Fixed-Wing UAVs flocking in continuous spaces: A Deep reinforcement learning approach

Article *in* Robotics and Autonomous Systems · June 2020

DOI: 10.1016/j.robot.2020.103594

3 authors, including:

Chao Yan
National University of Defense Technology
8 PUBLICATIONS **44** CITATIONS
SEE PROFILE

Chang Wang
National University of Defense Technology
27 PUBLICATIONS **141** CITATIONS
SEE PROFILE

Robotics and Autonomous Systems 131 (2020) 103594

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Fixed-Wing UAVs flocking in continuous spaces: A deep reinforcement learning approach ☆

Chao Yan, Xiaojia Xiang, Chang Wang *

College of Intelligence Science and Technology, National University of Defense Technology, Changsha 410073, China

ARTICLE INFO

Article history:
Received 30 January 2020
Received in revised form 17 April 2020
Accepted 8 June 2020
Available online 11 June 2020

Keywords:
Fixed-wing UAV
Flocking
Reinforcement learning
Actor–critic

ABSTRACT

Fixed-Wing UAVs (Unmanned Aerial Vehicles) flocking is still a challenging problem due to the kinematics complexity and environmental dynamics. In this paper, we solve the leader–followers flocking problem using a novel deep reinforcement learning algorithm that can generate roll angle and velocity commands by training an end-to-end controller in continuous state and action spaces. Specifically, we choose CACLA (Continuous Actor–Critic Learning Automation) as the base algorithm and we use the multi-layer perceptron to represent both the actor and the critic. Besides, we further improve the learning efficiency by using the experience replay technique that stores the training data in the experience memory and samples from the memory as needed. We have compared the performance of the proposed CACER (Continuous Actor–Critic with Experience Replay) algorithm with benchmark algorithms such as DDPG and double DQN in numerical simulation, and we have demonstrated the performance of the learned optimal policy in semi-physical simulation without any parameter tuning.

1. Introduction

In recent years, Unmanned Aerial Vehicles (UAVs) have become popular for entertainment purposes such as aerial photography and light show, or aiding in high-risk missions such as disaster rescue [1] and anti-terrorist operations [2]. For example, a team of quad-rotor UAVs was coordinated to play the piano [3]. In 2018, Intel broke the Guinness World Record for simultaneously flying more than a thousand of quad-rotor UAVs [4]. However, compared with quad-rotor UAVs, controlling a fixed-wing UAV is often more difficult and requires a different approach, considering the nonholonomic constraints due to the kinematics of fixed-wing UAVs [5]. Therefore, a squad of fixed-wing UAVs flocking is even more challenging because it is difficult to synchronize the behaviors of the fixed-wing UAVs in the same way as the quad-rotor UAVs. Besides, they have to keep appropriate distances from each other while avoiding collisions, and each UAV has limited information about other UAVs' states or control policies due to communication constraints. In this paper, we solve the fixed-wing UAVs flocking problem [6] using a leader–followers model that the leader randomly changes its control policy and a follower has to learn how to flock with the leader by controlling its roll angle and velocity autonomously.

Early in 1987, Reynolds [6] first established the three basic rules to simulated flocking, termed as separation, alignment, and cohesion. Inspired by Reynolds, the flocking framework has been widely used in multiagent systems due to its simplicity and effectivity. For example, Olfati-Saber [7] proposed and analyzed several distributed control algorithms for free-flocking and constrained flocking; this author also discussed consensus problems in networked multi-agent systems and diverse applications including flocking [8,9]. More recently, Jafari et al. [10–12] designed several biologically-inspired distributed intelligent controllers for flocking of multi-agent systems. Moreover, Rezaee et al. [13–15] proposed several control algorithms for cooperative control and obstacle avoidance based on the potential field. Focused on aerial vehicles, [16–18] considered the UAV flocking problem with the consensus theory.

Traditional control systems usually require precise models of the plant and disturbances to design control laws [5,19]. However, these models are complex, non-linear, and time-varying, therefore are hard to obtain in real-world environments [5,19]. As an alternative approach, Reinforcement Learning (RL) methods require no such model for developing the flocking behavior. For example, the multi-agent flocking problem has been studied in the predator–prey scenario [20,21]. Q-learning [22] was used for particle-based agents to learn how to flock in a self-organized way to avoid predators. Additionally, Mhamdi et al. [23] used Q-learning with eligibility trace to address the gathering problem in

☆ This work was co-supported by National Natural Science Foundation of China under grant 61906203 and National Key Laboratory of Science and Technology on UAV, Northwestern Polytechnical University, China under grant 614230110080817.

* Corresponding author.
E-mail address: wangchang07@nudt.edu.cn (C. Wang).

https://doi.org/10.1016/j.robot.2020.103594
0921-8890/

a non-communicating and partially observable environment. In recent years, Deep Reinforcement Learning (DRL) methods such as Deep Q-Networks (DQN) [24], double DQN [25], and Deep Deterministic Policy Gradient (DDPG) [26] have attracted great attention. In addition to playing Atari games [24,25], DRL-based methods also have many successful applications in the field of robots, such as path planning [27], obstacle avoidance [28], and attitude control [29]. In the flocking control field, a DRL-based approach was proposed in [30] for flocking and navigation of quad-rotor UAVs in complex environments. However, fixed-wing UAVs flocking is more difficult because environmental dynamics such as airspeed and side wind have to be considered, which would make the learned policy hard to converge.

Not much work has been done about fixed-wing UAVs flocking in dynamic environments. Quintero et al. [31] used the dynamic programming approach for fixed-wing UAVs flocking, which could be reformulated in the context of RL. Hung et al. [5,32] used the Dyna-Q(λ) algorithm and the Q(λ) algorithm to learn the control policy for the leader–follower problem. The algorithms were proved effective for fixed-wing UAVs flocking in numerical simulations. However, the authors made several assumptions that are inappropriate for controlling fixed-wing UAVs in more realistic environments. First, the state and action spaces were discretized for simplification. Second, the velocity of each UAV was assumed the same. In our previous work, we considered the fixed-wing UAVs flocking problem under the assumption that UAVs fly at a constant average speed [33]. In this paper, we will relax this assumption towards developing fixed-wing UAVs flocking behavior in semi-physical environments.

In this paper, we solve the fixed-wing UAVs flocking problem within the DRL framework in continuous state and action spaces. We choose Continuous Actor–Critic Learning Automation (CACLA) [34,35] as the base algorithm because it is suitable for the continuous problem and easy to implement. In recent years, a variety of improvements have been done based on the original CACLA algorithm. For example, the actor's leaning rule was corrected by the critic's absolute Temporal Difference (TD) error in Corrected CACLA [36]. Sampled Policy Gradient (SPG) was proposed as an off-policy version of CACLA [37]. However, these methods are still far from real-world robotic applications. In our previous work, we proposed an active affordance learning approach integrated an active exploration policy with CACLA for service robots to perform household tasks in the real world [38]. In this paper, we will modify the CACLA algorithm within the DRL framework to address the fixed-wing UAVs flocking problem.

To the best of our knowledge, we are the first to develop a DRL-based approach for the flocking of fixed-wing UAVs in continuous state and action spaces. The main contributions of this paper are as follows:

- A novel DRL algorithm has been proposed to solve the fixed-wing UAVs flocking problem in continuous spaces. Both the actor and the critic are approximated by multilayer perceptron, and the learning process is further sped up using the experience replay mechanism.
- An end-to-end flocking controller can be trained from scratch using the proposed DRL algorithm, which can generate continuous roll angle and velocity commands for the followers based on the state of the leader–followers system.
- The proposed DRL algorithm has been compared with the state-of-the-art DRL algorithms in numerical simulation, and the learned optimal policy can be directly transferred from the numerical simulation to the semi-physical simulation without any parameter tuning.

The rest of the paper is organized as follows. Section 2 briefly introduces the background of DRL algorithms. Section 3 formulates the flocking problem. Section 4 presents the DRL-based flocking approach. Section 5 describes the experiments and results. Finally, Section 6 concludes the paper and outlines our plans for future work.

2. Background

2.1. Reinforcement learning

Reinforcement learning (RL) has been used to solve sequential decision-making problems. An RL agent aims to learn an optimal policy through sequential interaction with the environment which gives the agent reward signals to shape its behaviors.

