Complex & Intelligent Systems (2024) 10:6867–6883
https://doi.org/10.1007/s40747-024-01504-1

ORIGINAL ARTICLE

Multi-UAV pursuit-evasion gaming based on PSO-M3DDPG schemes

Yaozhong Zhang[1] · Meiyan Ding[1] · Jiandong Zhang[1] · Qiming Yang[1] · Guoqing Shi[1] · Meiqu Lu[2] · Frank Jiang[3]

Received: 21 October 2023 / Accepted: 15 May 2024 / Published online: 24 June 2024

Abstract
The sample data for reinforcement learning algorithms often exhibit sparsity and instability, making the training results susceptible to falling into local optima. Mini-Max-Multi-agent Deep Deterministic Policy Gradient (M3DDPG) algorithm is a multi-agent reinforcement learning algorithm, which introduces the minimax theorem into Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithm. It also has unstable convergence caused by sparse sample data and randomization. However, the Particle Swarm Optimisation (PSO) algorithm, unlike traditional reinforcement learning methods, involves the construction of independent populations of policy networks to generate sample data, followed by training the reinforcement learning algorithm. PSO optimizes and updates the policy population based on a fitness function, aiming to enhance the efficiency and convergence speed of the algorithm in learning from the sample data. In order to address the multi-agent pursuit-evasion problem, we propose the PSO-M3DDPG algorithm, which combines the PSO algorithm with the M3DDPG algorithm. Through experimental simulations, the improved algorithm demonstrates superior training results and faster convergence speeds, thus validating its effectiveness.

Keywords Pursuit-evasion game · Particle swarm optimization algorithm · Reinforcement learning · M3DDPG (mini-max-multi-agent deep deterministic policy gradient)

Introduction

In the context of aerial military competition, the pursuit-evasion problem of Unmanned Aerial Vehicles (UAVs) serves as a typical example. It can be described as a reciprocal confrontation between two aircrafts based on some form of conflicting interests. Researching the optimal solutions for this pursuit-evasion problem involving aircraft is of paramount practical significance. The pursuit-evasion problem, as a typical class of Differential Games (DG), is widely present in the natural world or in military domains [1].

Game theory is a theoretical and methodological framework that models real-world situations of conflict, competition, and cooperation as mathematical models. Differential games are an important kind of game theory. It refers to participants who use differential equations to describe the phenomena or rules of the game while playing the game [2]. Differential games originally originated from military problems; however, in recent years they have been applied not only in military research, but also extensively in various fields such as life and economics [3]. Differential games are a method that combines game theory with modern control theory. It has evolved from unilateral optimal control to bilateral (or multilateral) optimal control and from static games to dynamic conflicts. The application areas of this method have become increasingly extensive. Military problems often involve complex non-linear models rather than linear control problems, such as the fixed-time prescribed performance trajectory tracking control for the unmanned surface vehicle with unknown dynamics and disturbances [4], naval combat, satellite interception, and missile defence [5]. Differential game theory can provide effective modelling and analysis for both unilateral and multilateral conflict scenarios. It can also be solved using control methods such as optimal control, leading to the determination of optimal control strategies for all parties involved. Therefore, differential game problems have significant theoretical research value and promising application prospects.

✉ Frank Jiang
frank.Jiang@uts.edu.au

[1] Northern Polytechnical University, Xi'an, China

[2] Guangxi Minzu University, Nanning, China

[3] UTS: University of Technology Sydney, Ultimo, NSW, Australia

The pursuit-evasion game in aerial combat, as a typical instance of differential games, can be considered a multi-agent dynamic game system because the objectives of both sides conflict with each other. Therefore, the choice of optimal control strategies for both aircrafts depends on the respective interests of the pursuer and the evader [6]. Reinforcement Learning (RL), as an important component of machine learning and a hot topic in current research, is an intelligent autonomous learning method. It does not require expert signals or strict mathematical models. Instead, it learns by interacting with the environment through a "trial-and-error" approach. RL continuously tries different behavioral strategies and improves them, adapting to dynamic and unknown environments. As its applications continue to expand, the ability of reinforcement learning to address the performance of multidimensional and complex nonlinear systems has gained increasing attention from researchers and leaders in various fields. Reinforcement learning can adaptively find optimal control methods for complex nonlinear systems [7]. It does not rely on precise mathematical models, is computationally straightforward, and requires relatively little training data. Based on the characteristics of reinforcement learning mentioned above, the control strategies obtained can comprehensively consider interactions between multiple agents and the impact of interactions between agents and the environment. This enables the implementation of adversarial or cooperative behaviors among multiple agents, providing rational, reliable, and dynamic strategic support to multiple agents. Therefore, there is great potential for applications in the field of differential games.

To harness the strengths of both game theory and reinforcement learning methods, some scholars have proposed intelligent agent adversarial algorithms based on reinforcement learning, incorporating RL into the process of modelling adversarial games. Reference [8] applied the Deep Deterministic Policy Gradient (DDPG) algorithm to multi-agent environments and proposed the MADDPG algorithm, which employs a method of decentralized action execution and centralized training of policies. This algorithm exhibits good stability and addresses the issue of high variance in the policy gradient. It enables groups of agents to develop cooperative strategies on both the physical and informational levels in environments characterized by both cooperation and competition. Reference [9] introduces a cooperative learning model called the joint action learner and demonstrates its effectiveness through experiments. Reference [10] uses G2ANet to model the interactions between two intelligent agents. It assesses whether there is interaction between the two agents, and, if so, further evaluates the importance of this type of interaction on the agents' strategies to accelerate the learning convergence speed. Reference [11] introduces mean field game reinforcement learning, which transforms multi-agent problems into problems involving two adjacent agents and replaces the influence of all other agents within the range with an average value. This approach addresses the issue of dimensionality explosion caused by a large number of agents in large-scale problems. Furthermore, there are many different approaches to solving multi-agent RL problems, such as ES-Q(Q learning based on experience sharing) [12], Pareto-Q(Pareto-Q learning) [13] algorithms, and others, which offer diverse perspectives and methods.

The M3DDPG algorithm [14] incorporates the minimax theorem from game theory into the MADDPG algorithm, enhancing the robustness of the algorithm and achieving good results in multiagent environments. However, the M3DDPG algorithm still faces problems of sparse sample data and instability in convergence caused by randomisation. In response to the important characteristics of intelligent agents that are difficult to adapt to complex dynamic environments and perceive the environment, this paper introduces the Maximum Minimum Multi Agent Deep Deterministic Policy Gradient (M3DDPG) algorithm. To improve the learning efficiency and convergence speed of the algorithm, particle swarm optimization is introduced to search and optimize the experience sample set of M3DDPG algorithm to a certain extent, and good training samples are obtained.

In summary, we use the M3DDPG algorithm as a foundation for improvement and employ the Multi-Agent Adversarial Learning (MAAL) approach to address the issue of excessive computation in continuous action spaces. The particle swarm optimisation (PSO) algorithm is introduced to optimise and update the set of sample experiences. As a result, this study proposed an enhanced PSO-M3DDPG algorithm. The main contributions are as follows.

- Using the MAAL approach to construct local linear functions to approximate the nonlinear state value function, employing gradient descent methods to approximate the objective instead of inner-loop minimisation methods, reducing computational complexity.
- Introducing the PSO algorithm to optimise the sample data set in order for parameter optimisation, addressing issues related to local optima and unstable convergence.

Problem description and modelling

Modelling the multi-agent pursuit-evasion task for UAVs, utilising the PSO-M3DDPG algorithm as the decision-maker unit for the chasing UAV cluster. These UAVs cooperate with each other to effectively perform the task of pursuing multiple escaping UAVs in a complex battlefield environment.

