entropy

Article

An Improved Approach towards Multi-Agent Pursuit–Evasion Game Decision-Making Using Deep Reinforcement Learning

Kaifang Wan *, Dingwei Wu, Yiwei Zhai, Bo Li, Xiaoguang Gao and Zijian Hu

School of Electronics and Information, Northwestern Polytechnical University, Xi'an 710072, China; wudingwei@mail.nwpu.edu.cn (D.W.); zyw@mail.nwpu.edu.cn (Y.Z.); libo803@nwpu.edu.cn (B.L.); cxg2012@nwpu.edu.cn (X.G.); huzijian@mail.nwpu.edu.cn (Z.H.)
* Correspondence: wankaifang@nwpu.edu.cn

Abstract: A pursuit–evasion game is a classical maneuver confrontation problem in the multi-agent systems (MASs) domain. An online decision technique based on deep reinforcement learning (DRL) was developed in this paper to address the problem of environment sensing and decision-making in pursuit–evasion games. A control-oriented framework developed from the DRL-based multi-agent deep deterministic policy gradient (MADDPG) algorithm was built to implement multi-agent cooperative decision-making to overcome the limitation of the tedious state variables required for the traditionally complicated modeling process. To address the effects of errors between a model and a real scenario, this paper introduces adversarial disturbances. It also proposes a novel adversarial attack trick and adversarial learning MADDPG (A2-MADDPG) algorithm. By introducing an adversarial attack trick for the agents themselves, uncertainties of the real world are modeled, thereby optimizing robust training. During the training process, adversarial learning was incorporated into our algorithm to preprocess the actions of multiple agents, which enabled them to properly respond to uncertain dynamic changes in MASs. Experimental results verified that the proposed approach provides superior performance and effectiveness for pursuers and evaders, and both can learn the corresponding confrontational strategy during training.

Keywords: pursuit–evasion; multi-agent; deep reinforcement learning; decision-making; adversarial learning; MADDPG

check for updates

Citation: Wan, K.; Wu, D.; Zhai, Y.; Li, B.; Gao, X.; Hu, Z. An Improved Approach towards Multi-Agent Pursuit–Evasion Game Decision-Making Using Deep Reinforcement Learning. *Entropy* **2021**, *23*, 1433. https://doi.org/10.3390/e23111433

Academic Editors: Kelvin Wong and Dhanjoo N. Ghista

Received: 22 September 2021
Accepted: 25 October 2021
Published: 29 October 2021

1. Introduction

With the development of the three generations of artificial intelligence [1], the technology of multi-agent systems (MASs) has been widely used in many areas of society, such as multi-agent motion planning, complex IT systems, computer communication technology, and so on [2–5]. Pursuit–evasion games have been widely investigated in MASs during recent years. They have been extended to various fields, to include maneuvering target tracking, surveillance early warning, anti-intrusion protection, and intelligent transportation [6,7]. The goal of these studies is to provide good strategies for pursuers and evaders. For pursuers, their goal is to round up the evaders as much as possible through cooperative decision-making. For evaders, they need to choose the best strategy based on the actions of pursuers to design an escape path to prevent being captured [8].

To address this problem, a series of research activities on agent-based pursuit–evasion games has been carried out in the differential gaming field. Isaacs [9] proposed a one-to-one robot hunting problem where partial differential equations describing the pursuer and the evader were created and solved analytically. Furthermore, a generalized maximum–minimum solving method of the Hamilton Jacobi equation for pursuit–evasion games was provided by Krasovskii [10]. Because in complex control problems, directly solving differential equations is very complicated and consumes many computing resources, researchers proposed some intelligent optimization algorithms that provide new ideas for solving the differential equation problems associated with pursuit–evasion games. Chen et al. [11]

simulated fish foraging behavior and proposed a cooperative pursuit strategy that studied pursuit and evasion when trackers have a constrained turning rate. Wang et al. [12] introduced an alliance generation algorithm that generates a synergistic strategy based on the emotional factors of multirobot systems. This ensured that a team's agents worked towards a common goal. However, there are many constraints and state variables involved in the complicated control process governing these issues, which make the solution intricate especially in a complex and dynamic scenario with multi-agent confrontation. Therefore, more intelligent algorithms are needed to effectively solve the problem of pursuit–evasion games.

By combining deep learning's ability to perceive highly dimensional data [13] and reinforcement learning's decision-making ability [14], deep reinforcement learning (DRL) provides a new optimization scheme for intelligent decision-making or control. Because techniques based on deep reinforcement learning do not require the establishment of a differential game model and agents can learn the optimal confrontation strategy only through interaction with the environment [15], some scholars introduced deep reinforcement learning in pursuit–evasion games and acquired the Nash equilibrium of the problem. Xu et al. [16] established a multi-agent reinforcement learning model for UAV pursuit–evasion in which relative motion state equations were employed. As a result, the pursuit–evasion issue was converted into a zero-sum game addressed through minimax-Q learning. In predatory games, Park et al. [17] set up a co-evolution framework for predator and prey to allow multiple agents to learn good policies by deep reinforcement learning. Gu et al. [18] presented an attention-based fault-tolerant model, which could also be applied to pursuit–evasion games, and the key idea was to utilize the multihead attention mechanism to select the correct and useful information for estimating the critics. To solve the complicated training problems caused by discrete action sets introduced by deep Q networks [19], Liu et al. [20] transformed a space rendezvous optimization problem between a space vehicle and noncooperative target into a pursuit–evasion differential game. They introduced a branching architecture with a group of parallel neural networks and shared decision modules. To overcome the unstable recognition ability of pursuers, Qadir et al. [21] proposed a novel approach for self-organizing feature maps and deep reinforcement learning based on the agent group role membership function model. Experiments verified the effectiveness of this method for facilitating the capture of evaders by mobile agents. Singh et al. [22] built on the actor–critic model-free multi-agent deep deterministic policy gradient algorithm to operate over the continuous spaces of pursuit–evasion games. In their approach, the evader's strategy is not learned. It is based on Voronoi regions that pursuers try to minimize and evaders try to maximize.

Although they represent progress, previous studies on DRL-based pursuit–evasion games are still in their early stages. In these studies, pursuing platforms are assumed to be equipped with error-free identification and measurement systems that allow them to acquire precise information about the position, velocity, and other characteristics of evaders and cooperators [6,23]. However, sensors and other equipment configured in an unmanned system encounter positioning, sensing, and actuator error in reality [24,25]. These errors cause the environment to become uncertain, thereby affecting the strategies of the pursuers and evaders and making their performance worse. Therefore, this research is about designing a robust algorithm for MASs to effectively mitigate these errors and that would be significant for application research in real-world multi-agent decision-making.