An RL problem can be modeled by a Markov Decision Process (MDP) [22], represented as a tuple $(S, A, P(s, s', a), R(s, s', a))$, where S is the state space; A is the action space; $P(s, s', a)$ is the state transition model that describes the probability of transmitting from the current state $s \in S$ to the next state $s' \in S$ after executing the given action $a \in A$; $R(s, s', a)$ is the reward function. The aim is to find the optimal policy $\pi^* : S \to A$ that maps from the state space to the action space to maximize the discounted future return:

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}, \tag{1}$$

where T is the terminal time-step, $0 \leq \gamma \leq 1$ is the discounted factor that trades off the influence of the immediate reward and the future reward, and r_t is the immediate reward at the time step t.

2.2. DQN and experience replay

Q-learning [22] is one of the most popular off-policy algorithms that leans to predict the state–action values (also called Q-values). The Q-value is defined as follows:

$$Q(s_t, a_t) := \mathbb{E}[R_t | s = s_t, a = a_t]. \tag{2}$$

where the symbol $\mathbb{E}[\cdot]$ stands for the mathematical expectation.

According to the Bellman equation [22], the Q-value is updated as follows:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha^Q \cdot \delta_t, \tag{3}$$

in which the TD-error δ_t is:

$$\delta_t = r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t), \tag{4}$$

where s_t is the state at the time step t, s_{t+1} is the subsequent state after executing the action a_t, and $0 < \alpha^Q < 1$ is a learning rate.

Once the Q-values are obtained, a greedy policy can be constructed by selecting the action with the highest Q-value However, Q-learning stores the Q-values in a table, which is not suitable for continuous problems and high-dimensional state–action spaces. Alternatively, Deep Neural Networks (DNNs) have been used as function approximators to predict the Q-values. Combining the advantages of Convolutional Neural Networks (CNNs) and Q-learning, Deep Q-Networks (DQN) can achieve human-level performance in playing Atari games [24].

However, the training of DNNs in the RL framework is not always converged due to that the sequential samples obtained from exploration are correlated, which violates the requirement of the backpropagation algorithm that the training data should be independent and identically distributed. The experience relay technique [39] has been proposed for DQN to guarantee the independence of the training data. Specifically, every tuple (s_t, a_t, r_t, s_{t+1}) obtained from exploration is stored in an experience buffer. If the buffer is full, the oldest experience will be

replaced by the latest one. At each time step of training, the weights of DNNs are updated by randomly sampling a mini-batch data from the experience buffer. This mechanism alleviates the correlation between the sequential training data while increasing the data utilization. As a result, the stability and performance of DQN can be improved [24].

2.3. CACLA

Both Q-learning and DQN deal with discrete actions while Continuous Actor–Critic Learning Automaton (CACLA) [34,35] has been proved effective for continuous state and action spaces.

As an actor–critic algorithm, CACLA consists of an actor and a critic. The actor is the policy that maps from the state space to the action space, and the critic is the state value function defined as follows:

$$V(s_t) := \mathbb{E}[R_t | s = s_t]. \tag{5}$$

The critic is updated using TD learning [22]:

$$V(s_t) = V(s_t) + \beta \cdot \delta_t, \tag{6}$$

where the TD-error δ_t is defined as follows:

$$\delta_t = r_{t+1} + \gamma \cdot V(s_{t+1}) - V(s_t), \tag{7}$$

where $0 < \beta < 1$ is the critic's learning rate.

The actor is updated only when the TD-error is positive, which indicates the current state is better than expected. Therefore, the probability of selecting the previous action a_t should be increased. As a result, the actor Act is updated as follows:

$$Act(s_t) = Act(s_t) + \alpha (a_t - Act(s_t)) \text{ if } \delta_t > 0, \tag{8}$$

where $0 < \alpha < 1$ is the actor's learning rate.

CACLA is easy to implement and performs well in a range of continuous problems [34,35]. However, it only uses a single-layer feed-forward neural network as the function approximator for both the actor and the critic. As a result, CACLA can hardly solve more complex RL problems that require the use of deep learning techniques. In this paper, we will improve the original CACLA in the DRL framework.

2.4. DDPG

As one of the state-of-the-art DRL algorithms, the Deep Deterministic Policy Gradient (DDPG) algorithm is able to solve RL problems in high-dimensional state space and continuous action space [40]. DDGP also uses the actor–critic architecture that both the policy and the value function are implemented by DNNs.

Specifically, DDPG uses a Q-value network to represent the critic, and the training of the critic is based on DQN. The corresponding loss function L is defined as follows:

$$L(\theta^Q) = \mathbb{E}\left[\left(y_t - Q(s_t, a_t | \theta^Q)\right)^2\right], \tag{9}$$

where

$$y_t = r_t + \gamma Q\left(s_{t+1}, \arg\max_a Q(s_{t+1}, a) | \theta^Q\right), \tag{10}$$

where θ^Q is the parameter of the critic network.

Different with the critic, the actor of DDPG is parameterized as $\mu(s|\theta^\mu)$ that uses policy gradient to update its parameters as follows:

$$\begin{aligned} \nabla_{\theta^\mu} \mu &\approx \mathbb{E}\left[\nabla_{\theta^\mu} Q(s, a|\theta^Q)\big|_{s=s_t, a=\mu(s_t|\theta^\mu)}\right] \\ &= \mathbb{E}\left[\nabla_a Q(s, a|\theta^Q)\big|_{s=s_t, a=\mu(s_t)} \nabla_{\theta_\mu} \mu(s|\theta^\mu)|_{s=s_t}\right]. \end{aligned} \tag{11}$$

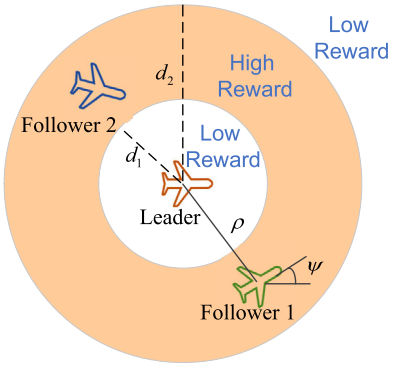


Fig. 1. The top view of the leader–follower topology.

The same with DQN, DDPG also uses the experience replay technique to relieve the correlation issue between the sequential samples. Besides, it uses the target network to stabilize the training process. However, instead of directly cloning the weights with a certain frequency (e.g., every 4 time steps), DDPG uses soft target updates. Denote by θ^- the parameter of the target network and θ the parameter of the online network, then the weight of the target network is updated as follows:

$$\theta^- \leftarrow \tau\theta + (1 - \tau)\theta^-, \tag{12}$$

where $\tau \ll 1$ is the soft update rate. This modification can improve learning stability.

3. Problem statement

In this paper, we deal with the problem of flocking control with fixed-wing UAVs in the context of deep reinforcement learning. In our scenario, the only one leader is followed by several RL agents (i.e., followers) that learn to keep a certain distance with the leader. In accordance with [5,32], we assume that the leader already has its own task-specific control policy, such as searching a given area or tracking a ground vehicle. The leader broadcasts its state to the followers through a wireless communication channel, including its position, pose, and airspeed, et al. A follower has to control its roll angle and velocity to keep a certain distance from the leader ($d_1 < \rho < d_2$) in the top view (see Fig. 1). To avoid the collisions among the UAVs, we make the same assumption with the previous work [5,31,32] that the followers and the leader fly at different altitudes. Therefore, the followers can use the same control policy and the aggregate behavior emulates flocking in a leader–follower topology. Additionally, each follower selects roll angle and velocity setpoints regulated by an autopilot using a Proportional–Integral–Derivative (PID) controller, and the roll angle setpoint and the velocity setpoint are both updated once every second.

3.1. Fixed-wing UAV kinematics model

Reinforcement learning methods typically require many episodes of trial-and-error training through continuous interaction between the agent and the environment [36]. However, it is infeasible for a UAV to learn an optimal policy from scratch in the physical world. Alternatively, training in simulation also requires many samples. In the first step, we establish the numerical kinematics model of a fixed-wing UAV with stochastic disturbances to generate samples for the DRL-based flocking controller. Then, we generalize the learned policy to a semi-physical environment that uses a real autopilot to control the UAVs.