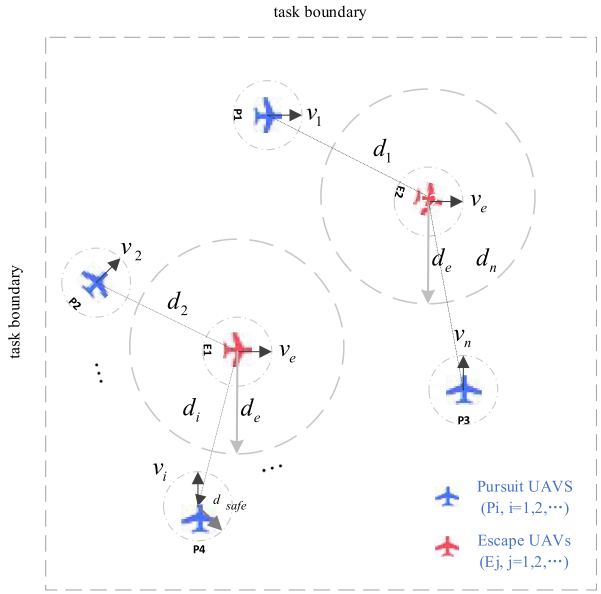

Fig. 1 The four-to-two pursuit-evasion game of UAVs

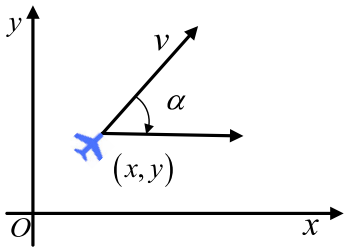

Fig. 2 UAV motion state diagram

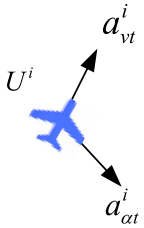

Fig. 3 Acceleration control of unmanned aerial vehicles

Battlefield environment

This study builds up a two-dimensional continuous battlefield as the task environment for the many-to-many pursuit and escape problem of UAVs. The length of the battlefield is L and the width is W. Setting the task scenario, $n(n > m)$ pursuit UAVs hunt $m(m > 1)$ escaping UAVs. $P = \{P_1, P_2, \cdots, P_n\}$ represents the collection of n pursuit UAVs, $E = \{E_1, E_2, \cdots E_n\}$ represents m escaping UAVs. At the beginning of the mission, the initial positions and velocities of both pursuing UAVs and escaping UAVs are randomly initialized. We set d_E as the radius of the safety zone established around escaping UAVs. If a pursuing UAV enters this zone, it successfully completes its tracking mission. When a pursuing UAV or an escaping UAV flies out of the battlefield boundary, the mission is considered a failure. The Fig. 1 illustrates a conceptual representation of a four-to-two pursuit-evasion UAV mission.

Motion model

The UAV is simplified to the mass-point model, and its motion state is determined by its position and velocity, as depicted in Fig. 2.

The instantaneous status information Q_t^i of UAVs i at the current moment t is represented as:

$$Q_t^i = \left[x_t^i,\ y_t^i,\ v_t^i,\ \alpha_t^i\right]^{\mathrm{T}} \tag{1}$$

In the equations: x_t^i and y_t^i are the position coordinates of the UAV i at time t; v_t^i is the speed magnitude of the UAV i at time t; α_t^i is the angle between the speed direction of the UAV i at time t and the positive X axis direction, known as the heading angle, defined as positive when rotating counterclockwise from the positive X axis direction.

By utilizing linear acceleration a_{vt}^i and angular acceleration $a_{\alpha t}^i$ to control the speed and direction of the UAV, maneuverable flight can be achieved, as shown in Fig. 3.

The instantaneous status information of the UAV i at the moment $t + 1$ is as follows:

$$\begin{cases} v_{t+1}^i = v_t^i + a_{vt}^i \cdot \Delta t \\ \alpha_{t+1}^i = \alpha_t^i + a_{\alpha t}^i \cdot \Delta t \\ x_{t+1}^i = x_t^i + v_{t+1}^i \cdot \cos\alpha_{t+1}^i \cdot \Delta t \\ y_{t+1}^i = y_t^i + v_{t+1}^i \cdot \sin\alpha_{t+1}^i \cdot \Delta t \end{cases} \tag{2}$$

$$Q_{t+1}^i = \left[x_{t+1}^i,\ y_{t+1}^i,\ v_{t+1}^i,\ \alpha_{t+1}^i\right]^{\mathrm{T}} \tag{3}$$

In the formula: Δt is the simulation step size.

Task allocation model

The task allocation model for the many-to-many pursuit-evasion game involves modeling the task assignment for the UAVs of both pursuing parties. The task assignment is achieved using the Apollo-Apollonius circle design advantage function.

The positions of the pursuing UAV i is denoted as $\left(x_{P_i}, y_{P_i}\right)$, and the positions of the escaping UAV j is represented $\left(x_{E_j}, y_{E_j}\right)$. The speed ratio between them is given as $\frac{v_{E_j}}{v_{P_i}} = k_{ij} < 1$. With this information, the coordinates of the Apollonius circle can be calculated as $\left(\frac{x_{E_j} - x_{P_i} k_{ij}^2}{1 - k_{ij}^2},\right.$

$\frac{y_{E_j}-y_{P_i}k_{ij}^2}{1-k_{ij}^2}\Bigg)$. The radius of the Apollonius circle can be determined as $\left(\frac{k_{ij}\sqrt{\left(x_{E_j}-x_{P_i}\right)^2+\left(y_{E_j}-y_{P_i}\right)^2}}{1-k_{ij}^2}\right)$.

The advantage function can be defined as follows:

$$X_{ij}(x)=\frac{x_{E_j}-k_{ij}^2x_{P_i}-k_{ij}\sqrt{(x_{E_j}-x_{P_i})^2+(y_{E_j}-y_{P_i})^2}}{1-k_{ij}^2} \tag{4}$$

The task allocation process can be described as n pursuit UAVs completing the pursuit of m escaping UAVs. At any given time, only one pursuit UAV can pursue a single escaping UAV. Each escaping UAV is pursued by at least one pursuit UAV. Task allocation is carried out during the initial stage, and remains unchanged until the end of the mission. The following assumptions are made:

$$a_{ij}=\begin{cases}0 \text{ The i-th pursuit drone did not perform the j-th task}\\ 1 \text{ The i-th pursuit drone was assigned to perform the j-th task}\end{cases} \tag{5}$$

Assign tasks to the pursuit and escape UAVs of both sides, as shown below, with a formula of 0–1 planning (Fig. 4):

$$s.t.\begin{cases}\sum_{i=1}^{n}a_{ij}=1 \quad j=1,2\cdots,m\\ \sum_{j=1}^{m}a_{ij}=1 \quad i=1,2,\cdots,n\\ \sum_{i=1}^{n}\sum_{j=1}^{m}a_{ij}=m=n\\ a_{ij}=0,1\end{cases} \tag{6}$$

The overall objective function is as follows:

$$V(x)=\sum_{i=1}^{n}\sum_{j=1}^{m}a_{ij}X_{ij} \tag{7}$$

The optimal task allocation is as follows:

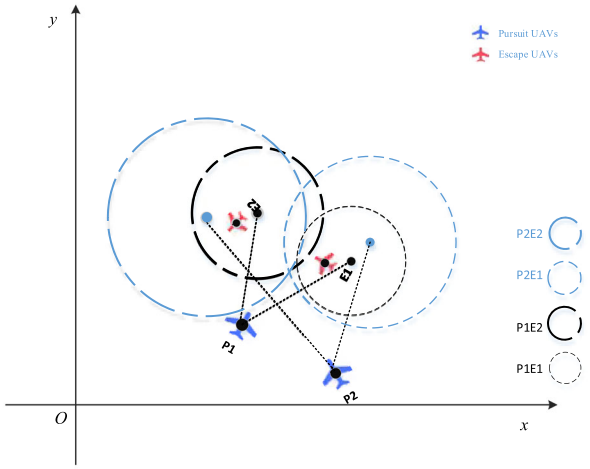

Fig. 4 The Apollonius circle formed by two-to-two pursuit

$$A_\tau^*=\arg\max V(x) \tag{8}$$

Escape strategy

Drawing on the ideas learned in the course, three gradually complex and intelligent escape strategies have been designed to facilitate progressive training of drones for pursuit. The escaping UAVs employ three different maneuvering strategies: straight-line motion, curved motion, and intelligent evasion motion. Straight-line motion: Escaped UAVs perform variable-speed straight-line motion. Curved motion: escaping UAVs are set to follow a sinusoidal curve motion pattern within the mission scenario. Intelligent evasion motion: when multiple pursuing UAVs enter the detection range of an escaping UAV, the escaping UAV will move in a direction perpendicular to the geometric center of the pursuing UAV cluster. In each round of algorithm training, the escape UAV randomly adopts a motion mode for maneuvering. Three progressively intelligent escape strategies are expressed as follows (Figs. 5, 6, 7).