This paper introduces a novel multi-agent algorithm to address the decision-making problem of pursuit–evasion games. The algorithm can solve pursuit–evasion games in complex virtual and real environments, where there are static or moving obstacles and pursuers and evaders need to avoid them while making decisions. Specifically, we make the following contributions in this paper:

(1) We develop an actor–critic-based motion control framework based on the multi-agent deep deterministic policy gradient (MADDPG) [26], which can take the state and behavior of other partners into account and is used to provide collaborative decision-making capabilities for each agent in the MAS;

(2) We propose an advanced algorithm called A2-MADDPG, which uses two skills to make the training strategy robust. The first is adversarial attack tricks for agents. It proposes to sample the status after stochastic Gaussian noise is applied, and this approach can train a robust agent to cope with measurement errors in the real world. The second is the optimized adversarial learning technique [27]. It is introduced to improve agent stability and to assist in adapting to noise produced by interactions between multiple agents;

(3) We verified the effectiveness and robustness of the algorithm in simulation experiments. We compared the performance of the proposed method with two common and advanced algorithms, namely the MADDPG and the independent multiagent deep deterministic policy gradient (IMDDPG), where the IMDDPG is a natural extension of the DDPG [28] in the field of multi-agents. Through a series of experiments, we show that the proposed method presents excellent performance for both pursuers and evaders compared with the MADDPG and IMDDPG in the case of the same hyperparameter settings and simulation environment parameter settings, and it can help them both develop robust motion strategies.

The rest of the paper is structured as follows: Section 2 provides background information about multi-agent pursuit–evasion games and describes related theoretical approaches. Section 3 introduces a framework for collaborative pursuit missions and an improved A2-MADDPG algorithm where an adversarial attack trick and an adversarial learning-based optimization method are combined with the MADDPG. Section 4 verifies the robustness and high performance of the algorithm through simulation experiments. Section 5 provides a conclusion and envisages future work.

2. Background

In this section, the kinematic and observation model of agents executing a pursuit–evasion task is presented. In addition, the essential theoretical background of the DRL-based MADDPG algorithm and adversarial learning is introduced.

2.1. Problem Definition

The multi-agent pursuit–evasion game problem can be described as follows: there are pursuers (red agents) and evaders (blue agents), as shown in Figure 1. Both agent types have different tasks based on their maneuverability. Each agent can perceive the relative position of the threat zone (gray circle) using radar and sensors. The velocity and position of each agent are provided by its navigation equipment, and they share information by transmitting through a signaling connection. Pursuers are equipped with an attack or shielding interference device (the red circle represents the attack range of the pursuit), and their mission is considered successful when they suppress an evader by approaching it. Evaders must stay away from pursuers. Neither pursuers nor evaders can exceed their boundaries.

2.1.1. Comparisons of Operators

A general decision-making program for pursuit–evasion gaming is primarily used to determine the communication and cooperation between platforms and achieve target pursuit. This is performed without fully considering the maneuvering characteristics of the platforms. Both agents in this paper are mobile UAVs flying at a fixed altitude with nonholonomic constraints [29], as portrayed in Figure 2. The status update of each UAV can be described as:

$$\begin{cases} p^t = \begin{bmatrix} x^{t-1} + v_x^t \cdot \Delta t \\ y^{t-1} + v_y^t \cdot \Delta t \end{bmatrix} \\ v^t = \begin{bmatrix} v_x^{t-1} + a_x^t \cdot \Delta t \\ v_y^{t-1} + a_y^t \cdot \Delta t \end{bmatrix} \\ \varphi^t = a\tan\left(\frac{v_y^t}{v_x^t}\right) \end{cases} \tag{1}$$

where p^t, v^t, and φ^t denote the position, velocity, and yaw angle parameters. The superscript t represents time t; Δt is the time interval; a is UAV's acceleration. Considering power systems and mechanical limitations, the maximal velocity and acceleration are assumed to be v_{max} and a_{max}, which are introduced in the following simulation.

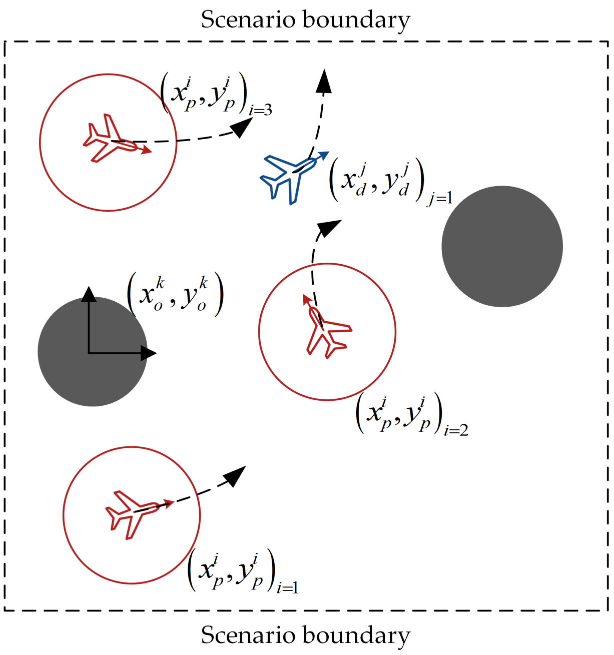


Figure 1. The scenario of multi-agent pursuit–evasion games.

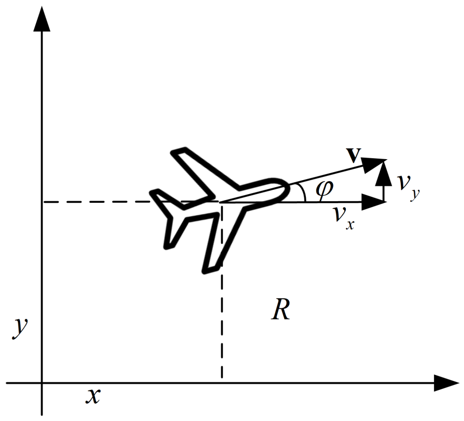


Figure 2. Motion analysis of the UAV.