In the real world, a fixed-wing UAV's kinematics can be described by a 6 degree of freedom (DoF) aircraft model. However, given the assumption that each UAV flies at a fixed altitude, we use a simplified 4-DoF model [5,31,32] instead. In order to compensate for the loss of unmodeled dynamics as well as to simulate the environmental disturbances such as wind gusts and control noises that the UAV would encounter in the physical world, we introduce stochasticity into the UAV's kinematics. Specifically, the stochastic terms for representing disturbances are added in the roll, airspeed, and each of the substates of the 4-DoF model. The stochastic kinematics model of the UAV is expressed as follows:

$$\dot{\xi} = \frac{d}{dt}\begin{Bmatrix} x \\ y \\ \psi \\ \phi \\ v \end{Bmatrix} = \begin{Bmatrix} v\cos\psi + \eta_x \\ v\sin\psi + \eta_y \\ -\left(\alpha_g/v\right)\tan\phi + \eta_\psi \\ f\left(\phi, \phi_{ref}\right) \\ f\left(v, v_{ref}\right) \end{Bmatrix}, \tag{13}$$

where (x, y) is the planar position of the UAV; ψ is the heading angle; ϕ is the roll angle; v is the airspeed; α_g is the acceleration due to gravity. The disturbance terms $\left(\eta_x, \eta_y, \eta_\psi\right)$ leading to the changes of the UAV's planar position and heading 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. In addition, the function $f\left(\phi, \phi_{ref}\right)$ and $f\left(v, v_{ref}\right)$ describe the response relationship between (i) the roll angle setpoint ϕ_{ref} and the actual roll angle ϕ, i.e., the roll dynamics, and (ii) the desired/reference airspeed v_{ref} and the actual airspeed v, this is the airspeed dynamics, respectively.

In order to simulate the initial condition response of the roll dynamics, we use a second-order system [5,32] where the undamped natural frequency ω_n and the damping ratio ζ are determined according to the autopilot parameters of the UAV. Additionally, stochastic terms are also introduced to make the response more realistic. In this paper, the parameters of the second-order system (i.e., ω_n and ζ) are drawn from the normal distributions $\mathbb{N}\left(\overline{\omega}_n, \sigma_\omega^2\right)$ and $\mathbb{N}\left(\overline{\zeta}, \sigma_\zeta^2\right)$.

Instead of considering a second-order system similar to the roll dynamics, we use a simple linear model for the airspeed dynamics to reduce the computational complexity. Therefore, we replace the integral process with the average airspeed to calculate the planar position of the UAV. Additionally, we also introduce stochasticity in the airspeed. The average airspeed in the interval is selected by the following expression and held constant until the next airspeed setpoint is updated:

$$\overline{v} = \frac{1}{2}\left(v + v_{ref}\right) + \mathbb{N}\left(0, \sigma_v^2\right), \tag{14}$$

where $\mathbb{N}\left(0, \sigma_v^2\right)$ is a normal distribution with zero mean.

We note that the chosen UAV kinematics model is used to generate the state transition data for training the flocking behavior in the numerical environment. As the introduced stochasticity is too complex, the state transition probability is difficult to be explicitly calculated. Therefore, we choose the model-free RL approach rather than the model-based RL approach.

3.2. MDP model of flocking

The problem of UAV flocking can be modeled as a Markov decision process. The main elements of the MDP are described as follows.

3.2.1. State representation

As described by the kinematics model of UAV, the UAV's state can be represented by a tuple $\xi := (x, y, \psi, \phi, v)$. Denote by $\xi_l := (x_l, y_l, \psi_l, \phi_l, v_l)$ the leader's state and $\xi_f := (x_f, y_f, \psi_f, \phi_f, v_f)$ the follower's state. Combining ξ_l with ξ_f, we create the system state $s := (s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9)$ as follows:

$$\begin{cases} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} \cos\psi_l & \sin\psi_l \\ -\sin\psi_l & \cos\psi_l \end{bmatrix}\begin{bmatrix} x_f - x_l \\ y_f - y_l \end{bmatrix} \\ s_3 = \psi_f - \psi_l \\ s_4 = \phi_f \\ s_5 = \phi_l \\ s_6 = \phi_{ref}^l \\ s_7 = v_f \\ s_8 = v_l \\ s_9 = v_{ref}^l \end{cases}, \tag{15}$$

where (s_1, s_2) represents the planar position of the follower relative to the leader; s_3 denotes the difference in the heading between the leader and the follower. Additionally, ϕ_{ref}^l and v_{ref}^l denote the leader's roll-angle and velocity setpoint, respectively.

In practice, the leader's roll and velocity commands are determined by its own task-based control policy. However, in order to introduce additional stochasticity to the flocking problem, we assume that the leader's control commands (i.e., roll and velocity) are generated randomly in the training stage. We note that we do not discretize the system state space for simplification as in the literature [5]. The continuous state space makes the problem much more difficult than the discrete case.

3.2.2. Action space

As mentioned above, the UAV is maneuvered by selecting the roll and velocity commands, which are updated every one second. During the interval, the two commands are fixed. According to these commands, the autopilot carries out the closed-loop control with a PID controller. Considering the adverse effects caused by sharp changes in the roll and the limitation of UAV's maximum acceleration, we define the roll command $a_r \in A_r$ and the velocity command $a_v \in A_v$ in a continuous space where

$$\begin{cases} A_r := [-10^\circ, +10^\circ] \\ A_v := [-1, +1] \end{cases}. \tag{16}$$

Denote by ϕ and v the UAV's current roll angle and its current airspeed, respectively. The next roll angle setpoint ϕ_{ref} is defined by:

$$\phi_{ref} = \begin{cases} r_{\text{bd}} & \text{if } \phi + a_r > r_{\text{bd}} \\ -r_{\text{bd}} & \text{if } \phi + a_r < -r_{\text{bd}} \\ \phi + a_r & \text{otherwise} \end{cases}, \tag{17}$$

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

The next velocity setpoint v_{ref} is determined by:

$$v_{ref} = \begin{cases} v_{\max} & \text{if } v + a_v > v_{\max} \\ v_{\min} & \text{if } v + a_v < v_{\min} \\ v + a_v & \text{otherwise} \end{cases}, \tag{18}$$

where the v_{max} and v_{min} represent the maximum velocity and the minimum velocity of the UAV, respectively.

3.2.3. Reward function

Similar to [5], we define the reward function as follows:

$$\begin{cases} r = -Cost \\ Cost = \max\left\{d, \dfrac{d_1\,|s_3|}{\pi(1+\omega d)}\right\} \\ d = \max\{d_1 - \rho, 0, \rho - d_2\} \\ \rho = \sqrt{s_1^2 + s_2^2} \end{cases}, \tag{19}$$

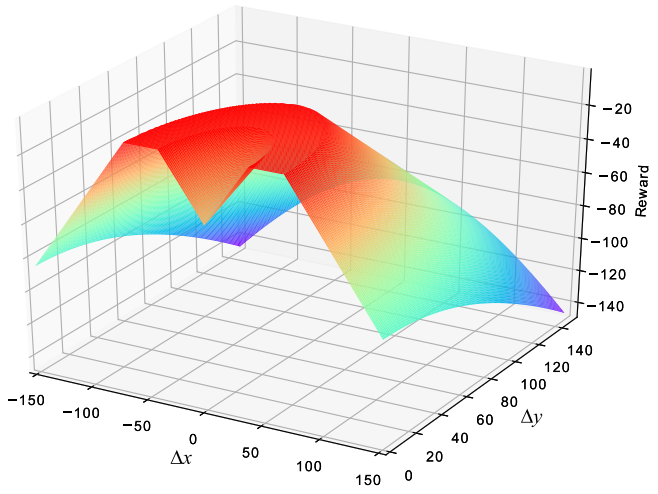


Fig. 2. Illustration of the reward for a subset of states.

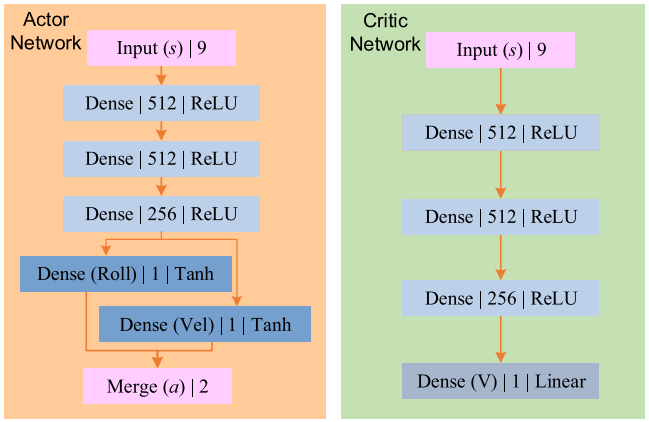


Fig. 3. The network structure for CACER. Each layer is featured by its type, dimension and activation function. We note that the Merge layer means directly combining the multiple streams into a single one.

where r is the immediate reward, ρ is the distance between the leader and the follower (see Fig. 1), and ω is a tuning parameter that adjusts the weight of d.