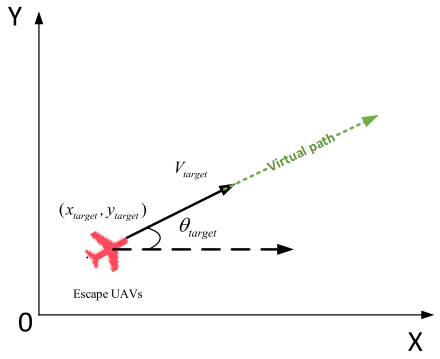

Fig. 5 The model diagram of straight-line motion for escape UAV

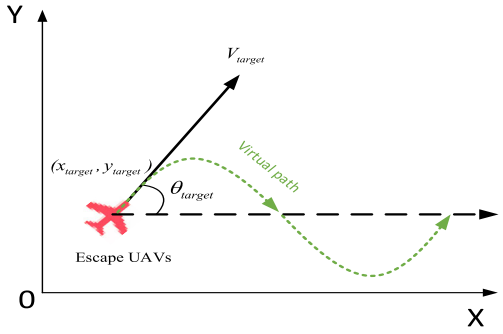

Fig. 6 The model diagram of curve motion for escape UAV

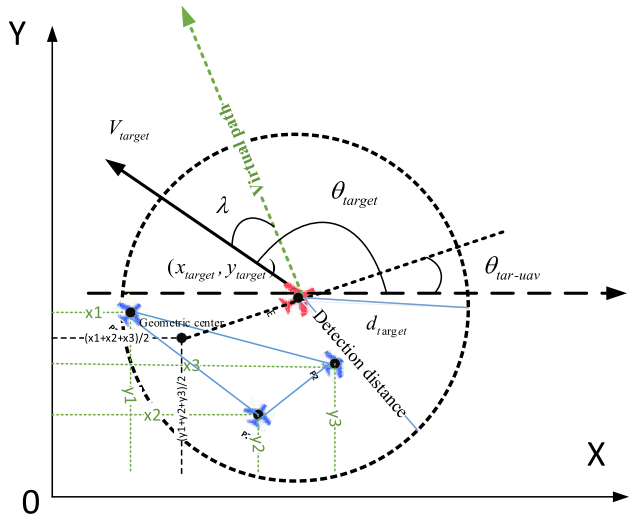

Fig. 7 The model diagram of intelligent escape motion for escape UAV

Straight-line motion:

$$\begin{cases} v'_{target} = v_{target} + a_v \cdot \Delta t \\ \theta'_{target} = \theta_{target} \end{cases} \tag{9}$$

$$a_v \in \left[-\frac{\pi}{6}, -\frac{\pi}{6}\right] a_\theta = 0 \tag{10}$$

$$\begin{cases} x'_{target} = x_{target} + v'_{target} \cdot \Delta t \cdot \cos\theta_{target} \\ y'_{target} = y_{target} + v'_{target} \cdot \Delta t \cdot \sin\theta_{target} \end{cases} \tag{11}$$

Curved motion:

$$y'_{target} = y_{target} + k \cdot \sin(\frac{1}{k}(x'_{target} - m) \tag{12}$$

$$\theta'_{target} = \arctan\left(-\cos\left(\frac{1}{k}\left(x_{target} - m\right)\right)\right) \tag{13}$$

$$\begin{cases} x'_{target} = x_{target} + v_{target} \cdot \Delta t \cdot \cos\theta'_{target} \\ y'_{target} = y_{target} + k \cdot \sin(\frac{1}{k}(x'_{target} - m) \end{cases} \tag{14}$$

$$\begin{aligned} a_v &= 0 \\ a_\theta &= (\theta'_{target} - \theta_{target})/\Delta t \end{aligned} \tag{15}$$

Intelligent evasion motion:

$$\begin{aligned} x_{center} &= \frac{x_1 + x_2 + \cdots x_n}{2} \\ y_{center} &= \frac{y_1 + y_2 + \cdots y_n}{2} \end{aligned} \tag{16}$$

$$\theta_{tar-uav} = \arctan((y_{target} - y_{center})/(x_{target} - x_{center})) \tag{17}$$

$$\theta'_{target} = \frac{\pi}{2} + \theta_{tar-uav} \tag{18}$$

$$a_v \in \left[0, \frac{\pi}{6}\right] a_\theta = \left(\frac{\pi}{2} + \theta_{tar-uav} - \theta_{target}\right) \Big/ \Delta t \tag{19}$$

Here t represents the simulation step. x_{target}, y_{target} represents the coordinates of the escaping UAV. $x\prime_{target}$, $y\prime_{target}$ represents the coordinates of the escaping UAV at the next time step. v_{target}, θ_{target} represents the magnitude and direction of the escaping UAV's velocity, where the direction is the angle between the velocity direction and the positive X-axis. $v\prime_{target}$, $\theta\prime_{target}$ represents the magnitude and direction of the escaping UAV's velocity at the next time step. a_v, a_θ represents the linear acceleration and angular acceleration of the escaping UAV. k and m are parameters influencing the curvature of the curve motion. x_{center} center and y_{center} are the coordinates of the geometric center of the pursuing UAV cluster. $\theta_{tar-uav}$ is the angle between the line connecting the escaping UAV and the geometric center of the pursuing UAV cluster within the detection range.

Algorithm design

MADDPG

The Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [15] is an algorithm used to address reinforcement learning in multi-agent environments where agents interact with each other. The core idea of this algorithm is distributed execution and centralized training. In other words, each agent uses its own state to estimate policies and output actions. It optimizes the policy network through Q-values. Additionally, it uses joint states to estimate values and output Q-values, optimizing the action-value function through environmental rewards.

MADDPG is a framework-based reinforcement learning algorithm, with each agent corresponding to one framework. The algorithm model consists of two parts: the actor and the critic. The actor is responsible for policy estimation, while the critic is responsible for value estimation. The shared information among all agents is the joint state action space. For an agent, the basic process to achieve the optimal strategy is as follows: First, the actor network selects a policy based on its input state, which yields an action. Then, the critic network calculates the agent's action value, i.e., the Q-value, based on the joint state-action information. Finally, the critic network performs value estimation and optimizes the value based on environmental feedback, while the actor network simultaneously performs policy estimation and optimizes the policy based on the value. This cycle continues, ultimately leading to the maximum value and the optimal strategy for all agents.

During the training phase, a batch of s samples, denoted as $S\left(x^j, a^j, r^j, s'_j\right)$ is randomly sampled from the sample buffer D and fed into the critic network [15]. This process calculates the Q-values as follows:

$$y^j = r_i^j + \gamma Q_i^{\mu}\left(x'^j, a'_1, \cdots, a'_n\right)\Big|_{a'_k=\mu'_k(o_k)} \tag{20}$$

The loss function for the critic network is as follows:

$$L(\theta_i) = \frac{1}{s}\sum_j \left[\left(Q_i^{\mu}(x^j, a_1^j, \cdots, a_n^j) - y^j\right)^2\right] \tag{21}$$

Then, the parameters of the critic network are updated through gradient descent using the following update formula.

$$\theta'_{critic} = \theta_{critic} - \alpha\frac{\partial(y' - y)}{\partial\theta_{critic}} \tag{22}$$

After receiving the Q values, the actor network is trained and updated using the gradient update formula as follows.

$$\begin{aligned}\nabla_{\theta_i} J(\mu_i) &= \frac{1}{s}\sum\nolimits_j \nabla_{\theta_i}\mu_i(a_i|o_i^j)\nabla_{a_i} Q_i^{\mu}(x^j, a_1^j, ..., a_n^j)\Big|a_i \\ &= \mu_i(o_i^j)\end{aligned} \tag{23}$$

Since the goal of the actor network is to maximise the Q values, it is updated by gradient ascent, and the update formula is as follows [16]:

$$\theta'_{actor} = \theta_{actor} + \beta\nabla_{\theta_i} J(\mu_i)\log\pi_{\theta_{actor}}(s_t, a_t)\cdot y \tag{24}$$

where α and β are the weighting parameters.