2.1.2. Observation Model

The observation model of the agent was presented to provide the agent with the ability to sense the environment [30]. In this multi-agent pursuit–evasion task, (x_a^i, y_a^i) is the position of each agent in the pursuit formation, and both (x_e^j, y_e^j) and (x_o^k, y_o^k) represent the position of the center point of the evader and the threatened area, respectively. The number of pursuers, evaders, and obstacles in the environment is defined as num_P, num_E, and num_o, respectively. Since pursuers and evaders need to consider avoiding obstacles to prevent being hit when making decisions, these obstacles make it more difficult to solve the problem of pursuit–evasion. The formation of all pursuers is denoted as A. An agent i on the pursuers' team can use radar detection and communication transmission to obtain its own local observations from the environment as follows:

$$\begin{cases} O_s^i = \left[\boldsymbol{v}_s^i, \boldsymbol{p}_\mathbf{s}^\mathbf{i}\right] = \left[v_{sx}^i, v_{sy}^i, x_s^i, y_s^i\right] \\ O_c^i = \left[\boldsymbol{p}_s^l\right]_{l=1,\ldots,i-1,i+1,num\_P} = \left[x_c^l, y_c^l\right]_{l=1,\ldots,i-1,i+1,num\_P} \\ O_e^i = \left[\boldsymbol{p}_s^j\right]_{j=1,\ldots,num\_E} = \left[x_e^j, y_e^j\right]_{j=1,\ldots,num\_E} \\ O_o^i = \left[\boldsymbol{p}_o^k\right]_{k=1,\ldots,num\_o} = \left[x_o^k, y_o^k\right]_{j=k,\ldots,num\_o} \end{cases} \tag{2}$$

Here, $v_s x^i$, $v_s y^i$, x_s^i, and y_s^i represent the self-observed velocity and position of the agent on the x and y axis. O_c^i indicates the observed location of other pursuers in the formation, and l is the sequence number of other pursuers on the team. O_e^i denotes the observed location of an evader, and j represents its sequence number. O_o^i represents information observed about an obstacle, and k is the obstacle number. Considering a real mission scenario, a set of range sensors is employed to help the unmanned system detect possible threats from obstacles ahead of it in the range. As shown in Figure 3, the 90° angle containing the blue arc within the sensor range is the agent's threat detection area. An agent's observations about an obstacle are divided into five parts:

$$O_o^i = [d_1, d_2, d_3, d_4, d_5] \tag{3}$$

where d_{1-5} denotes the five sensor indications. We set $d_{1-5} = L$ when a threat is not detected. Based on the comprehensive observation information above, an agent can perceive and assess the situation.

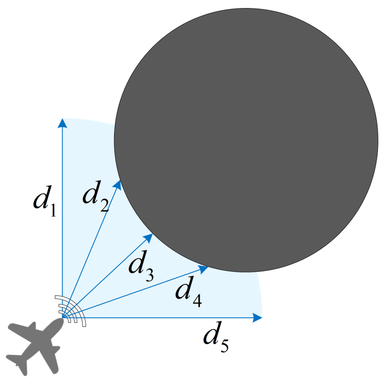


Figure 3. Unmanned system obstacle threat detection based on range sensors.

2.2. Theoretical Context

Deep reinforcement learning is a representative intelligent machine learning algorithm, and adversarial learning can increase the stability and robustness of the model trained by reinforcement learning [31]. They provide new research ideas for multi-agent pursuit–evasion decision-making. In this section, adversarial learning, the DRL-based DDPG algorithm, and the MADDPG algorithm are introduced.

2.2.1. Adversarial Learning

Adversarial learning is a technique of defending against adversarial samples [32]. This approach attempts to improve the accuracy of neural network models by training adversarial samples and normal samples together and reducing the interference of the adversarial samples. The robustness and generalization ability of the resulting network are improved. Adversarial training can be expressed as follows:

$$\min_{\theta} \max_{D(x,x')<\eta} L_{adv}(x', y; \theta) \tag{4}$$

where x, x' denote the original sample and adversarial sample, respectively, y is the label value, and θ is the weight of the networks. $D(x, x')$ represents the distance measurement between the original sample and the adversarial sample, and $L_{adv}(x', y; \theta)$ represents the adversarial loss function. In the min–max form, the internal maximization optimization problem is to find the optimal adversarial sample, and the external minimization optimization problem is to minimize the loss function. The learning process of confrontation training is depicted in Figure 4.

The fast gradient sign method (FGSM) efficiently generates adversarial samples [33]. The FGSM uses a model's objective loss function to determine the input vector needed to calculate its counter disturbance, which it adds to the corresponding input. This generates counter samples that correspond to the original samples. Suppose that in a classification problem, the output label of the model is $class = \{0, 1\}$. After adversarial training, the model will have higher prediction confidence, that is the model will output the correct sample label even if a small adversarial disturbance is added to the sample. This process can be defined as:

$$x^* = x + \eta_{adv} \tag{5}$$

$$class(x^*) = class(x) \tag{6}$$

where η_{adv} represents the sample perturbation added and x^* represents adversarial samples after adding perturbation. Each time the model is trained, the FGSM performs an optimization along the gradient direction of the counter loss function $L_{adv}(x', y; \theta)$, and counter samples are obtained. The generation process of sample disturbance η_{adv} can be expressed as:

$$\eta_{adv} = -\varepsilon \frac{g}{\|g\|_2} \tag{7}$$

$$g = \nabla_x L_{adv}(x', y; \theta) \tag{8}$$

where ε denotes the magnitude of disturbance and g is the inverse gradient of the input vector. Moreover, the FGSM-based target loss function can be defined as:

$$J(x, y; \theta) = cJ_{adv}(x^*, y; \theta) + (1 - c)J_{raw}(x, y; \theta) \tag{9}$$

where c is an equilibrium coefficient that is used to balance both the original and attack samples. As a result, the adversarial examples used in the adversarial learning method can improve the generalization ability of a model by adding a regular term to the loss function. The goal of adversarial training is to minimize the loss function in the worst case.

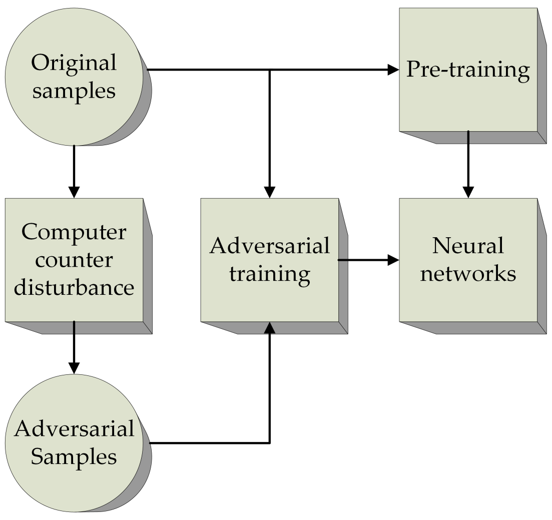


Figure 4. Schematic of adversarial learning.