We note that maximizing the reward is equivalent to minimizing the cost. When the distance between the leader and the follower is too far or too close, the first term of the *Cost* function is activated to maintain the distance in an appropriate range, corresponding to the separation and cohesion rules of flocking. On the other hand, the second term of the *Cost* function emphasizes the alignment rule. In other words, this term encourages the follower to align its heading with the leader if the follower is near to the boundary of the annulus around the leader (see Fig. 1).

Fig. 2 illustrates the relation between the distance and the reward. It shows that the maximal reward is obtained when $d_1 < \rho < d_2$, which is consistent with our settings shown in Fig. 1.

4. CACER algorithm for fixed-wing UAVs flocking

In this section, we propose a DRL-based algorithm to solve the flocking problem in continuous state and action spaces. We develop an algorithm called Continuous Actor–Critic with Experience Replay (CACER, Algorithm 1) based on the original CACLA algorithm [34,35]. Similar to DDPG, CACER also uses DNNs as function approximators to represent the policy and the value function.

As an actor–critic algorithm, CACER has both an actor and a critic. Instead of using a single-layer feed-forward neural network to represent the actor as in CACLA, CACER utilizes a Multi-Layer Perceptron (MLP, see Fig. 3) to map the state space to the action space, i.e., $Act^* : S \rightarrow A$, where $Act^*\left(s|\theta^A\right)$ denotes the optimal action in the state s, and θ^A represents the parameter of the actor network. Additionally, the critic of CACER also uses a MLP (see Fig. 3) to approximate the state value function $V\left(s|\theta^V\right) : S \rightarrow \mathbb{R}$, where θ^V denotes the parameter of the critic network. The update of this parameter is derived from (6):

$$\theta^V = \theta^V + \beta\delta_t \frac{\partial V\left(s_t|\theta^V\right)}{\partial\theta^V}. \tag{20}$$

Different from the update of the critic, the actor is updated only when the sign of the TD-error is positive which indicates the current state is better than expected. Then, the probability of selecting the corresponding action should be increased. Therefore, according to (8), the update rule of the actor network is as follows:

$$\theta^A = \begin{cases} \theta^A + \alpha\Delta_a \frac{\partial Act\left(s_t|\theta^A\right)}{\partial\theta^A} & \text{if } \delta_t > 0 \\ \theta^A & \text{otherwise} \end{cases}, \tag{21}$$

where

$$\Delta_a = a_t - Act\left(s_t|\theta^A\right). \tag{22}$$

As described in (21), the parameter θ^A of the actor network is updated only when the executed action a_t differs from the output of the actor network $Act\left(s_t|\theta^A\right)$. In other words, the actor of CACER is only updated in exploratory trials. We choose the Gaussian exploration strategy for selecting exploratory actions. Specifically, the action is randomly selected from the Gaussian distribution $G(x, \mu, \sigma)$ centered at the current output of the actor networks $Act\left(s_t|\theta^A\right)$:

$$G\left(x, Act\left(s_t|\theta^A\right), \sigma\right) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\left(x - Act_t\left(s_t|\theta^A\right)\right)^2/2\sigma^2}, \tag{23}$$

where σ is the Gaussian exploration parameter. In this paper, the action space is two-dimensional, thus the exploration parameter for each dimension is selected separately.

The network structure for CACER is illustrated in Fig. 3. Both the actor and the critic use the similar structure of MLP that consists of 3 fully-connected neural network layers (Dense layers) with 512, 512 and 256 nodes, respectively. Each Dense layer is followed by a Rectified Linear Unit (ReLU) [41] activation function layer. After the 3 Dense layers, the actor transfers the input state to the roll and the velocity commands of the UAV. To guarantee the output of the actor is within the range of $[-1, +1]$, the output layer of the critic is activated by a hyperbolic tangent (tanh) activation function. As the roll action space is $[-10, +10]$, the output of the roll stream should be linearly magnified by multiplying with a constant 10. For the critic, it also uses 3 Dense layers to estimate the state value. Accordingly, its output layer uses a linear activation function.

Similar to other DRL algorithms such as DQN and DDPG, we also use the experience replay technique to increase training efficiency. Instead of only using the current experience as in the original CACLA algorithm, CACER randomly samples a mini-batch from the experience buffer to update its network parameters. This buffer is used to store state transition samples collected at each time step. If the buffer is full, then the oldest experience will be replaced by the latest one.

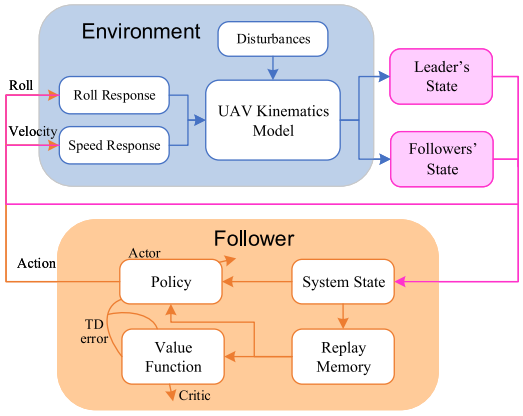


Fig. 4. The interaction between a follower and a leader.

As mentioned in Section 3, the control policy of the leader is determined by itself. Accordingly, the leader can be regarded as part of the environment with respect to the follower. Fig. 4 illustrates the interaction between a follower (i.e., the DRL agent) and a leader.

We summarize the workflow of the CACER algorithm in Algorithm 1. In every training episode, both the follower and the leader are randomly initialized. Then, the follower receives the leader's state, including the position, the pose, and the velocity, and then combine them with its own state to create the system state s (Line 3). Taking s as the input, the actor outputs an optimal action. Then, the agent selects an action a (i.e., roll command and velocity command) according to the Gaussian exploration strategy (Line 5). After applying this action, the follower's state is updated (Line 7). Consequently, the immediate reward r is given (Line 8) and the system state of the next time step s' is obtained (Line 10). Every state transition tuple (s, a, r, s') is saved to the experience buffer (Line 11). At every time step, a mini-batch updating is conducted (Line 14–15) by sampling from the experience buffer uniformly (Line 12).

Algorithm 1 CACER-Flocking

Input: N_s — maximum time steps; N_b — training batch size

1: Empty replay memory D with capacity N
2: **repeat**
3: Initialize $s \leftarrow \left(\xi_l, \xi_f, \phi_{ref}^l, v_{ref}^l\right)$ randomly
4: **for** $t = 1$ to N_s **do**
5: Select a (i.e., a_r and a_v) using (23)
6: Calculate ϕ_{ref}^f and v_{ref}^f using (17) and (18)
7: $\xi_l' \leftarrow$ UAV Kinematics $\left(\xi_l, \phi_{ref}^l, v_{ref}^l\right)$; $\xi_f' \leftarrow$ UAV Kinematics $\left(\xi_f, \phi_{ref}^f, v_{ref}^f\right)$
8: Calculate r using (19)
9: Choose $\phi_{ref}^{\prime l}$ and $v_{ref}^{\prime l}$ randomly
10: $s' \leftarrow \left(\xi_l', \xi_f', \phi_{ref}^{\prime l}, v_{ref}^{\prime l}\right)$
11: Store (s, a, r, s') in D, and replace the oldest experience if $\|D\| > N$
12: Sample a minibatch of N_b tuples (s_j, a_j, r_j, s_{j+1}) from D randomly
13: Calculate the TD error δ^j using (7)
14: Update θ_j^A using (21) if $\delta_j > 0$
15: Update θ_j^V using (20)
16: $\left(\xi_l, \xi_f, \phi_{ref}^l, v_{ref}^l\right) \leftarrow \left(\xi_l', \xi_f', \phi_{ref}^{\prime l}, v_{ref}^{\prime l}\right)$
17: **end for**
18: **until** desired number of training episodes

5. Experiments and results

In this section, we evaluate the proposed CACER algorithm in both numerical simulation and semi-physical simulation. In the numerical simulation, we first trained the followers using the stochastic kinematics model discussed in Section 3.1, and then we tested the learned policy by comparing it with the result of DDPG as well as the greedy policy and the imitation policy. Then, we transferred the learned policy in numerical simulation to semi-physical simulation for further evaluation.