Max–min principle

The application of the max–min principle in reinforcement learning algorithms involves considering, in the worst-case scenario, finding the maximum value of the minimum value of the Q function [17]. To learn more robust strategies, it is assumed that other agents make the most unfavourable actions for themselves, meaning that other UAVs that pursue adopt actions with the minimum Q-values. This optimization of the agent's cumulative rewards enhances the robustness of the strategy. This results in the formation of the minimax learning objective $J_M(\theta_i)$:

$$\begin{aligned}\nabla_{\theta_i} J_M(\theta_i) = E_{x\sim D}[&\nabla_{\theta_i}\mu_i(O_i)\nabla_{a_i} Q_{M,i}^{\mu} \\ &(x, a_1^*, \ldots a_i^*, \ldots a_N^*)]\end{aligned}$$

$$a_i = \mu_i(O_i)$$

$$a^*_{i\neq j} = \arg\min_{a_{i\neq j}} Q^{\mu}_{M,i}(x, a_1, a_2 \ldots a_N) \tag{25}$$

The critic network is updated by minimizing the estimation error. The loss function is as follows:

$$L(\theta_i) = E_{x,a,r,x'\sim D}[(Q^{\mu}_{M,i}(x, a_1, a_2 \ldots a_N) - y)^2]$$

$$y = r + \gamma Q^{\mu'}_{M,i}(x', a_1^{*'}, \ldots a_i\prime, \ldots a_N^{*'})$$

$$a\prime_i = \mu\prime_i(O_i)$$

$$a^{*'}_{i\neq j} = \arg\min_{a_{i\neq j}} Q^{\mu'}_{M,i}(x', a\prime_1, a\prime_2 \ldots a\prime_N) \tag{26}$$

The actor network is updated using the sampled policy gradients to optimize its parameters. The optimization formula is as follows:

$$\nabla_{\theta_i} J \approx \frac{1}{S} \sum\nolimits_k \nabla_{\theta_i} \mu_i(o_i) \nabla_{a_i} Q^{\mu}_{M,i}(x^k, a_1^*, \cdots, a_i, \cdots, , a_N^*)$$

$$a_i = \mu_i \tag{27}$$

Multi-agent adversarial learning

Using the max–min principle for objective solving, the continuous action space and non-linear Q function result in a tremendous computational load. The introduction of Multi-Agent Adversarial Learning (MAAL) methods involves approximating the nonlinear state-value function, i.e., the Q-function, by constructing local linear functions. This approach replaces the inner-loop minimization method with a one-step gradient descent approximation to effectively address the problem.

Introducing a set of perturbations ε to disrupt the behavior a^* that minimizes the Q-value, and linearizing the Q-function $Q^{\mu}_{M,i}\left(x, \ a', \cdots a'_n\right)$. A perturbation value ε_j is sought that can locally approximate the Q-function in the gradient direction. Then, by taking a small gradient step, the value of this perturbation, which leads to the behavior a^* that minimizes the Q-value reduction, is approximated as expressed in the following equation:

$$a^{*'}_{j\neq i} = a'_{j\neq i} + \varepsilon_{j\neq i}$$

$$\varepsilon_{j\neq i} = \arg\min_{\in j\neq i} Q^{\mu'}_{M,i}(x', a'_1 + \varepsilon_1, \ldots a'_i \ldots a'_N + \varepsilon_N)$$

$$\widehat{\varepsilon_{j\neq i}} = -\alpha \nabla_{a_i} Q^{\mu'}_{M,i}(x, a'_1, \ldots a_i, \ldots a'_N) \tag{28}$$

where α represents an adjustable coefficient that can influence the step size of the gradient descent solver. A smaller α results in a smaller step size, which can improve computational precision but may make it more challenging to find an appropriate perturbation value. Conversely, a larger α leads to a larger step size but may result in poor performance of the linear fitting function, which is not conducive to effective training and learning.

Minimax multi-agent deep deterministic policy gradient

The M3DDPG algorithm, proposed as an improvement on the aforementioned algorithm, addresses the issue of deep reinforcement learning agents that are often fragile and sensitive to the training environment, especially in multi-agent scenarios [14].

To learn robust policies, the M3DDPG algorithm introduces the maximin principle, assuming that other agents make the most disadvantageous decisions for one's own agent. It also employs a multi-agent adversarial learning approach to reduce the significant computational complexity associated with nonlinear Q-function maximization and minimize. This enhances the robustness and convergence of the M3DDPG algorithm.

The pseudocode for the M3DDPG algorithm is as follows:

Algorithm: M3DDPG Algorithm

For training $\text{ep}^{isode} = 1 \cdots M$:

Initialize the agent's environment, initial state x, perform random exploration N to obtain actions.

For training rounds numbered $n = 1 \cdots N$:

For each agent i, select action $a_i = \mu_{\theta_i}(O_i) + N_t$ the current-time policy and random exploration.

For agent $i = 1 \cdots n$:

Sample random sample $S(x^k, a^k, r^k)$ from the memory replay buffer for training.

Update the critic_eval network parameters by minimizing the loss.

Train the critic network and compute the objective function.

$$y^k \quad r_i^k \quad \gamma Q_{M\,i}^{\mu} \; x^{\,k}\, a \qquad a_N \quad a_l \quad \mu_l \quad o_l^K$$

$$a'_{j\neq i} = \mu'_j\left(O_j^k\right) + \hat{\varepsilon'_j}$$

$$\hat{\varepsilon'_j} = -\alpha_j \nabla_{a'_j} Q_{M,i}^{\mu'}(x', a'_1, \cdots a'_N)$$

Calculate the loss function and obtain the corresponding gradients.

$$Loss = \frac{1}{S}\sum\nolimits_k \left(y^k - Q_{M,i}^{\mu}(x^k, a_1^k, \cdots, a_n^k)\right)^2$$

Update the actor_eval network parameters through gradient descent.

Train the actor network and update the gradients.

$$\nabla_{\theta_i} J \approx \frac{1}{S}\sum_K \nabla_{\theta_i} \mu_i(o_i^K) \nabla_{a_i} Q_{M,i}^{\mu}(x^k, a_1^k, \cdots, a_n^k)$$

$$|a_i^k = \mu_i(o_i^k) \qquad a_{j\neq i}^k = a_j^k + \hat{\varepsilon}_j$$

$$\hat{\varepsilon'_j} = -\alpha_j \nabla_{a'_j} Q_{M,i}^{\mu'}(x', a'_1, \cdots a'_N)$$

If the "target" network update period is reached:

Perform a soft update to update the parameters of the actor_target and critic_target networks.

end for

end for

end for

Particle swarm optimization algorithm

Particle Swarm Optimization (PSO) algorithm is an evolutionary computing technique initially proposed by Eberhart and Kennedy in 1995 [18]. PSO is the only evolutionary algorithm that does not involve survival of the fittest. Due to its simplicity and low computational cost, it has been successfully applied to a range of continuous optimization problems. In the particle swarm algorithm, each particle represents a potential solution to the problem. Optimization problems are solved through the simple behaviors of individual particles and the exchange of information within the population. In the context of RL problems, each particle represents a candidate policy and, through iterations, the PSO aims to find the optimal policy [19]. In each iteration, particles update their velocities and positions by tracking two "extremes": One is the best solution found by the particle itself (pbest), and the other is the best solution found by the entire population (global best or gbest).

The update process for particles is as follows (Fig. 8).

In the diagram, $\overrightarrow{x}(t)$ represents the particle's position at time t, $\overrightarrow{x}(t+1)$ represents the particle's position at the next time step, $\overrightarrow{v}(t)$ represents the particle's velocity at time t, $\overrightarrow{v}(t+1)$ represents the particle's velocity at the next time

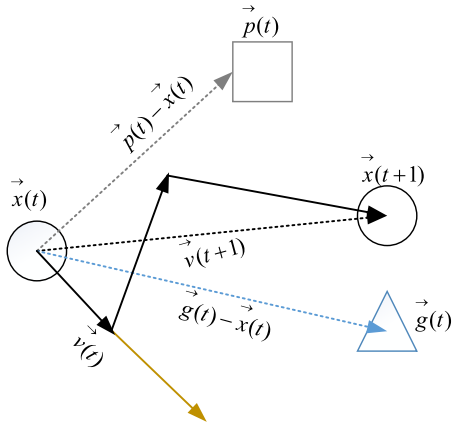

Fig. 8 Update method of the particle

step, $\vec{p}(t)$ represents the best solution found by the particle at time t, $\vec{g}(t)$ represents the historical best solution found by the entire particle swarm up to time t.

Each particle determines its own velocity and adjusts its trajectory based on its individual experience $\vec{p}(t)$ and the collective experience $\vec{g}(t)$ of the group. They move towards the optimal point. Different particles calculate their individual fitness values based on the corresponding objective function and assess their own quality [20]. The update formulas for particle velocity and position are as follows:

$$v_{id}(t+1) = wv_{id}(t) + c_1 r_1(p_i(t) - x_{id}(t)) + c_2 r_2(g(t) - x_{id}(t)) \tag{29}$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1) \tag{30}$$

In the equations above:w is the Inertia weight, which controls the change in particle velocity.r_1, r_2 express the random numbers between [0, 1], used to control the weight.c_1, c_2 are learning factors representing the random acceleration weights by which particles move towards their individual and global best values, respectively.