2.2.2. DRL-Based DDPG Algorithm

During the deep reinforcement learning process, an agent completes its interaction with the environment by perceiving the environment and taking appropriate actions. It performs adaptive iterative optimization according to a reward signal, as shown in Figure 5. An effective approach to describe the DRL-based training process is the Markov decision process (MDP) [34], which is represented by a five-tuple $\langle S, A, R, P, \gamma \rangle$. At each time step, an agent interacts with the environment and makes observations, which comprise the agent's state $s \in S$. Agents then perform the action $a \in A$ to obtain reward R from s to a new state s'. P denotes the environmental model, and it represents the probability distribution of transitioning to a new state. γ is a discount factor used to balance the impact of instantaneous returns and long-term returns on cumulative rewards.

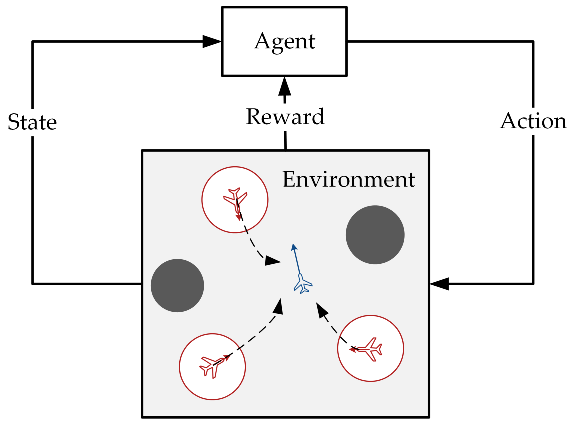


Figure 5. The basic process of deep reinforcement learning.

The deep deterministic policy gradient is an algorithm that combines policy-based actor neural networks with value-based critic neural networks that can be employed for continuous control [28]. The actor online network μ reacts according to the agent's current observation state s_t and generates a reasonable action $a_t = \mu(s_t)$. The critic online network Q is responsible for evaluating the current action and outputting the action value function $Q(s_t, a_t; \theta^Q)$. θ^μ and θ^Q denote the corresponding parameters of an actor online network and a critic online network. In addition, actor target networks μ' and critic target networks Q' are constructed for future updates.

After each decision, a training sample $[s_t, a_t, r_t, s_t']$ for time t is collected in the experience buffer M, which is applied to iteratively improve the agent's strategies, that is in the update optimization phase, a stochastic mini-batch of N arrays of samples of the previous format is extracted for every training time. The critic online network is updated according to the TD-error, which is defined as:

$$L\left(\theta^Q\right) = \mathbb{E}_{s,a,r,\,s'}\left[\left(y - Q^\mu\left(s, a; \theta^Q\right)\right)^2\right] \tag{10}$$

$$y = r + \gamma Q^{\mu'}\left(s', a'; \theta^{Q'}\right)\Big|_{a'=\mu'(s)} \tag{11}$$

Here, $L\left(\theta^Q\right)$ is the loss function of critic networks, y is the target value Q-target, and i represents the sequence of extracted samples. Additionally, the actor online network would be trained by minimizing the following policy gradient:

$$\nabla_{\theta^\mu} J = \mathbb{E}_{s,a\sim M}\left[\nabla_{\theta^\mu}\mu(a|s)\nabla_a Q^\mu\left(s, a; \theta^Q\right)|a = \mu(s)\right] \tag{12}$$

At regular intervals, the soft update approach and update factor τ are used to copy the network parameters to the target network, which can be expressed as:

$$\begin{cases} \theta^{Q'} = \tau\theta^{Q} + (1-\tau)\theta^{Q'} \\ \theta^{\mu'} = \tau\theta^{\mu} + (1-\tau)\theta^{\mu'} \end{cases} \tag{13}$$

2.2.3. Multi-Agent Deep Deterministic Policy Gradient Algorithm

The MADDPG algorithm is an effective DRL algorithm derived from the DDPG algorithm and can be used to address problems with multi-agent strategies. In the MADDPG, each agent has its own actor–critic framework [35]. For a multi-agent system, the observation set consisting of n agents is $\mathrm{x} = (s_1, s_2, \ldots, s_n)$, the action set is $\mathrm{a} = (a_1, a_2, \ldots, a_n)$, and the reward set is $\mathrm{r} = (r_1, r_2, \ldots, r_n)$. For each agent, its observations and actions are denoted as s_i and $a_i = \mu_i(s_i|\theta_i^{\mu})$ at a point in time. The agent's actor online network outputs a policy according to the agent's own observations, and its critic online network estimates a centralized action value function $Q_i^{\mu}(\mathrm{x}, a_1, a_2, \ldots, a_n)$. This function is based on the status and actions of all agents, as depicted in Figure 6.

During each interaction within the environment, an agent will store relevant experiences in the experience buffer. Unlike the DDPG, the N comprehensive learning samples $[\mathrm{x}, a_1, a_2, \ldots, a_n, r_1, r_2, \ldots, r_n, \mathrm{x}']$ in the MADDPG are drawn and spliced from the experience buffer of all agents each time one is trained. For agent i, the critic online network is updated according to:

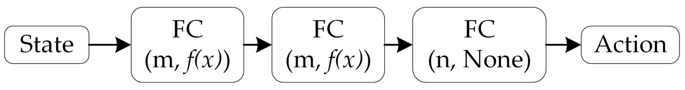
$$L(\theta_i) = \mathbb{E}_{\mathrm{x},a,r,\mathrm{x}'}\left[\left(y_i - Q_i^{\mu}\left(\mathrm{x}, a_1, \ldots, a_n; \theta_i^{Q}\right)\right)^2\right] \tag{14}$$

$$y_i = r_i + \gamma Q_i^{\mu'}\left(\mathrm{x}', a_1', \ldots, a_n'; \theta_i^{Q'}\right)\Big|_{a_i=\mu_i(s_i)} \tag{15}$$