5.1. Experiment settings

In our experiments, we used TensorFlow[1] and Keras[2] to implement the actor network and the critic network. The network parameters were both updated using the Adam optimizer [42] with the MSE (Mean Square Error) loss function and the batch size $N_b = 32$. The desired number of training episodes was set to 50 000, and each episode had a maximal number of time steps $N_s = 50$, i.e., 50 s. We note that 200 episodes of pre-training were carried out to collect samples for the experience replay buffer before the actual training. In other words, the network parameters were updated only when the number of samples in the buffer was more than $200 \times 50 = 10\,000$. During the training process, the exploration parameter σ was annealed exponential from the initial values 0.5 to the minimum values 0.05. In other words, this parameter was multiplied by 0.99 every 100 episodes. We list other parameters in Table 1 [5,32].

Table 1
Parameter settings.

Name	Value	Name	Value
d_1	40	d_2	65
ω	0.05	α_g	9.8
$\overline{\omega}_n$	6.3	σ_ω	0.5
$\overline{\zeta}$	0.5561	σ_ζ	0.01
r_{bd}	30°	σ_v	0.5
v_{max}	18	v_{min}	12
$\overline{\eta}_x$	0	σ_x	0.5
$\overline{\eta}_y$	0	σ_y	0.5
$\overline{\eta}_\psi$	0	σ_ψ	0.2
α	0.001	β	0.0001
N	100 000	γ	0.95

5.2. Numerical simulation

In the sequel, we use the averaged reward as the metrics for evaluating the performance of algorithms:

$$G_{\text{Ave}} = \frac{1}{N_e N_s} \sum_{n=1}^{N_e} \sum_{t=1}^{N_s} r_t^n, \tag{24}$$

where G_{Ave} denotes the average reward within a certain number of N_e episodes and r_t^n represents the immediate reward determined by (19).

In the training process, we calculated G_{Ave} obtained by the follower agent using CACER every 100 episodes, i.e., $N_e = 100$. Additionally, we compared CACER with the original CACLA algorithm and a state-of-the-art DRL algorithm DDPG (see Fig. 5). For a fair comparison, the above algorithms used the same network

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

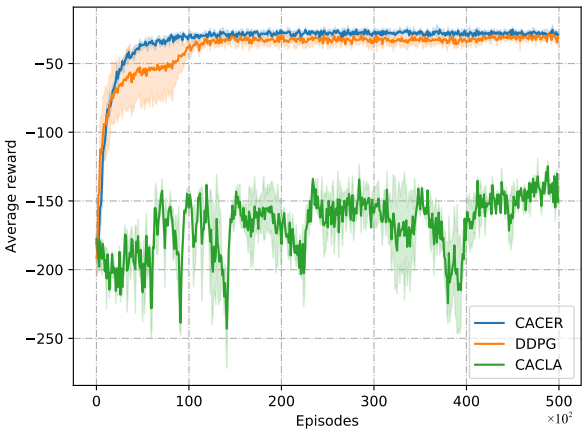


Fig. 5. Averaged reward comparison of CACER, DDPG, and CACLA. We note that $N_e = 100$, i.e., we calculated G_{Ave} every 100 episodes.

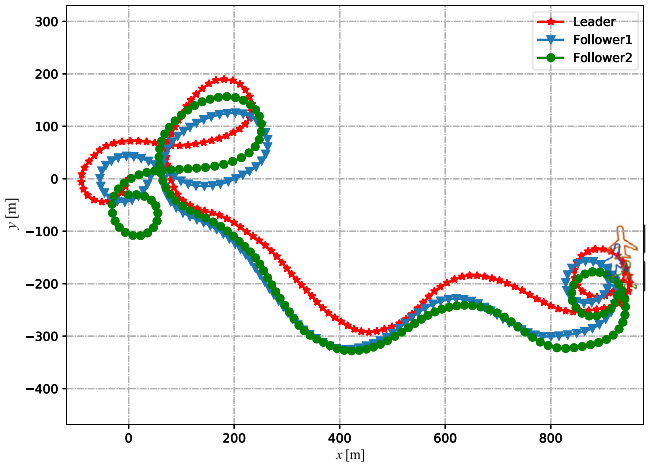


Fig. 6. The trajectory results in the numerical simulation.

structure (see Fig. 3) and the same parameters (see Table 1) except for CACLA. We note that CACLA only used a single-layer feed-forward neural network with 64 nodes. Additional parameters used by DDPG were the same with [26].

As shown in Fig. 5, CACLA did not perform as well as CACER and DDPG, which demonstrated the advantage of DNNs as function approximators and the better training efficiency of the experience replay technique. Besides, the averaged reward of CACER and DDPG both increased rapidly in the early stage before around 2000 episodes. Compared with DDPG, the reward curve of CACER grew faster from 2000 episodes and gradually became stable. After about 12 000 episodes, their reward curves were both stable, but CACER obtained a higher reward than DDPG. The above results illustrated that CACER learned faster than DDPG and its reward curve became stable earlier than DDPG. In other words, CACER slightly outperformed DDPG in terms of learning speed and stability.

Then, we tested the learned CACER policy in the flocking task lasting for 150 s, i.e., the maximum time step $N_s = 150$. The leader's roll commands and velocity commands were randomly generated, but the two followers adopted the learned CACER policy to select their roll and velocity commands. As shown in Fig. 6, the followers were able to keep up with the leader even if the leader changed its heading sharply in the beginning and at the end of the experiment.

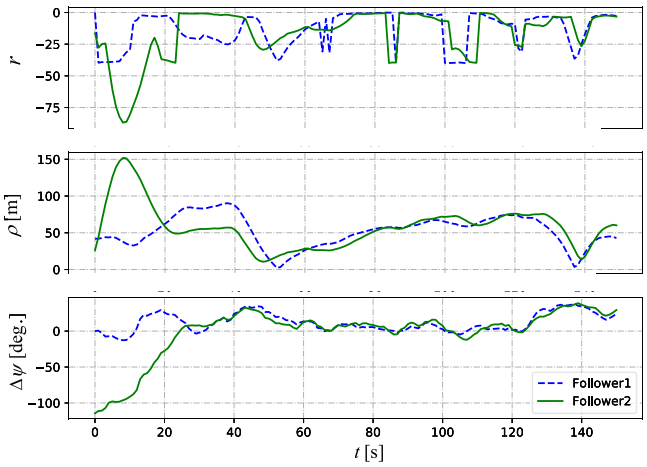


Fig. 7. The immediate reward (r), the distances between the leader and followers (ρ), and the differences in the heading ($\Delta\psi$) in the numerical simulation.

In addition, we also recorded the immediate reward (r), the distances between the leader and followers (ρ), and the differences in the heading ($\Delta\psi$), as shown in Fig. 7. Follower 1 performed better than Follower 2 because it started with better initial states than Follow 2, as indicated by ρ and $\Delta\psi$ in Fig. 7. In other words, the heading of Follower 1 was the same as the leader, but the heading of Follower 2 was almost in the opposite direction of the leader. Nevertheless, after about 30 s, Follower 2 successfully reduced the distance to the leader to the desired range and aligned its heading with the leader. From then on, both Follower 1 and Follower 2 achieved a satisfying performance. This experiment showed that the learned CACER policy was able to deal with different initial conditions.

We also compared CACER with several commonly used baselines for further quantitative evaluation. In addition to DDPG and CACLA, we selected a state-of-the-art discrete DRL algorithm Double DQN (DDQN) [25] as well as the following two policies to compare with:

- **Greedy**: At each time step, the follower agent used the UAV kinematics model to choose sequential actions that resulted in maximum immediate reward by iterating all possible actions. We had to discretize the action space for this purpose. In other words, the agent selected its roll command from $\{-10, 0, +10\}$ and velocity command from $\{-1, 0, +1\}$ corresponding to the highest immediate reward. Denote by F_{Greedy} the follower who employed the greedy policy.
- **Imitation**: When the follower's initial state was the same or Similar to the leader, it seemed to be a good policy that the follower imitated the leader's actions. In other words, the follower selected the same roll angle and velocity setpoints with the leader. Denote by $F_{\text{Imitation}}$ the imitation follower.