The PSO-M3DDPG algorithm

The M3DDPG algorithm is capable of conducting rapid local exploration and acquiring a significant amount of sample data. However, due to the sparsity of the sample space, it is prone to getting stuck in local optima or facing challenges in convergence. Therefore, by using the PSO algorithm to initialise multiple policy networks, creating a population of policy networks that interact with the environment, generating sample data, storing them in a buffer and applying them to train the M3DDPG algorithm, continuously improving the set of sample experience, and combining their strengths, it becomes possible to effectively address the problems inherent to each algorithm. This enables them to learn more efficiently which using PSO for the strategy improvement that is called the PSO-M3DDPG algorithm.

To address the issue of the enormous computational burden that arises when solving the objective in the M3DDPG algorithm using the principle of minimax optimization, a multi-agent adversarial approach is introduced. This approach involves linearly approximating the complex Q function to obtain a simpler linear function. Furthermore, improvements are made to the inner-loop minimization process by replacing it with a one-step gradient descent method. This modification significantly reduces the computational load and achieves algorithm optimization.

The workflow of the PSO-M3DDPG algorithm is as follows (Fig. 9):

Step 1: Initialize N policy network populations and M3DDPG network parameters.

Step 2: Compute the cumulative reward R for all policies within the population and store the transitions (s_t, a_t, r_t, s_{t+1}).

Step 3: Agents interact with the environment based on decisions made by deep neural networks, completing one episode of operation.

Step 4: Rank the policy networks based on the cumulative reward provided by the environment as the fitness value.

Step 5: Select the top $\varphi\%$ of policy networks as elites.

Step 6: Add random noise to the remaining policy networks to induce mutations.

Step 7: Store the transitions (s_t, a_t, r_t, s_{t+1}) for the mutated policy networks.

Step 8: Train the M3DDPG network using the acquired experience dataset from the samples.

Step 9: Copy the M3DDPG network parameters to the policy network population.

The structure diagram of the PSO-M3DDPG algorithm is as follows:

The pseudocode for the PSO-M3DDPG algorithm:

Algorithm: PSO-M3DDPG Algorithm

For training $\text{ep}^{isode} = 1 \cdots M$:

Initialize the agent's environment, initial state x, perform random exploration N to obtain actions.

For training rounds numbered $n = 1 \cdots N$:

For each agent i, select action $a_i = \mu_{\theta_i}(O_i) + N_t$ the current-time policy and random exploration.

For agent $i = 1 \cdots n$:

Sample random sample $S(x^k, a^k, r^k)$ from the memory replay buffer for training.

Update the critic_eval network parameters by minimizing the loss.

Train the critic network and compute the objective function.

$$y^k = r_i^k + \gamma Q_{M,i}^{\mu'}(x'^k, a'_1, \cdots, a'_N)|_{a'_l = \mu'_i(o_l^K)}$$

$$a'_{j \neq i} = \mu'_j\left(O_j^k\right) + \hat{\varepsilon}'_j$$

$$\hat{\varepsilon}'_j = -\alpha_j \nabla_{a'_j} Q_{M,i}^{\mu'}(x', a'_1, \cdots a'_N)$$

Calculate the loss function and obtain the corresponding gradients.

$$Loss = \frac{1}{S}\sum_k \left(y^k - Q_{M,i}^{\mu}(x^k, a_1^k, \cdots, a_n^k)\right)^2$$

Update the actor_eval network parameters through gradient descent.

Train the actor network and update the gradients.

$$\nabla_{\theta_i} J \approx \frac{1}{S}\sum_K \nabla_{\theta_i} \mu_i(o_i^K) \nabla_{a_i} Q_{M,i}^{\mu}(x^k, a_1^k, \cdots, a_n^k)$$

$$|a_i^k = \mu_i(o_i^k) \qquad a_{j \neq i}^k = a_j^k + \hat{\varepsilon}_j$$

$$\hat{\varepsilon}'_j = -\alpha_j \nabla_{a'_j} Q_{M,i}^{\mu'}(x', a'_1, \cdots a'_N)$$

If the "target" network update period is reached:

Perform a soft update to update the parameters of the actor_target and critic_target networks.

end for

end for

end for

$$y^k = r_i^k + \gamma Q_{M,i}^{\mu'}(x'^k, a'_1, \cdots, a'_N)|_{a'_i=\mu'_i(o_i^K)}$$

$$a'_{j\neq i} = \mu'_j\left(O_j^k\right) + \hat{\varepsilon'_j}$$

$$\hat{\varepsilon'_j} = -\alpha_j \nabla_{a'_j} Q_{M,i}^{\mu'}(x', a'_1, \cdots a'_N)$$

Compute the loss function and obtain the corresponding gradients.

$$Loss = \frac{1}{S}\sum_k \left(y^k - Q_{M,i}^{\mu}(x^k, a_1^k, \cdots, a_n^k)\right)^2$$

Update the actor_eval parameters through gradient descent:

Train the actor network and update the gradients.

$$\nabla_{\theta_i} J \approx \frac{1}{S}\sum_K \nabla_{\theta_i}\mu_i(o_i^K)\nabla_{a_i} Q_{M,i}^{\mu}(x^k, a_1^k, \cdots, a_n^k)$$

$$|a_i^k = \mu_i(o_i^k) \qquad a_{j\neq i}^k = a_j^k + \hat{\varepsilon}_j$$

$$\hat{\varepsilon'_j} = -\alpha_j \nabla_{a'_j} Q_{M,i}^{\mu'}(x', a'_1, \cdots a'_N)$$

If the 'target' network update period is reached:

Perform soft updates to update the parameters of the actor_target and critic_target networks.

end if

Copy the policy network of M3DDPG into the policy population.

For population individual j=1~K

Utilize the PSO algorithm to optimize and update the policy population, updating the parameters and velocities of the policy population.

$$\theta_{\mathrm{j}}^{\mu} = \theta_j^{\mu} + v_{\mathrm{j}+1}$$

$$v_{\mathrm{j}+1} = w \cdot v_j + c_1 \cdot rand1 \cdot \left(p_{best_j} - \theta_j^{\mu}\right) + c_2 \cdot rand2 \cdot \left(g_{best_i} - \theta_j^{\mu}\right)$$

end for

end for

end for

Multi-agent pursuit and evasion strategies for UAVs based on the PSO-M3DDPG algorithm

State space

In multi-agent pursuit-evasion decision tasks for UAVs, the local observation of the chasing UAVs includes their own state information, local interaction information, and the state information of the evading UAVs. The self-state information of a UAV can be described as $(x_i,\ y_i,\ v_i,\ \theta_i,\ team)$, where x_i and y_i represent the position information, v_i is the speed magnitude, θ_i is the speed direction, and 'team' indicates whether the UAV i is part of a pursuit team, with 'team' taking values 0,1.

The local interaction information includes data from the three nearest UAVs within communication range, expressed based on their relative distances $(x_k,\ y_k,\ v_k,\ \theta_k)$, $(x_l,\ y_l,\ v_l,\ \theta_l)$, and $(x_m,\ y_m,\ v_m,\ \theta_m)$, representing the positions, speed magnitudes, and speed directions of nearby friendly UAVs. When there are not enough other pursuing UAVs within communication range, the information content is filled with zeros.