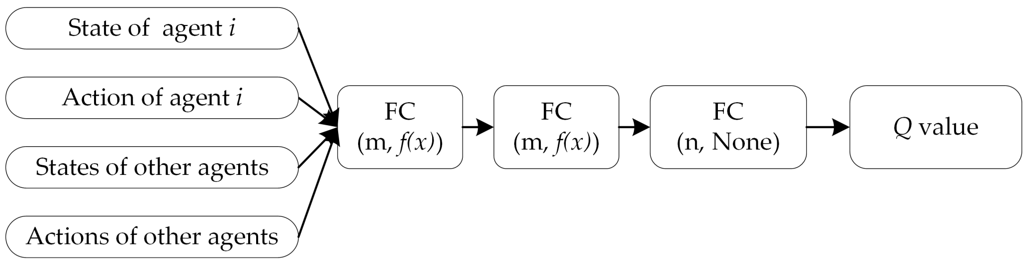
Meanwhile, by minimizing the policy gradient, its actor online network can be optimized. This is expressed as:

$$\nabla_{\theta_i^{\mu}} J = \mathbb{E}_{\mathrm{x},a\sim\mathcal{D}}\left[\nabla_{\theta_i^{\mu}}\mu_i(a_i|s_i)\nabla_{a_i}Q_i^{\mu}\left(\mathrm{x}, a_1, \ldots, a_n; \theta_i^{Q}\right)\Big|_{a_i=\mu_i(s_i)}\right] \tag{16}$$

The MADDPG algorithm also borrows the soft update technique described in Equation (13) from DDPG.



(**a**)



(**b**)

Figure 6. Critic and actor network structures of the MADDPG algorithm. (**a**) Actor and (**b**) critic network structures.

Although agents trained with the MADDPG can achieve good results in some simple environments, the multi-agent system is very sensitive to changes in the training environment, and the convergence strategies obtained by agents are likely to fall into a local optimum, that is when the strategies of other agents change, the agent cannot produce the optimal action strategy. In order to improve the robustness of the strategy, this paper combines the MADDPG and adversarial learning to propose the A2-MADDPG algorithm, which is introduced in Section 3.

3. Proposed Method

This section proposes an approach for realizing control for pursuers and evaders in a game that contains an uncertain environment. There are obstacles in the environment that both pursuers and evaders need to avoid, and when acquiring specific values in the state space, sensors and other devices encounter positioning, sensing, and actuator errors, resulting in inaccurate values, so the environment is uncertain for pursuers and evaders. An MADDPG-based control framework for multi-agent systems is presented, and it includes action, state, space, and specific reward functions. Furthermore, an improved approach called the A2-MADDPG is described. The A2-MADDPG incorporates an adversarial attack trick and adversarial learning into the MADDPG algorithm.

3.1. MADDPG-Based Framework

3.1.1. Actor Space

When addressing DRL-based multi-agent decision-making, state and action spaces must be defined based on the MDP framework. To ensure that mission control is more similar to the real world, UAVs use dual-channel control, that is the force on a UAV is controlled directly. The effects of this force are then applied to the UAV's movement attitude and flight velocity. The action output A^i of a dual-channel thrust UAV thrust can be expressed as:

$$A^i = [F_x^i, F_y^i] \tag{17}$$

where the superscript i represents the sequence number of the UAV in an MAS. F_x^i, F_y^i represent the force on the x and y axes that the UAV received. Therefore, the acceleration can be given by:

$$a^i = \begin{bmatrix} a_x^i \\ a_y^i \end{bmatrix} = \begin{bmatrix} F_x^i \\ F_y^i \end{bmatrix} / m_u \tag{18}$$

where m_u represents the mass of the UAV. The UAV's attitude can then be adjusted when combined with Equation (1).

3.1.2. State Space

The state space of a UAV provides useful information based on an agent's observation model. This is used to help the agent sense its surroundings and make decisions. To help both sides during confrontation training, the state spaces of pursuers and evaders should be presented. As per Section 2.1.2, each pursuer's state information is processed and integrated and includes its position relative to partners $s_{pc}^i = [x_{pc}^l - x_{ps}^i, y_{pc}^l - y_{ps}^i]_{l=1,\dots,i-1,i+1,num\_P}$, evader targets $s_{pe}^i = [x_{pe} - x_{ps}^i, y_{pe} - y_{ps}^i]_{j=1,\dots,num\_E}$, and obstacles $s_{po}^i = [d_1^i, \dots, d_5^i]$, as mentioned in Equation (3). The state space of pursuer i can be defined as:

$$S_p^i = [s_{ps}^i, s_{pc}^i, s_{pe}^i, s_{po}^i] \tag{19}$$

where $s_{ps}^i = [v_{px}^i, v_{py}^i, x_{ps}^i, y_{ps}^i]$ denotes its position and speed based on self-observed information that has not been processed. Similarly, the state space of an evader j can be defined as:

$$S_e^j = [s_{es}^j, s_{ec}^j, s_{ep}^j, s_{eo}^i] \tag{20}$$

where $s_{es}^j=[v_{ex}^j, v_{ey}^j, x_{es}^j, y_{es}^j]$ represents the position and speed of evader j, s_{ec}^j represents its position relative to other evaders, $s_{ec}^j = [x_{ec}^l - x_{es}^j, y_{ec}^l - y_{es}^j]_{l=1,\ldots,j-1,j+1,num\_E}$ $s_{ep}^j = [x_{ep}^i - x_{es}^j, y_{ep}^i - y_{es}^j]_{i=1,\ldots,num\_P}$ represents its position relative to pursuers, and $s_{eo}^j = [d_1^j, \ldots, d_5^j]$ represents sensed obstacles.

3.1.3. Reward Function

In the traditional MADDPG algorithm, formation cooperation cannot be uniformly controlled since each agent has an independent actor and critic network. When a unit successfully hunts down a target, all agents belonging to the formation receive a positive reward regardless of whether the agent was in effective tracking range or played a positive role in the mission. This is contrary to the incentive policy of real pursuit–evasion scenarios.

