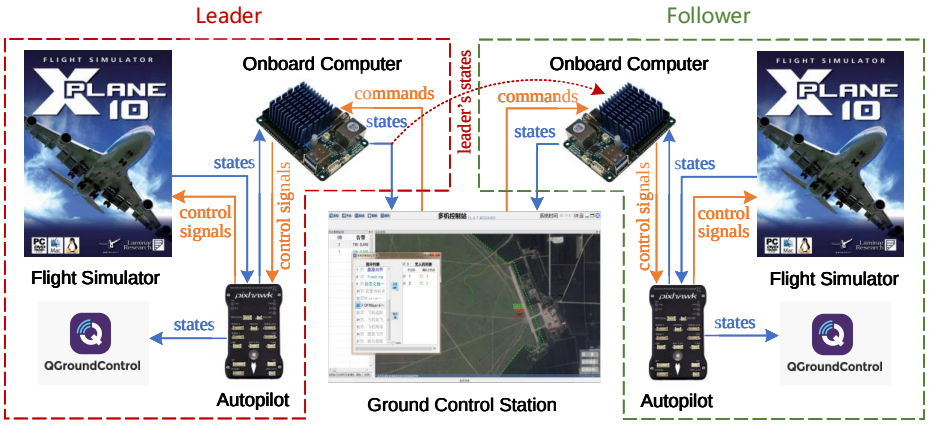

Figure 7: The high-fidelity semi-physical simulation system

any revision to select a roll angle setpoint. After 120 seconds, the RETURN mode was turned on and the experiment ended.

The trajectory result is shown in Figure 8, and the results of the distance between the leader and followers (ρ), the difference in the heading (z_3) and the immediate reward (g) are shown in Figure 9. In the beginning of the 120 seconds, the distance was beyond 125 meters, and the follower was in front of the leader. As both of the UAVs had the same speed of 10 m/s, the follower had to decrease the distance by changing its heading angle. After about 51 seconds (time steps), the distance dropped below 65 meters for the first time. Then, the distance was kept between 25 to 50 meters most of the time. The results illustrated that the trained CACER policy was able to deal with new situations without the need of parameter tuning.

5. Conclusion

In this paper, we have proposed a CACER algorithm to solve the leader-follower problem for flocking a squad of fixed-wing UAVs. In contrast to the literature, we have solved a more challenging problem in continuous state and action spaces without the need to discretize the spaces. Compared with the original CACLA algorithm, the CACER algorithm uses a more advanced MLP network to represent both the actor and the critic. In addition, the training efficiency has been improved by using the proposed double prioritized experience replay technique. In the numerical simulation, the results have shown that the CACER-based algorithms are efficient for learning the UAV following behavior. Specifically, CACER-DP is better than the state-of-the-art DDPG algorithm in both the learning and testing stages. Then, the policy learned in the numerical simulation has been transferred directly to the semi-physical simulation without any tuning, and the UAVs are able to adjust its distance to keep up with the leader autonomously. However, we have made the same assumptions with the previous research that the UAVs fly at a constant average speed and fixed different heights to simplify the collision problem. In the future work, we will relax these assumptions towards more UAVs flocking in real-world environments.

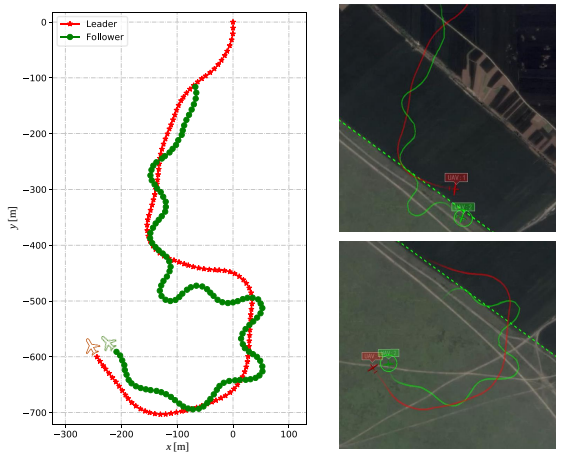

Figure 8: The trajectory results of the learned CACER policy in the semi-physical simulation.