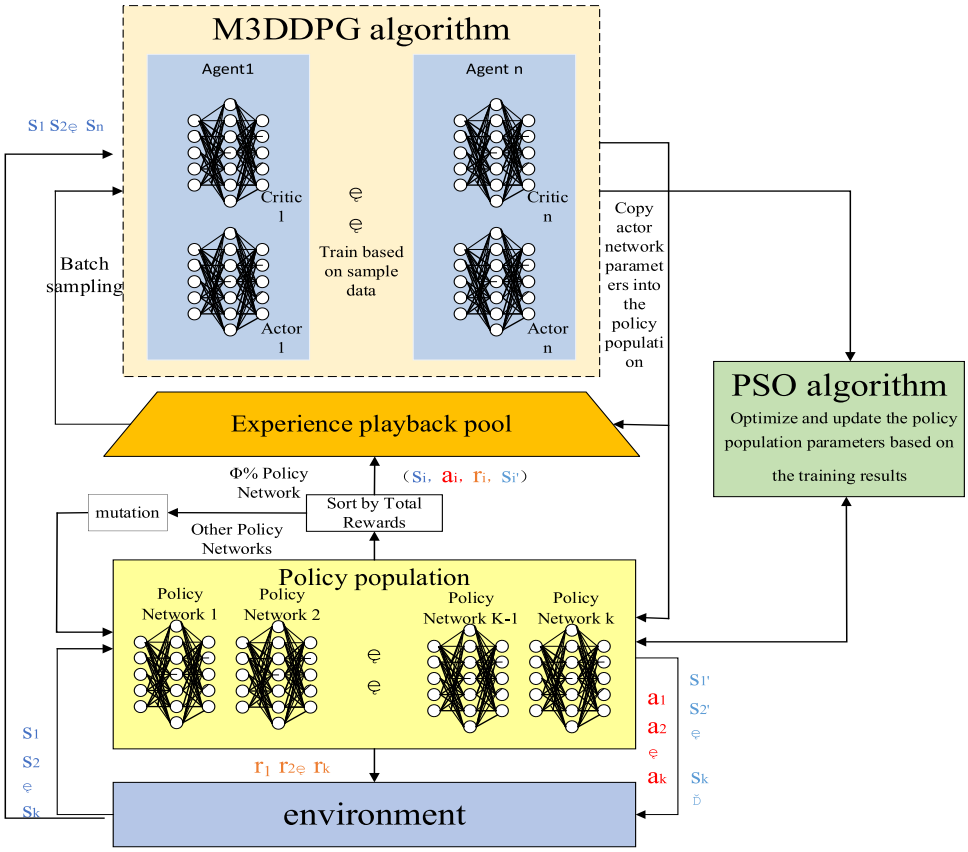

Fig. 9 PSO-M3DDPG algorithm structure diagram

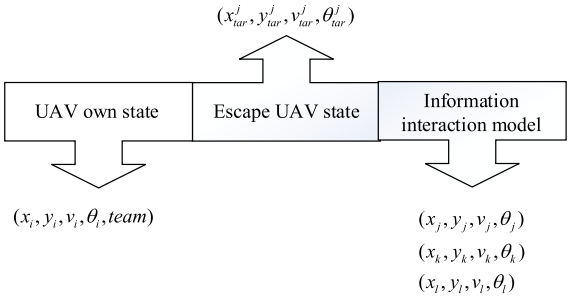

Fig. 10 State space for many-to-many pursuit and escape missions of UAVs

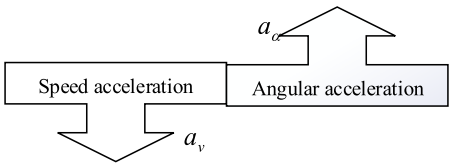

Fig. 11 Action space of UAV

In multi-UAV multi-pursuer evasion decision tasks, the state information of evading UAVs can be represented as $\left(x_{tar}^{j},\ y_{tar}^{j},\ v_{tar}^{j},\ \theta_{tar}^{j}\right),\ j = 1,2,\cdots$, where $j = 1,2,\cdots$, represents different evading UAVs, respectively indicating the position, speed magnitude, and direction of each evading UAV.

The complete state space is as follows (Fig. 10).

Action space

For the control of UAV motion, an acceleration-based approach is employed, where the UAV's action at each step consists of velocity acceleration a_v and angular acceleration a_α. This forms a two dimensional motion space represented as $(a_v,\ a_\alpha)$, as shown in Fig. 11.

Network model

The particle swarm algorithm optimized M3DDPG algorithm is applied to decision-making in multi-UAV pursuit-evasion tasks. Both the actor network and the critic network are constructed with 4 layers of fully connected neural networks, and the specific number of neurons in each layer is shown in Fig. 12.

Reward function

In multi-UAV pursuit-evasion tasks, the primary considerations revolve around the completion of the pursuit task and the

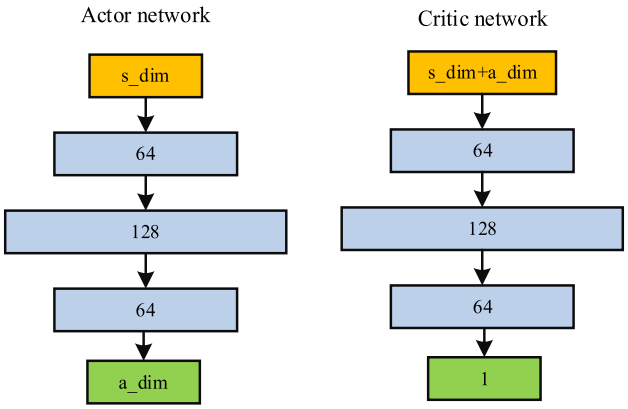

Fig. 12 Neural network structure diagram

collaborative requirements among the pursuit team. Regarding the completion of the pursuit task, two types of guiding global rewards are designed based on distance and direction, along with two local rewards for successful capture and task failure. In terms of the collaborative requirements among the pursuit team, two local rewards are designed for forming a pursuit team and avoiding collisions among UAVs.

The reward function for pursuing UAV i in the multi-UAV pursuit-evasion task is defined as follows:

$$r_i = r^i_{global} + r^i_{local} \tag{31}$$

The configuration of the global reward r_{golbal} is as follows:

$$r^i_{global} = r^i_d + r^i_a \tag{32}$$

In the global reward r_{global}, r_d represents the reward generated by the relative distance change between pursuing UAVs and evading UAVs, and its expression is as follows:

$$r^i_d = \beta \cdot (dis^i - dis^i\_) \tag{33}$$

r_a represents the directional guidance reward, and its expression is as follows:

$$r^i_a = \gamma \cdot \cos\varphi \tag{34}$$

where dis represents the current–time relative distance;$dis\_$ represents the next-time relative distance;φ represents the angle between the velocity vector of the pursuing UAV and the line connecting both the pursuing and evading UAVs' positions;β and γ are hyperparameters representing weight coefficients.

The expression for the local reward, r_{local}, is as follows:

$$r^i_{local} = r^i_{final} + r^i_{bound} + r^i_{team} + r^i_{danger} \tag{35}$$

The expression for the task completion reward, r_{final}, which represents the reward value for a UAV successfully capturing a single escaping UAV, is as follows:

$$r^i_{final} = \begin{cases} 20 & successfully\ captured\ target \\ 0 & other \end{cases} \tag{36}$$

The expression for the boundary reward/punishment, r_{bound}, which is used to assess whether both the pursuing and escaping UAVs have flown out of the mission area, signifying a mission failure, is as follows:

$$r^i_{bound} = \begin{cases} -20 & pursuit\ drone\ flies\ out\ of\ the\ mission\ area \\ -20 & escape\ drone\ flies\ out\ of\ the\ mission\ area \end{cases} \tag{37}$$

The expression for the team reward, r_{team}, which is used to determine whether the pursuing UAV has formed a sub-pursuit team and has been assigned a pursuit mission, and provides a positive reward when pursuing UAV i forms a sub-pursuit team, is as follows:

$$r^i_{team} = \begin{cases} 10 & drones\ form\ a\ pursuit\ team \\ 0 & other \end{cases} \tag{38}$$

The expression for the danger reward, r_{danger}, which represents the reward or penalty for collisions between pursuing UAV and is used to ensure that pursuing UAVs maintain a safe distance from each other, is as follows:

$$r^i_{danger} = \begin{cases} -20 & if\ d_{ij} \le d_{danger} \\ \alpha_{danger}(d_{safe} - d_{ij}) & if\ d_{danger} < d_{ij} \le d_{safe} \\ 0 & other \end{cases} \tag{39}$$

where α_{danger} represents the weight coefficient.

Simulation experiments

Simulation parameters are set as follows (Table 1).

Training process

Using both the basic M3DDPG algorithm and the improved PSO-M3DDPG algorithm as decision units for pursuing UAVs, model training is conducted for the task of multi-UAV pursuit-evasion game. In each training round, the initial state of the UAVs is randomly initialized. The different initial scenarios for the multi-UAV pursuit and evasion task examples are illustrated in Figs. 13, 14.