We defined 9 discrete actions for DDQN. Specifically, the DDQN agent selected the roll command from $\{-10, 0, +10\}$ and the velocity command from $\{-1, 0, +1\}$. It should be noted that DDQN used the same parameters as CACER, and used the same network structure as the actor/critic of CACER up to the last output layer. Additional parameters used by DDQN were the same with [27]. Denote by F_{DDQN}, F_{DDPG}, and F_{CACER} the DDQN follower, the DDPG follower, and the CACER follower, respectively.

Aside from the averaged reward, we also chose the *failure rate* as a metric for quantitative comparison. We considered that the following task failed if the distance between the follower to the leader was more than 150 m. Thus, the *failure rate* meant

Table 2
Comparison results of CACER, DDPG, DDQN, CACLA, greedy, and imitation policies.

Method	$\Delta\psi_0 \in [-30°, 30°]$			$\Delta\psi_0 \in [-60°, 60°]$			$\Delta\psi_0 \in [-90°, 90°]$		
	G_{Ave}	Var	$F_r(\%)$	G_{Ave}	Var	$F_r(\%)$	G_{Ave}	Var	$F_r(\%)$
CACER	**−11.29**	**8.42**	**0.00**	**−11.51**	**8.83**	**0.00**	**−12.03**	**10.99**	**0.02**
DDPG	−14.25	11.94	0.05	−14.29	12.2	0.05	−14.57	11.41	0.04
DDQN	−18.77	59.75	1.35	−18.84	57.37	1.44	−20.10	88.65	1.94
CACLA	−264.14	10650.60	70.50	−269.28	9205.46	72.82	−287.47	11622.01	75.55
Greedy	−83.05	905.67	28.79	−82.96	885.55	28.82	−83.21	793.11	28.98
Imitation	−50.77	1579.75	11.78	−56.12	1956.81	14.06	−53.94	1924.96	12.68

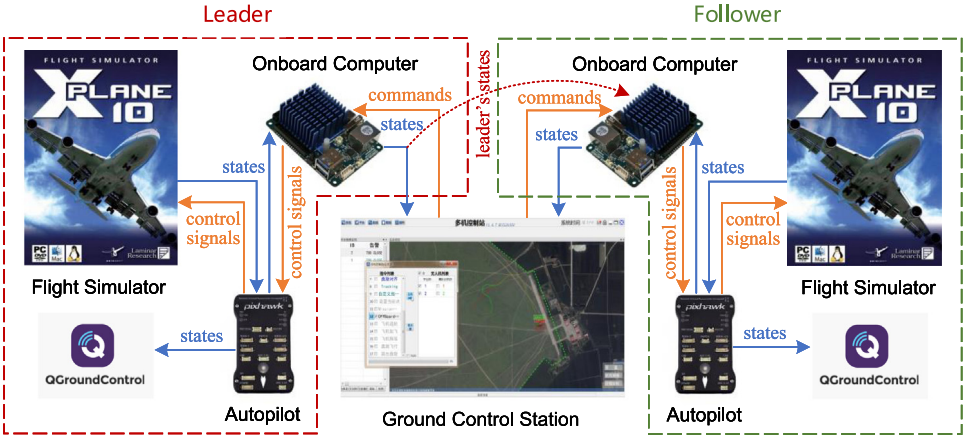


Fig. 8. The high-fidelity semi-physical simulation system.

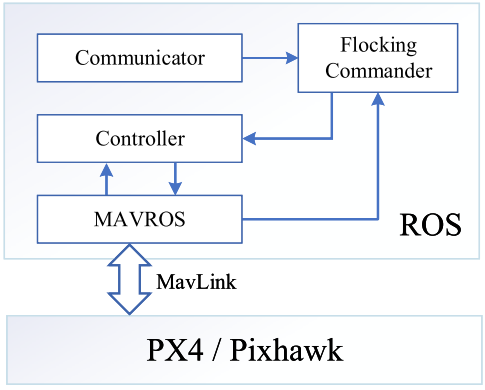


Fig. 9. The software architecture for flocking control.

the percentage of being too far (more than 150 m) with the leader during the testing duration. Denote by F_r this metric. We averaged the results of 500 episodes (i.e., $N_e = 500$) to compare the performance of the learned CACER policy with the five other policies. In one episode, the maximum time step N_s was set to 200. In this experiment, the leader was followed by six followers F_{CACER}, F_{DDPG}, F_{DDQN}, F_{CACLA}, F_{Greedy}, and $F_{\text{Imitation}}$, respectively. In each episode, the leader's initial state was randomly generated as well as its control commands. We note that the initial state of $F_{\text{Imitation}}$ was the same as the leader (to meet the applicable conditions of imitation policy mentioned above), while the initial states of other followers were the same and randomly generated. This meant that the task was more difficult for the five followers than $F_{\text{Imitation}}$, and it was a fair comparison for F_{CACER}, F_{DDPG}, F_{DDQN}, F_{CACLA}, and F_{Greedy}. The comparison results were provided in Table 2.

In Table 2, $\Delta\psi_0$ denoted the difference in heading between the leader and the follower at the initial time step; Var denoted the variance corresponding to the averaged reward G_{Ave}; F_r denoted the failure rate. As can be seen, the three DRL-based methods, i.e., CACER, DDPG and DDQN, achieved similar and higher averaged reward than CACLA, the imitation policy, and the greedy policy. Meanwhile, their results were also more stable with a much lower variance. This demonstrated the effectiveness of DRL-based methods for the UAV flocking problem. Furthermore, the initial heading difference $\Delta\psi_0$ did not have much influence on the performance of CACER, DDPG, and DDQN. With the increase of $\Delta\psi_0$, the averaged reward of the three algorithms both decreased slightly, and their failure rate both increased slightly. Compared with DDPG and DDQN, CACER always obtained a higher averaged reward, and its corresponding variance and failure rate were also smaller than that of DDPG and DDQN. This result demonstrated that the proposed CACER algorithm outperformed the state-of-the-art DDPG algorithm and DDQN algorithm.

5.3. Semi-physical simulation

A high-fidelity semi-physical simulation system was developed to further evaluate the performance of CACER towards real-world flocking. We designed a Hardware-In-Loop (HIL) experiment for this purpose.

As illustrated in Fig. 8, the semi-physical simulation system [33,43] for a UAV consisted of four main parts: An X-Plane 10 Flight Simulator[3], a Pixhawk[4] autopilot, an NVIDIA Jetson TX1 onboard computer, and a SuperStation ground control station. The commercial flight simulation software X-Plane 10 was responsible for simulating complex environmental conditions such as weather changes and wind disturbance. We chose the HilStar17F model in X-Plane for both the leader and the follower. In addition, the leader and the follower were both equipped with an NVIDIA Jetson TX1 as the onboard computer which was connected through an RJ45 patch cable. We note that the leader

[3] https://www.x-plane.com/manuals/desktop/.
[4] http://pixhawk.org/.