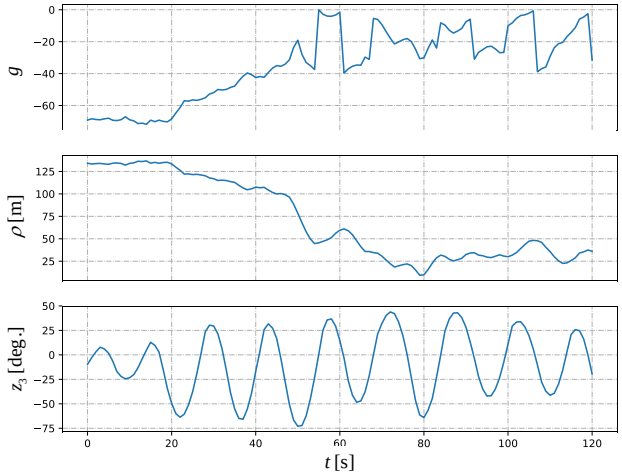

Figure 9: g, ρ, and z_3 results of the learned CACER policy in the semi-physical simulation.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (61906203 and 61803377) and the National Key Laboratory of Science and Technology on UAV, Northwestern Polytechnical University (614230110080817).

References

Yuenan Hou, Lifeng Liu, Qing Wei, Xudong Xu, and Chunlin Chen. A novel ddpg method with prioritized experience replay. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 316–321. IEEE, 2017.

Shao-Ming Hung and Sidney N. Givigi. A q-learning approach to flocking with uavs in a stochastic environment. *IEEE Transactions on Cybernetics*, 47(1):186–197, 2017.

Shao-Ming Hung, Sidney N. Givigi, and Noureldin Aboelmagd. A dyna-q(lambda) approach to flocking with fixed-wing uavs in a stochastic environment. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 1918–1923, 2015.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Morihiro Koichiro, Isokawa Teijiro, Nishimura Haruhiko, and Matsui Nobuyuki. Characteristics of flocking behavior model by reinforcement learning scheme. In *International Joint Conference on SICE-ICASE*, pages 4551–4556, 2006.

Hung M. La, Robert Lim, and Weihua Sheng. Multirobot cooperative learning for predator avoidance. *IEEE Transactions on Control Systems Technology*, 23(1):52–63, 2015.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

Zhaowei Ma, Chang Wang, Yifeng Niu, Xiangke Wang, and Lincheng Shen. A saliency-based reinforcement learning approach for a uav to avoid flying obstacles. *Robotics and Autonomous Systems*, 100:108–118, 2017.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on International Conference on Machine Learning (ICML)*, 2010.

Basant K. Sahu and Bidyadhar Subudhi. Flocking control of multiple auvs based on fuzzy potential functions. *IEEE Transactions on Fuzzy Systems*, 26(5):2539–2551, 2018.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Hado Van Hasselt and Marco A Wiering. Reinforcement learning in continuous action spaces. In *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 272–279, 2007.

Hado Van Hasselt and Marco A Wiering. Using continuous action spaces to solve discrete problems. In *IEEE International Joint Conference on Neural Networks*, pages 1149–1156, 2009.

Mnih Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Xiangke Wang, Jiahu Qin, and Changbin Yu. Iss method for coordination control of nonlinear dynamical agents under directed topology. *IEEE Transactions on Cybernetics*, 44 (10):1832–1845, 2014.

Yajing Wang, Xiangke Wang, Shulong Zhao, and Lincheng Shen. Vector field based sliding mode control of curved path following for miniature unmanned aerial vehicles in winds. *Journal of Systems Science and Complexity*, 31(1):302–324, 2018.

Jiao Wu, Rui Wang, Ruiying Li, Hui Zhang, and Xiaohui Hu. Multi-critic ddpg method and double experience replay. In *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 165–171. IEEE, 2018.