Table 1 UAVs pursuit-evasion task training parameters

Parameters	Value
The horizontal width of the battlefield (m)	$W = 500$
The vertical length of the battlefield (m)	$H = 500$
The number of our UAVs	$N = 2/4$
The speed of our UAVs (m/s)	$v_t^i \in [1, 2.5]$
The number of enemy UAVs	$M = 2$
The speed of enemy UAVs (m/s)	$v_t^j \in [2, 5]$
UAVs linear acceleration (m/s^2)	$a_{vt}^i \in [-1, 1]$
UAVs angular acceleration (rad/s^2)	$a_{\alpha t}^i \in [-\frac{\pi}{6}, \frac{\pi}{6}]$
Learning rate	$\alpha = 0.005$
Reward discount factor	$\gamma = 0.98$
Sample size	$Batch\_size = 128$
Training rounds	$T = 3000$

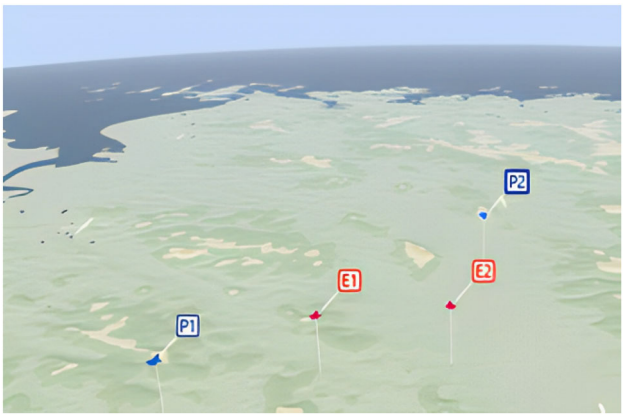

Fig. 13 Initial situation map of UAVs two-to-two pursuit and escape mission

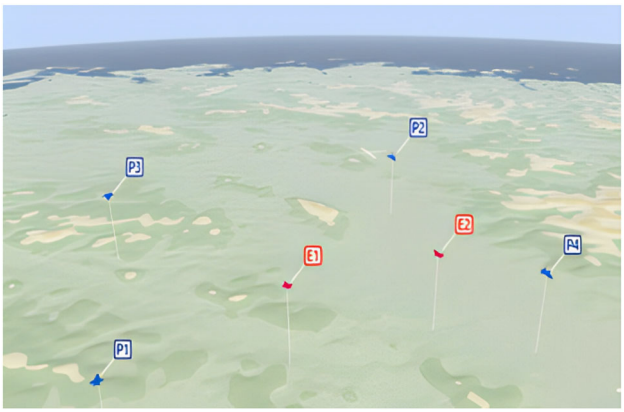

Fig. 14 Initial situation map of UAVs four-to-two pursuit and escape mission

After initialization, in each round of the training process, the escape UAV randomly adopts one of the maneuvering modes of linear motion, curved motion, or intelligent motion to escape, and the training effects are introduced below.

An analysis of the convergence of artificial neural network parameters is conducted. The following figure shows the numerical changes in the mean and variance of network weight parameters in the 'actor_eval' neural network of both algorithms as the training epochs process, as shown in Fig. 15.

From the above figure, it can be seen that in the initial training of the M3DDPG algorithm, due to the random initialization of neural network parameters following a normal distribution, it is prone to local optima during the pursuit-evasion decision-making process. Therefore, in the training process, there is a large optimization range for the parameters, resulting in slow convergence. On the other hand, in the training of the PSO-M3DDPG algorithm, the use of the PSO algorithm to optimize the strategy network population that generates sample data, combined with the M3DDPG algorithm for exploration, results in a better sample data set that is applied to algorithm training. In the neural network updates, the overall parameter optimization range is smaller, and the convergence speed is significantly faster. As the learning process progresses, the neural network parameters gradually approach their optimal values until they converge, reaching a stable state and obtaining a stable decision-making model for the UAVs behavior.

The UAVs was trained by using both the PSO-M3DDPG algorithm and the M3DDPG algorithm. The mean individual round rewards and the overall rewards for the UAVs were recorded in each training round. These metrics serve as critical indicators of how well the UAVs interacted with the environment. The results are shown in Fig. 16.

From the above figure, it can be observed that as the training progresses, the rewards gradually increase and eventually converge. However, the initial reward value for the PSO-M3DDPG algorithm is higher than that for the M3DDPG algorithm. Furthermore, the overall learning efficiency and final convergence results are significantly better for the PSO-M3DDPG algorithm compared to the M3DDPG algorithm. This indicates that the use of the PSO algorithm to optimize the sample data set significantly promotes the learning process of neural networks, accelerates the convergence speed of the algorithm, and leads to a better convergence result.

Validation process

Training the pursuit UAVs to engage with escape UAVs that employing different evasion strategies. Validate the performance of the trained neural network models. Employ the converged artificial neural networks as decision-making units for the pursuit UAVs. Conduct multi-UAV pursuit and

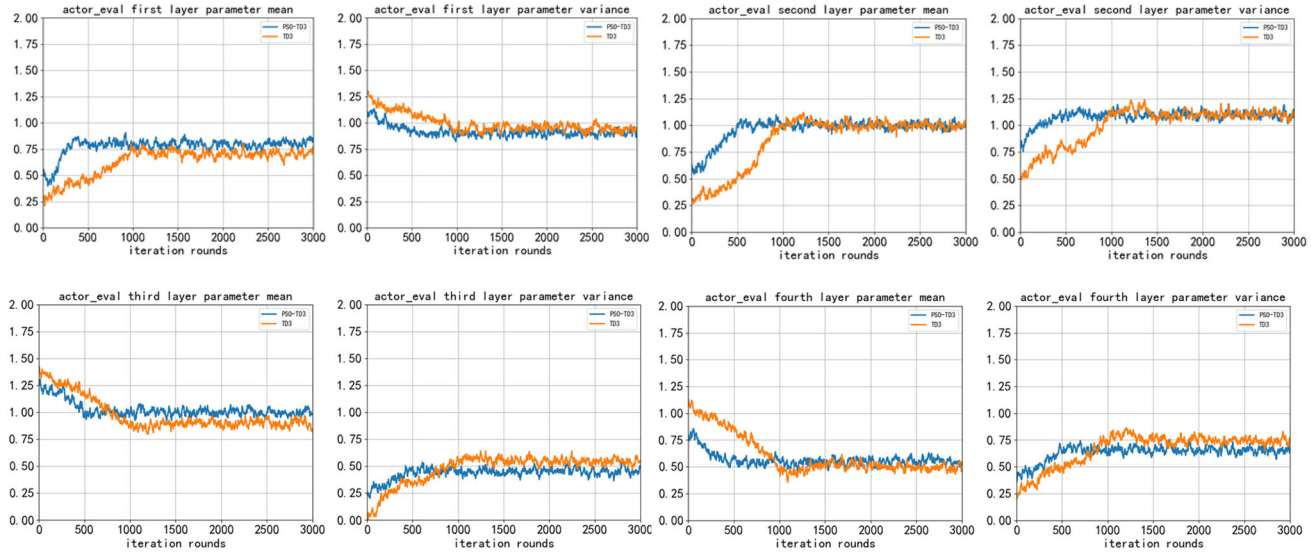

Fig. 15 Mean and variance variations of 'actor_eval' network weight parameters

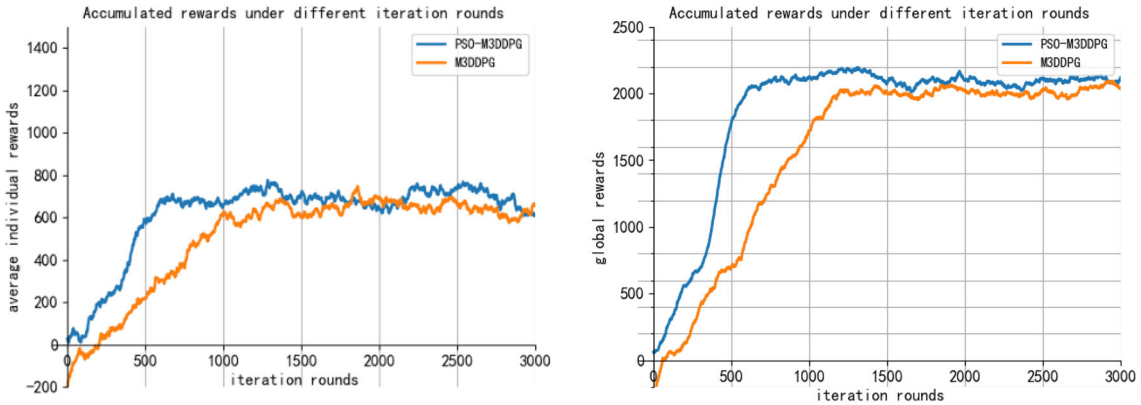

Fig. 16 Average individual and global rewards during training process

evasion tasks under varying conditions, including different quantities and initial states of UAVs. Analyze the trajectory of the pursuit UAVs to assess their performance.

When the escape UAVs perform simple straight-line movements, the trajectory diagram is as follows (Figs. 17, 18).

When the escape UAVs perform simple curved movements, the trajectory diagram is as follows (Figs. 19, 20).