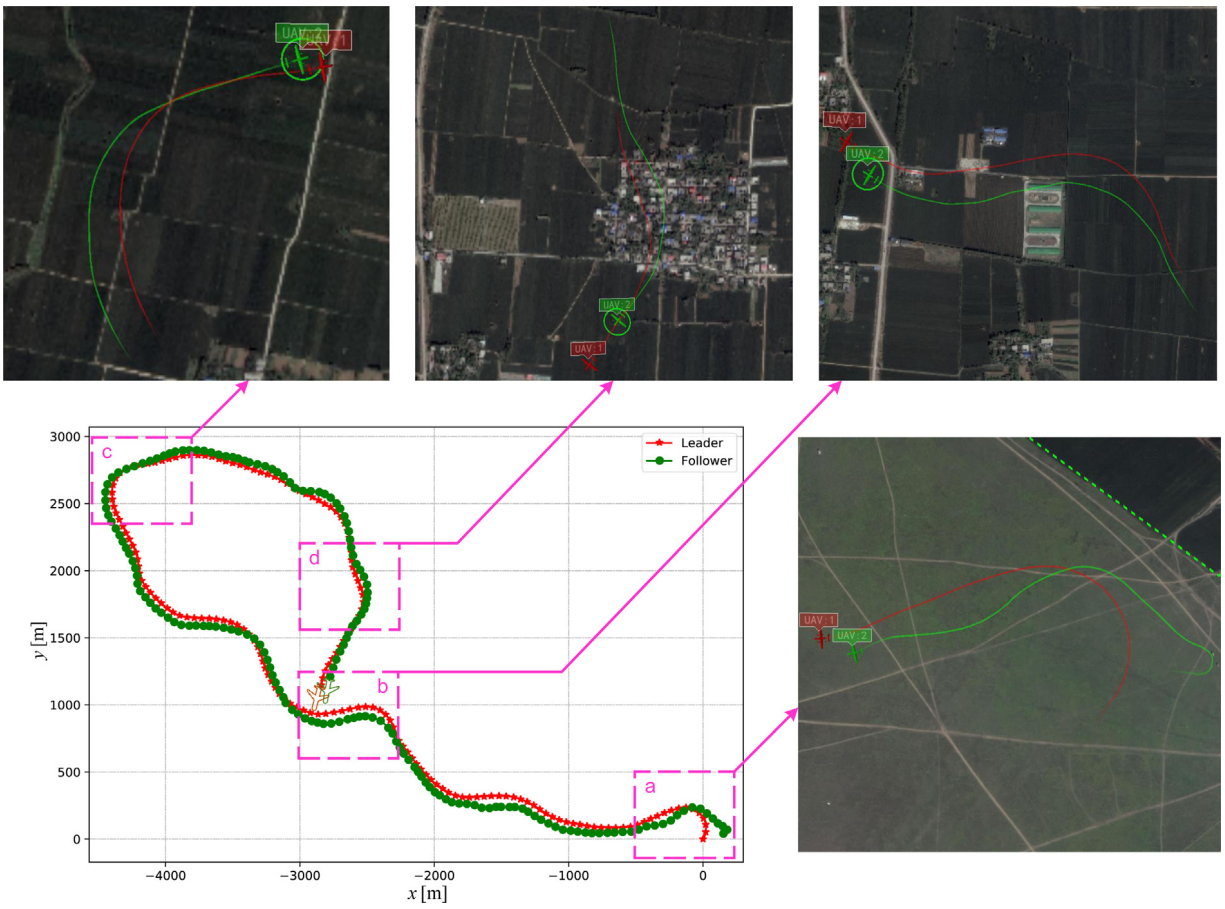


Fig. 10. The trajectory results in the semi-physical simulation.

and the follower shared one ground control station SuperStation which could simultaneously control the leader and the follower by switching control modes and planning paths.

As shown in Fig. 9, we selected the PX4[5] flight stack as the firmware for the Pixhawk autopilot, because it provided a robust and open source fight control solution for UAVs. In addition, we ran the Ubuntu 14.04 LTS and Robot Operating System (ROS)[6] on the TX1 onboard computer. All processing is done on this platform. The PX4/Pixhawk module communicated with the TX1 module through the MavLink protocol. We used the following ROS nodes to implement the flocking control: Communicator node for receiving leader's state via UDP socket, Flocking Commander node for decision-making based on the learned CACER flocking policy, Controller node for closed-loop control with a PID controller, and MAVROS node for communicating with PX4 via MavLink.

In this experiment, a follower employed the learned CACER policy to follow a leader that randomly selected its roll and velocity commands every one second. 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. At each time step, the follower decided its roll and velocity commands according to the following steps:

Step 1: Received the leader's state (i.e., position, pose, and airspeed) from the Communicator node, and then combined it with its own state acquired from the MAVROS node to obtain the system state.

Step 2: Loaded the actor network of CACER with the trained parameters in the previous numerical simulation experiment.

Step 3: Generated the roll command and the velocity command according to the output of the actor network, and then calculated its roll angle setpoint and velocity setpoint.

Step 4: Published the roll angle setpoint and the velocity setpoint to the Controller node that carried out the closed-loop control with a PID controller.

The semi-physical simulation experiment lasted for 680 time steps (seconds). After that, the RETURN mode was turned on and the experiment ended.

Fig. 10 shows the trajectory results in the semi-physical simulation, and Fig. 11 shows the results of the immediate reward (r), the distance between the leader and the follower (ρ), and the difference in the heading ($\Delta\psi$). As can be seen, the follower was always able to follow the leader. In the early time steps, the distance between the leader and the follower reached up to 157 m, and the difference in the heading was also beyond $-60°$. After about 54 s, the distance dropped below 65 meters for the first time. From then on, the distance was maintained between 50 and 100 meters most of the time, and the difference in heading was kept in the range of $[-10°, 10°]$. Even after a 10-minute flight test, the follower was still able to follow the leader with a relatively good performance. We note that the flocking control policy adopted by the follower in this semi-physical simulation is the CACER policy learned in the previous numerical simulation, without any revision. The above results verify that the learned

[5] https://px4.io/.

[6] http://www.ros.org/.

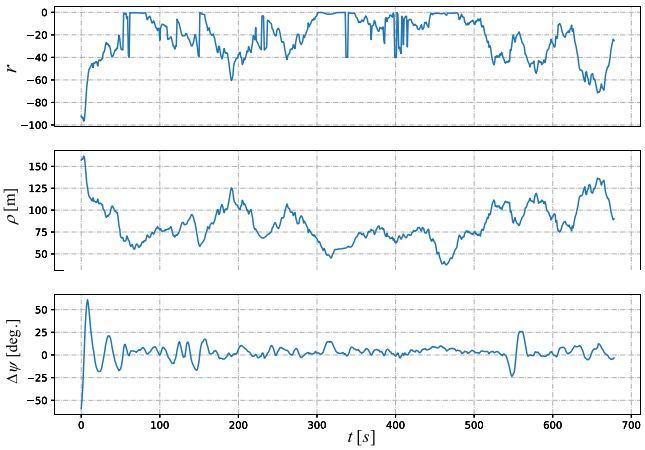


Fig. 11. The immediate reward (r), the distance between the leader and the follower (ρ), and the difference in the heading ($\Delta\psi$) in the semi-physical simulation.

CACER flocking policy can deal with new situations without any parameter finetuning.

6. Conclusion

In this paper, we have proposed a novel deep reinforcement learning algorithm CACER for fixed-wing UAVs flocking. We have solved the leader–follower problem in continuous state and action spaces which is more challenging than the discrete case in the literature [5,31,32]. Specifically, CACER not only uses deep networks to represent both the actor and the critic based on the original CACLA algorithm, but also uses the experience replay technique to improve the training efficiency. After training with CACER, the followers are able to keep appropriate distances with the leader which has its own control policy unavailable to the followers. The numerical simulation results have demonstrated that CACER outperforms the state-of-the-art DDPG algorithm and DDQN algorithm, as well as the CACLA algorithm, the imitation policy, and the greedy policy. In addition, the learned policy in the numerical simulation can be directly transferred to the semi-physical simulation without any parameter tuning. In contrast to the previous work that assumed the followers fly at a fixed velocity and can only adjust the roll angle to follow the leader, we allow the followers to change the speed in addition to the roll angle. However, we have made the same assumption with the previous work that the UAVs fly at fixed different heights to simplify the collision problem. In future work, we will relax this assumption towards a swarm of fixed-wing UAVs flocking in real-world environments.