To address this problem, a reward function based on the team strategy was presented. An agent could receive a positive reward only if it was within a certain distance $\zeta range_{att}$ of the target when the mission terminated. The reward is shaped by three basic elements: (1) distance $r_{distance}^i$: the Euclidean distance is used to judge whether the agent successfully pursued the evader; (2) maneuvering safety r_{safe}^i: the agent is punished if it has collided with obstacles or collaborators; and (3) mission criteria $r_{mission}^i$, $\zeta range_{att}$ are used to judge whether the agent completed the mission. These three subreward functions can be defined as:

$$r_{distance}^i = dis(p_p^i - p_e^j) \tag{21}$$

$$r_{safe}^i = \begin{cases} 1, \ dis(p_p^i - p_o^k) > dis_{obstacle} \ or \ dis(p_p^i - p_p^n)_{n \neq i} > dis_{safe} \\ 0, \ else \end{cases} \tag{22}$$

$$r_{safe}^i = \begin{cases} 1, \ dis(p_p^i - p_e^j) \leq \zeta range_{att} \\ 0, \ else \end{cases} \tag{23}$$

where i is the sequence number of the agent and function $dis(a, b)$ is used to calculate the Euclidean distance of positions a and b. $dis_{obstacle}$ represents the radius of an obstruction, and dis_{safe} represents the minimum safe distance between each pursuer. To summarize, the reward function for a pursuer i can be formulated as:

$$r_p^i = \beta_1 r_{distance}^i + \beta_2 r_{final}^i + \beta_3 r_{safe}^i \tag{24}$$

Three relative gain factors β_{1-3} are introduced, which represent the respective weights of the three rewards or punishments. Among them, β_1 is negative, and both β_2 and β_3 are positive. To train an autonomous evader, a specific reward function was built according to the distance among the pursuer, evaders, and obstacles, and it has weights that are the inverse of the pursuer's reward function.

3.2. A2-MADDPG Algorithm

3.2.1. Adversarial Attack Trick for the Agent

When perceiving the environment in a real scenario, an agent would encounter unavoidable errors due to the detection process, image recognition, signal processing, and satellite position-based parameter measurements. Improving model robustness is of great significance, especially in key intelligent control fields such as UAVs and robots, where tiny errors or noise could lead to immeasurable and undesirable consequences.

To train a robust agent to adapt to measurement errors and other noise in real environments, an adversarial attack trick for the agent itself is proposed. This approach aims to generate random noise in the agent's status, thereby confusing its perception and assisting it in producing a strategy for abnormal conditions [36]. Algorithm 1 summarizes the adversarial attack process, in which inputs are constituted by the action $a_i = \mu_i(s_i|\theta_i^\mu)$ of an actor online network, the action value $Q_i^{\mu'}(\mathrm{x}, a_1, a_2, \ldots, a_n)$ of a critic target network, and that of agent i. Through limited iterations N_a, the state is combined with stochastic

Gaussian noise under the minimum Q value that could be excavated. Algorithm 1, to control the sequence, introduces the pseudocode for an agent's adversarial attack trick.

Algorithm 1 Adversarial attack trick (with the MADDPG).

```
1: StateAttack($Q_i^{\mu'}(x, a_1, a_2, \dots, a_n)$, $\mu_i(s_i|\theta_i^{\mu})$, $s_i$)
2: $a_i = \mu_i(s_i|\theta_i^{\mu})$, $Q_i = Q_i^{\mu'}(x, a_1, a_2, \dots, a_n)$
3: for $i = 1, N_a$ do
4:     $s_{i(noise)} = s_i + Gaussian(0, \sigma_s^2)$
5:     $a_{i(noise)} = \mu_i(s_{i(noise)}|\theta_i^{\mu})$
6:     $Q_{i(noise)} = Q_i^{\mu'}\left(x, a_1, \dots, a_{i(noise)}, \dots, a_n\right)$
7:     if $Q_{i(noise)} < Q_i$ then
8:         $Q_i = Q_{i(noise)}$, $s_i = s_{i(noise)}$
9:     end if
10: end for
11: return $s_{i(noise)}$
```

The action of agent i can be remodeled based on its state after including stochastic Gaussian noise. Similarly, the robustness of the multi-agent intelligent control model could be optimized according to the adversarial attack of all agents by modeling the indeterminacies of the real world.

3.2.2. Adversarial Learning for Cooperators

In addition to uncertain influences from a real environment, an agent is susceptible to strategy changes made by other agents in the overall system [37]. In other words, an agent cannot produce the optimal action strategy to match other agents when those agents change strategies. Our algorithm preprocesses the actions of cooperators using adversarial training techniques, so the agent's strategy is updated based on the worst decisions of other agents. Specifically, as the neural network is updated, the cumulative return of agent i is optimized under the condition that all cooperators use adversarial strategies. The cumulative return of agent i combined with adversarial learning can be formulated as:

$$\begin{aligned} \mathbb{E}_{x\sim\rho^{\mu}}(R_i) &= \underset{a_{j\neq i}^{t}}{min}\mathbb{E}_{x\sim\rho^{\mu}}\left[\textstyle\sum_{t=0}^{T}\gamma^t r_i(x^t, a_1^t, \dots, a_n^t)\,|a_i^t = \mu(s_i^t)\right] \\ &= \mathbb{E}_{x^0\sim\rho^{\mu}}\left[\underset{a_{j\neq i}^{0}}{min}Q_i^{\mu}(x^t, a_1^t, \dots, a_n^0)\,|a_i^0 = \mu(s_i^0)\right] \end{aligned} \tag{25}$$

where ρ^{μ} represents the state distribution. That means x^{t+1} would be influenced by the actions of all agents. Furthermore, the action value Q function could be defined in a recursive form as:

$$Q_i^{\mu}(x, a_1, \cdots, a_n) = r_i(x, a_1, \cdots, a_n) + \gamma\mathbb{E}_{s'}\left[\min_{a'_{j\neq i}} Q_i^{\mu}(x', a'_1, \cdots, a'_n)\Big|_{a'_i=\mu(x'_i)}\right] \tag{26}$$