When the escape UAVs perform complex adversarial movements, the trajectory diagram is as follows (Figs. 21, 22).

From the motion trajectory diagram above, it can be seen that different numbers of UAVs can effectively completed the pursuit and evasion tasks for targets with different movement patterns, performing well. The target decomposition and task allocation for the pursuit of drones have been designed, so that multiple pursuit drones can form an effective sub team to capture the escaping drones one by one. In response to the problem of falling into local minima caused by unreasonable initial value assignment in neural networks, which may lead to convergence oscillation or non convergence, the particle

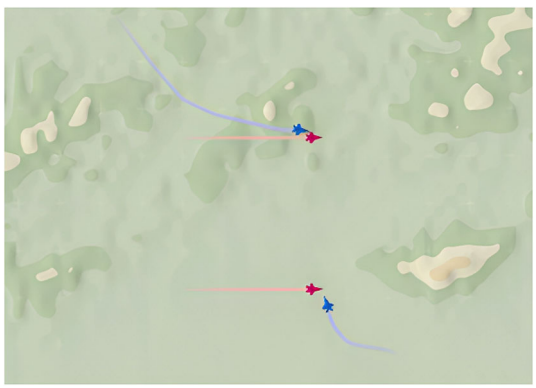

Fig. 17 Trajectory diagram of UAVs for two-to-two pursuit and escape mission

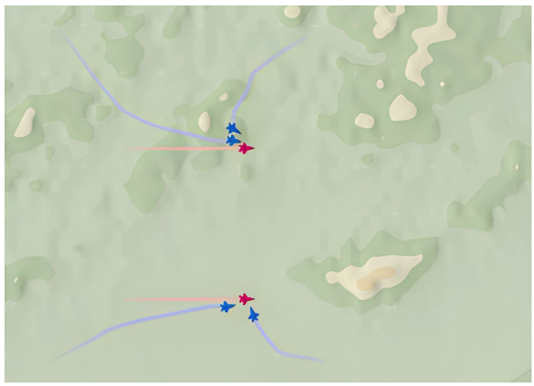

Fig. 18 Trajectory diagram of UAVs for four-to-two pursuit and escape mission

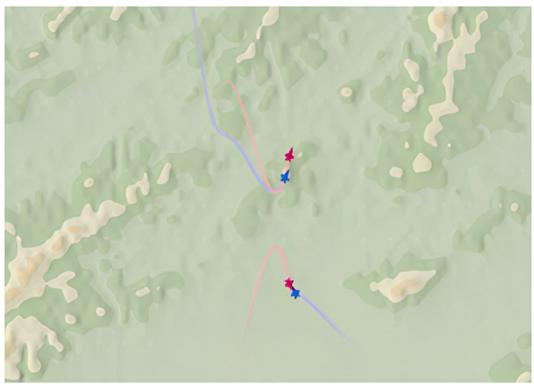

Fig. 19 Trajectory diagram of UAVs for two-to-two pursuit and escape mission

Fig. 20 Trajectory diagram of UAVs for four-to-two pursuit and escape mission

swarm optimization algorithm and M3DDPG algorithm are combined to search and learn the experience sample set of deep neural networks to a certain extent. The particle swarm algorithm is used to obtain a relatively optimal solution in the overall optimization process, and then the gradient descent of the neural network is used for detailed optimization learning, ultimately obtaining the optimal solution.

Fig. 21 Trajectory diagram of UAVs for two-to-two pursuit and escape mission

Fig. 22 Trajectory diagram of UAVs for four-to-two pursuit and escape mission

Use the improved PSO-M3DDPG algorithm for specific model construction and algorithm design. Through training, use the artificial neural network constructed by the improved algorithm to command the pursuit of drone clusters and gradually achieve the pursuit task of multiple escaping drones. The simulation verified the effectiveness of the improved algorithm as a behavioral decision-making unit for pursuing drones in achieving multi to many pursuit and evasion tasks. Moreover, the improved PSO-M3DDPG algorithm has a faster convergence speed and better decision-making strategy compared to the original algorithm.

Conclusion

This article focuses on the research of multi-UAV pursuit and evasion games [21], and improves upon traditional multi-agent cooperative algorithms [22] based on minimax optimization. It adopts a multi-agent adversarial learning approach for minimax target solving, co mbining the PSO algorithm with the M3DDPG algorithm, proposing the PSO-M3DDPG algorithm. This algorithm utilizes the PSO

algorithm to generate and continuously optimize the empirical sample set. Simulation experiments show that compared to the M3DDPG algorithm, this algorithm exhibits faster convergence, better robustness, and achieves a higher success rate in pursuit and evasion tasks.

Currently, most reinforcement learning algorithms are limited to small-scale intelligent agent environments and applied to large-scale cluster control problems, which suffer from dimension explosion and extremely high environmental complexity. The population optimization characteristics of evolutionary algorithms are expected to solve this problem, and future work will pay more attention to the deep combination of evolutionary algorithms and reinforcement learning.

Data availability The data supporting the findings of this study are available within the article.

References

1. Zhang X, Li L, Jia LL (2015) Research and simulation of multi robot pursuit and escape strategies based on differential games. Equip Manuf Technol 09:9–12
2. Tan FX, Liu DR, Guan XP et al (2014) Review and prospect of nonlinear control based on differential game theory. J Autom 40(1):1–15
3. Zhao L, Li C, Guo X (2018) Research of cooperative relief strategy between government and enterprise based on differential game. Syst Eng Pract 38:885–898
4. Song X, Wu C, Stojanovic V et al (2023) 1 bit encoding–decoding-based event-triggered fixed-time adaptive control for unmanned surface vehicle with guaranteed tracking performance. Control Eng Pract 135:105513
5. Fu L, Wang XG (2012) Research on differential game modelling for close range air combat of unmanned aerial vehicles. Def Technol 33(10):1210–1216
6. Li YL, Juan L, Liu C et al (2022) Application research of differential games in attack and defence of unmanned aerial vehicles clusters. Unmanned Syst Technol 5(05):39–50
7. Liu J, Wang G, Fu Q et al (2023) Task assignment in ground-to-air confrontation based on multiagent deep reinforcement learning. Def Technol 19:210–219
8. Lowe R, Wu Y I, Tamar A, et al (2017) Multi-agent actor-critic for mixed cooperative-competitive environments. Adv Neural Inform Process Syst 30
9. Hao J, Huang D, Cai Y et al (2017) The dynamics of reinforcement social learning in networked cooperative multiagent systems. Eng Appl Artif Intell 58:111–122
10. Donghua LI, Jiang J, Jiang C (2009) A flight path planning algorithm based on multi-agent reinforcement learning method. Electron Opt Control 16(10):10–14
11. Wang Q, Huang Y, Chang J (2021) Research on a downlink transmission power control algorithm for dense unmanned aerial vehicle networks. Electr Measure Technol 44(13):59–67
12. Fang M, Groen FCA (2013) Collaborative multi-agent reinforcement learning based on experience propagation. J Syst Eng Electron 24(4):683–689
13. Song M P, Gu G C, Zhang G Y, et al. (2007) Cooperative multi-agent learning in general sum games. Control Theory Appl (02):317–321
14. Li S, Wu Y, Cui X et al (2019) Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient. Proceed AAAI Conf Artif Intell 33:4213–4220
15. Zhang TT, Lan YS, Song AG (2021) Behavior decision learning reward mechanism for unmanned cluster systems. J Beijing Univ Aeronaut Astronaut 47(12):2442–2451
16. Wang S, Duan J, Shi D et al (2020) A data-driven multi-agent autonomous voltage control framework using deep reinforcement learning. IEEE Trans Power Syst 35(6):4644–4654
17. Martins RM, Gresse Von Wangenheim C (2023) Findings on teaching machine learning in high school: a ten-year systematic literature review. Inform Educ 22(3):421–440
18. Eberhart R, Kennedy J (1995) Particle swarm optimization. Proceed IEEE Int Conf Neural Netw 4:1942–1948
19. Yang W, Li QQ (2004) Overview of particle swarm optimization algorithms. Chin Eng Sci 6(5):87–94
20. Li AG, Qin Z, Bao FM et al (2002) Particle swarm optimization algorithm. Comput Eng Appl 38(21):1–3
21. Li B, Yang Z, Chen D et al (2021) Maneuvering target tracking of UAV based on MN-DDPG and transfer learning. Def Technol 17(02):457–466
22. Fan J, Li D, Li R et al (2020) Analysis on MAV/UAV cooperative combat based on complex network. Def Technol 16(01):150–157