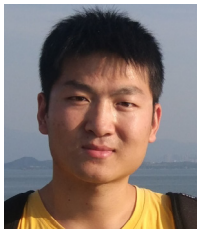
Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] A. Birk, B. Wiggerich, H. Bulow, M. Pfingsthorn, S. Schwertfeger, Safety, security, and rescue missions with an unmanned aerial vehicle (UAV), J. Intell. Robot. Syst. 64 (1) (2011) 57–76.
[2] N. Wen, X. Su, P. Ma, L. Zhao, Y. Zhang, Online UAV path planning in uncertain and hostile environments, Int. J. Mach. Learn. Cybern. 8 (2) (2017) 469–487.
[3] V. Kumar, Robot Quadrotors Perform James Bond Theme, University of Pennsylvania, General Robotics, Automation, Sensing and Perception (GRASP) Lab, 2012.
[4] N. Kshetri, D. Rojas-Torres, The 2018 Winter Olympics: A showcase of technological advancement, IT Prof. 2 (2018) 19–25.
[5] S.M. Hung, S.N. Givigi, A Q-learning approach to flocking with UAVs in a stochastic environment, IEEE Trans. Cybern. 47 (1) (2017) 186–197.
[6] C.W. Reynolds, Flocks, herds, and schools: A distributed behavioral model, Comput. Graph. 21 (4) (1987) 25–34.
[7] R. Olfati-Saber, Flocking for multi-agent dynamic systems: Algorithms and theory, IEEE Trans. Automat. Control 51 (3) (2006) 401–420.
[8] R. Olfati-Saber, R.M. Murray, Consensus problems in networks of agents with switching topology and time-delays, IEEE Trans. Automat. Control 49 (9) (2004) 1520–1533.
[9] R. Olfati-Saber, J.A. Fax, R.M. Murray, Consensus and cooperation in networked multi-agent systems, Proc. IEEE 95 (1) (2007) 215–233.
[10] M. Jafari, H. Xu, L.R.G. Carrillo, Brain emotional learning-based intelligent controller for flocking of multi-agent systems, in: Proceedings of American Control Conference, ACC, 2017, pp. 1996–2001.
[11] M. Jafari, H. Xu, A game theoretic based biologically-inspired distributed intelligent flocking control for multi-UAV systems with network imperfections, in: IEEE Symposium Series on Computational Intelligence, SSCI, 2018, pp. 1138–1144.
[12] M. Jafari, H. Xu, A biologically-inspired distributed fault tolerant flocking control for multi-agent system in presence of uncertain dynamics and unknown disturbance, Eng. Appl. Artif. Intell. 79 (2019) 1–12.
[13] H. Rezaee, F. Abdollahi, Mobile robots cooperative control and obstacle avoidance using potential field, in: Proceedings of IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM, 2011, pp. 61–66.
[14] H. Rezaee, F. Abdollahi, A decentralized cooperative control scheme with obstacle avoidance for a team of mobile robots, IEEE Trans. Ind. Electron. 61 (1) (2013) 347–354.
[15] H. Rezaee, F. Abdollahi, A cyclic pursuit framework for networked mobile agents based on vector field approach, J. Franklin Inst. 356 (2) (2019) 1113–1130.
[16] X. Wang, J. Qin, C. Yu, ISS method for coordination control of nonlinear dynamical agents under directed topology, IEEE Trans. Cybern. 44 (10) (2014) 1832–1845.
[17] W. Zhao, H. Chu, M. Zhang, T. Sun, L. Guo, Flocking control of fixed-wing UAVs with cooperative obstacle avoidance capability, IEEE Access 7 (2019) 17798–17808.
[18] P. Li, S. Xu, W. Chen, Z. Zhang, Adaptive finite-time flocking for uncertain nonlinear multi-agent systems with connectivity preservation, Neurocomputing 275 (2018) 1903–1910.
[19] H.X. Pham, H.M. La, D. Feil-Seifer, A. Nefian, Cooperative and distributed reinforcement learning of drones for field coverage, 2018, arXiv:1803.07250.
[20] K. Morihiro, T. Isokawa, H. Nishimura, N. Matsui, Characteristics of flocking behavior model by reinforcement learning scheme, in: Proceedings of SICE-ICASE International Joint Conference, 2006, pp. 4551–4556.
[21] H.M. La, R. Lim, W. Sheng, Multirobot cooperative learning for predator avoidance, IEEE Trans. Control Syst. Technol. 23 (1) (2015) 52–63.
[22] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, 1998.
[23] E.M.E. Mhamdi, R. Guerraoui, A. Maurer, V. Tempez, Learning to gather without communication, 2018, arXiv:1802.07834.
[24] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nature 581 (7549) (2015) 529–533.
[25] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double Q-learning, in: Proceedings of AAAI Conference on Artificial Intelligence, 2015, pp. 2094–2100.
[26] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, S. David, D. Wierstra, Continuous control with deep reinforcement learning, 2015, arXiv:1509.02971.
[27] C. Yan, X. Xiang, C. Wang, Towards real-time path planning through deep reinforcement learning for a UAV in dynamic environments, J. Intell. Robot. Syst. 98 (2019) 297–309.
[28] M. Duguleana, G. Mogan, Neural networks based reinforcement learning for mobile robots obstacle avoidance, Expert Syst. Appl. 62 (2016) 104–115.
[29] E. Bohn, E.M. Coates, S. Moe, T.A. Johansen, Deep reinforcement learning attitude control of fixed-wing UAVs using proximal policy optimization, in: Proceedings of International Conference on Unmanned Aircraft Systems, ICUAS, 2019, pp. 523–533.
[30] C. Wang, J. Wang, X. Zhang, A deep reinforcement learning approach to flocking and navigation of UAVs in large-scale complex environments, in: Proceedings of IEEE Global Conference on Signal and Information Processing, GlobalSIP, 2018, pp. 1228–1232.
[31] S.A.P. Quintero, G.E. Collins, J.P. Hespanha, Flocking with fixed-wing UAVs for distributed sensing: A stochastic optimal control approach, in: Proceedings of American Control Conference, 2013, pp. 2025–2031.

[32] S.M. Hung, S.N. Givigi, A. Noureldin, A Dyna-Q (λ) approach to flocking with fixed-wing UAVs in a stochastic environment, in: Proceedings of IEEE International Conference on Systems, Man, and Cybernetics, 2015, pp. 1918–1923.
[33] C. Wang, C. Yan, X. Xiang, H. Zhou, A continuous actor-critic reinforcement learning approach to flocking with fixed-wing UAVs, in: Proceedings of Asian Conference on Machine Learning, 2019, pp. 64–79.
[34] H. Van Hasselt, M.A. Wiering, Reinforcement learning in continuous action spaces, in: Proceedings of IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning, 2007, pp. 272–279.
[35] H. Van Hasselt, M.A. Wiering, Using continuous action spaces to solve discrete problems, in: Proceedings of International Joint Conference on Neural Networks, IJCNN, 2009, pp. 1149–1156.
[36] G. Leuenberger, M.A. Wiering, Actor-critic reinforcement learning with neural networks in continuous games, in: Proceedings of ICAART (2), 2018, pp. 53–60.
[37] A.O. Wiehe, N.S. Anso, M.M. Drugan, M.A. Wiering, Sampled policy gradient for learning to play the game Agar.io, 2018, arXiv:1809.05763.
[38] C. Wang, K.V. Hindriks, R. Babuska, Active learning of affordances for robot use of household objects, in: Proceedings of IEEE-RAS International Conference on Humanoid Robots, 2014, pp. 566–572.
[39] L.J. Lin, Self-improving reactive agents based on reinforcement learning, planning and teaching, Mach. Learn. 8 (3) (1992) 293–321.
[40] A. Faust, K. Oslund, O. Ramirez, A. Francis, L. Tapia, M. Fiser, J. Davidson, PRM-RL: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning, in: Proceedings of IEEE International Conference on Robotics and Automation, ICRA, 2018, pp. 5113–5120.
[41] V. Nair, G.E. Hinton, Rectified linear units improve restricted Boltzmann machines, in: Proceedings of International Conference on Machine Learning, ICML, 2010, pp. 807–814.
[42] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv:1412.6980.
[43] Z. Ma, C. Wang, Y. Niu, X. Wang, L. Shen, A saliency-based reinforcement learning approach for a UAV to avoid flying obstacles, Robot. Auton. Syst. 100 (2018) 108–118.



Chao Yan is currently pursuing his Ph.D. degree in the College of Intelligence Science and Technology at National University of Defense Technology (NUDT), Changsha, China. He received the B.S. degree (outstanding graduates) in the School of Information and Control Engineering from China University of Mining and Technology, Xuzhou, China, in 2017. In 2019, he got the M.S. degree in control science and engineering from NUDT. His research interests include the applications of reinforcement learning techniques for coordination control of UAVs.

Xiaojia Xiang is an associate professor in the College of Intelligence Science and Technology, National University of Defense Technology (NUDT), Changsha, China. He received the B.S., M.S., and Ph.D. degrees in automatic control from NUDT, in 2003, 2007, and 2016, respectively. He currently works in the field of mission planning, autonomous and cooperative control of unmanned systems.

Chang Wang is an assistant professor at the Institute of Unmanned Systems, National University of Defense Technology (NUDT), China. He holds his B.S. degree in mathematics from University of Science and Technology of China (USTC), M.Sc. degree in applied mathematics from NUDT, and Ph.D. degree in robot learning from Delft University of Technology, Netherlands. His research interests include reinforcement learning, developmental robotics, and humanrobot teamwork.