The single-step gradient descent method was introduced to overcome high computing costs [38]. Using this method, the actions taken by cooperators are those exhibiting a mixed disturbance, and the direction of the disturbance is the orientation in which the Q function is decreasing. To summarize, the update process of the critic online network can be formulated as:

$$L(\theta_i) = \mathbb{E}_{\mathrm{x},a,r,\mathrm{x}'}\left[\left(y_i - Q_i^{\mu}\left(\mathrm{x}, a_1, \ldots a_n; \theta_i^{Q}\right)\right)^2\right] \tag{27}$$

$$y_i = r_i + \gamma Q_i^{\mu'}\left(\mathrm{x}', a_1'^{*}, \ldots, a_i'^{*}, \ldots, a_n'^{*}; \theta^{Q'}\right) \tag{28}$$

$$a_i' = \mu_i'(s_i) \tag{29}$$

$$a'^{*}_{j\neq i} = a_j' + \eta_j \tag{30}$$

$$\eta_{j\neq i} = \arg\min_{\varepsilon_{j\neq i}} Q_i^{\mu'}\left(\mathrm{x}', a_1' + \eta_1, \ldots, a_i' + \eta_i, \ldots, a_n' + \eta_n\right) \tag{31}$$

where $\theta^{Q'}$ represents the critic target network, $a'^{*}_{j\neq i}$ represents the action of other agents in their minimum conditions, and $\eta_{j\neq i}$ represents the disturbance added for agent j. By linearizing the Q function, the parameter η_j is used to denote the gradient direction of $Q_i^{\mu}(\mathrm{x}, a_1, \ldots, a_n; \theta_i^{Q})$ at a_j'. η_j can be replaced with this gradient approximation:

$$\tilde{\eta}_{j\neq i} = -\varepsilon \nabla_{a_j} Q_i^{\mu'}\left(\mathrm{x}', a_1', \ldots, a_n'\right) \tag{32}$$

The critic network structure of the MADDPG combined with adversarial learning is illustrated in Figure 7. When the MADDPG is implemented, adversarial interference must be added to the actions of other agents without requiring the critic network to be remodeled.

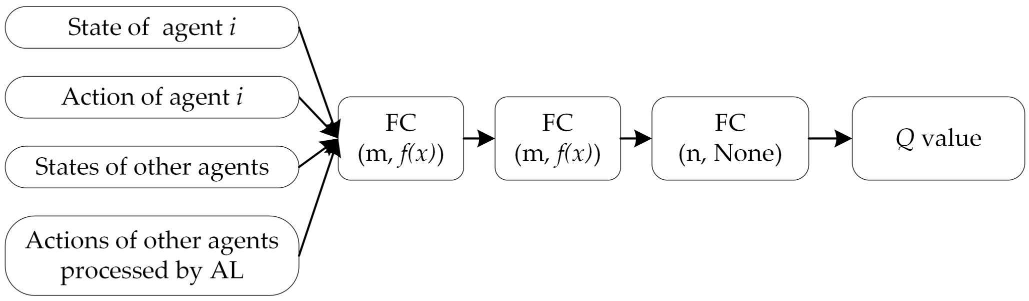


Figure 7. Critic network structures of the MADDPG algorithm combined with AL (the information about the actions of other agents in the input layer is processed by AL).

To summarize, the A2-MADDPG algorithm proposed in this paper employs an adversarial attack trick and adversarial learning to process an agent's state information and other agents' actions during training. This bridges the gap between simulated training and the real world by adding adversarial disturbances. The overall A2-MADDPG algorithm is described in Algorithm 2.

Algorithm 2 Adversarial attack trick and adversarial learning MADDPG (A2-MADDPG) algorithm.

```
1: for $N$ agents, randomly initialize their critic network $Q_i\left(s_i, a_i; \theta_i^Q\right)$ and actor network $\mu_i(s_i|\theta_i^\mu)$
2: synchronize target networks $Q_i'(s_i, a_i; \theta_i^{Q'})$ and $\mu_i'(s_i|\theta_i^{\mu'})$ with $\theta_i^{Q'} \leftarrow \theta_i^Q, \theta_i^{\mu'} \leftarrow \theta_i^\mu$
3: initialize hyperparameters: experience buffer$\mathcal{D}$, mini-batch size $m$, max episode $M$, max step $T$, actor learning rate $l_a$, critic learning rate $l_c$, discount factor $\gamma$, soft update rate $\tau$
4: for $episode = 1, M$ do
5:   reset environment, and receive the initial state x
6:   initialize exploration noise of action $\mathbb{N}_{action}$
7:   for $t = 1, T$ do
8:     $s_{i(noise)} \leftarrow$ StateAttack($Q_i^{\mu'}(\mathrm{x}, a_1, a_2, \ldots, a_n)$, $\mu_i(s_i|\theta_i^\mu)$, $s_i$)
9:     for each agent $i$, select action $a_i = \mu\left(s_{i(noise)}; \theta_i^\mu\right) + \mathbb{N}_{action}$
10:    execute $a_1, \ldots, a_n$, rewards $r_1, \ldots, r_n$, and next state x$'$
11:    store sample $(\mathrm{x}, a_1, \ldots, a_n, r_1, \ldots, r_n, \mathrm{x}')$ in $\mathcal{D}$
12:    for $agenti = 1, n$ do
13:      randomly extract $m$ samples $\left(\mathrm{x}^k, a^k, r^k, \mathrm{x'}^k\right)$
14:      update critic network by Equation (27)
15:      update actor network by:
           $$\nabla_{\theta_i^\mu} J = m^{-1} \sum_j \left[ \nabla_{\theta_i^\mu} \mu_i(a_i|s_i) \nabla_{a_i} Q_i^\mu\left(\mathrm{x}, a_1 \ldots a_i \ldots a_N; \theta_i^Q\right)\Big|_{a_i=\mu_i(s_i), a_{j\neq i}^*=a_j^k+\tilde{\eta}_j} \right]$$
16:    end for
17:    update target networks by $\theta_i^{Q'} \leftarrow \tau\theta_i^Q + (1-\tau)\theta_i^{Q'}$, $\theta_i^{\mu'} \leftarrow \tau\theta_i^\mu + (1-\tau)\theta_i^{\mu'}$
18:  end for
19: end for
```

4. Experiment and Result Analysis

This section describes the simulation's settings and a series of experiments implemented to analyze the effectiveness and performance of the approaches proposed in previous sections.

4.1. Simulation Environment Settings

The experiments were conducted using Pycharm and the gym module on an Ubuntu16.04 system with an Intel i7-6700K CPU, a GeForce1660Ti graphics card, and 16 G of RAM. As shown in Figure 8, the testbed was a square (20 km on a side) in a two-dimensional plane. Each obstacle was assumed to have a round threat area with radius $r_{ob} \in [0.6, 1.3]$ km (black circle). The attack range of the pursuers (red circles with a UAV inside) was set to 1.2 km, which means the mission was considered successful for pursuers when the distance between them and at least one evader (blue UAVs) was within 1.2 km. Table 1 provides the parameters used for the platforms.

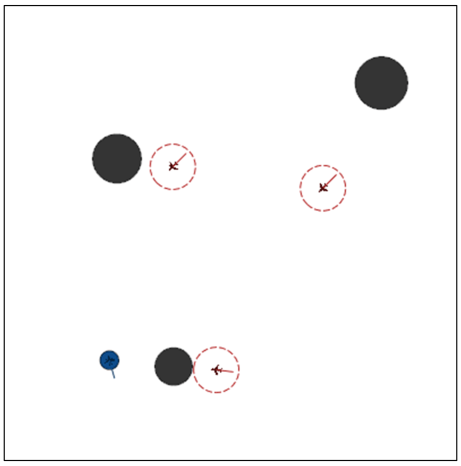


Figure 8. Critic network structures of the MADDPG algorithm combined with AL (the information about actions of other agents in the input layer is processed by AL).

Table 1. The detailed parameter settings of agent platforms in the pursuit–evasion game.

Agent Attributes	Pursuer	Evader
initial velocity	0 km/s	0 km/s
maximum velocity	1 km/s	1.3 km/s
acceleration	0.3 km/s^2	0.4 km/s^2
detect range	1.5 km	1.5 km
attack range	1.2 km	/

In the DRL-based multipursuer framework, a two-layer perceptron model was constructed for the actor and critic networks. Two fully connected $15 \times 64 \times 64 \times 2$ neural networks were created for the actor network and its target. Furthermore, two fully connected $17 \times 64 \times 64 \times 1$ neural networks were created for the critic network and its target. Each round ends when the pursuers capture an evader, a platform collides with an obstacle, or the simulation reaches the maximum number of steps. After each round, the environment was reset, and the next round began. Network training ended when the experience buffer was full, and an Adam optimizer was used to determine the neural network parameters. The hyperparameters of the network are shown in Table 2.

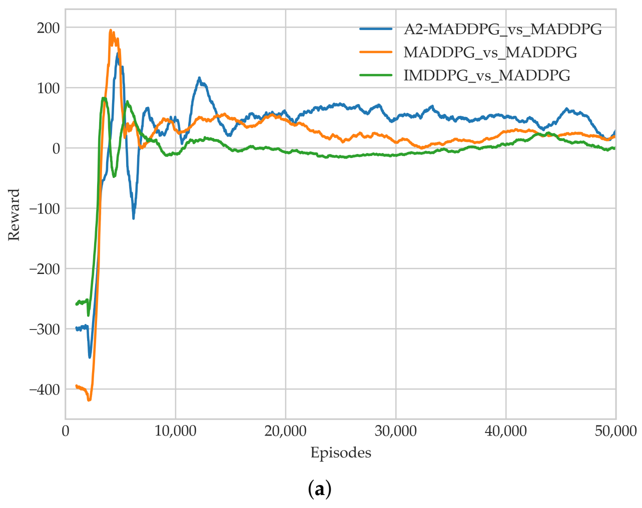
Table 2. The detailed parameter setting of agent platforms in the pursuit–evasion game.

Parameters	Values
Experience replay buffer $\mathcal{D}$	100,000
Mini-batch size m	1024
Max episode M	50,000
Max step T	50
Actor learning rate l_a	0.01
Critic learning rate l_c	0.01
Discount factor γ	0.95
Soft update rate τ	0.01

4.2. Experiment on the Performance of the A2-MADDPG

4.2.1. Performance of Pursuers

To examine the performance of trained pursuers, this study used different algorithms to train them to fight evaders that were trained using the constant MADDPG algorithm from Experiment 1. Specifically, we employed the IMDDPG, MADDPG, and A2-MADDPG algorithms in the MAS of pursuers and present comparative data about the average return values for the last 1000 training episodes in Figure 9.



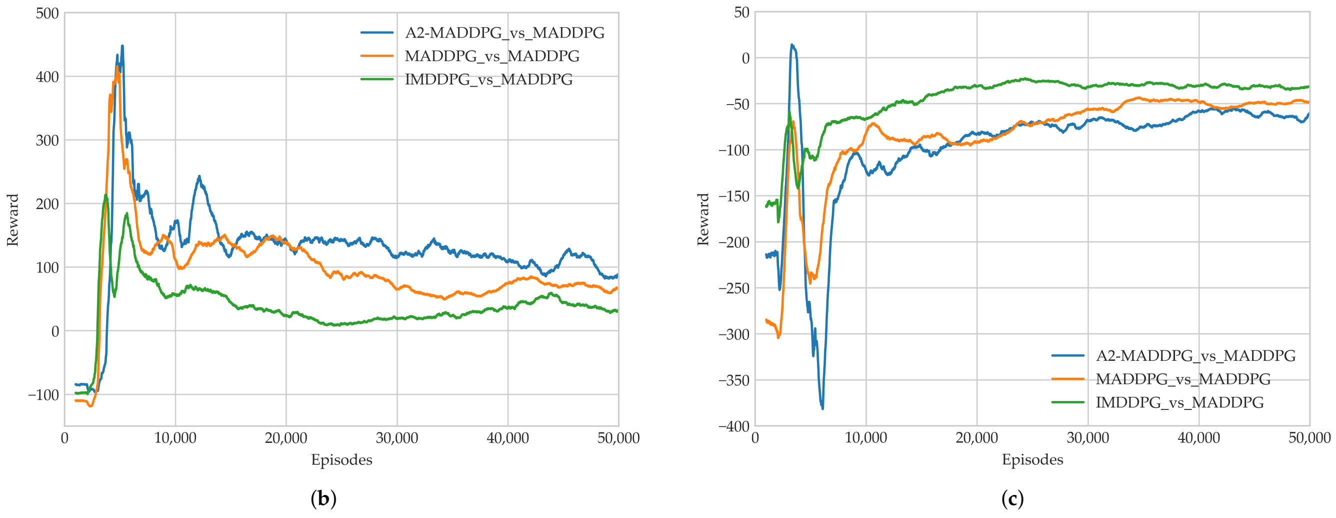


Figure 9. Average rewards in each episode during Training Experiment 1 of (**a**) all agents, (**b**) pursuers, and (**c**) evaders.

As illustrated in Figure 9a, UAVs trained using all three algorithms required roughly 8000 episodes to converge to a steady value with an average reward. In Figure 9b,c, the pursuers and evaders play against each other, and their respective average rewards do not converge to a stable state until about 16,000 episodes. The A2-MADDPG achieved higher convergence approaching the average reward for pursuers and a lower result for evaders. This means the proposed algorithm produced agents that were highly successful at pursuing while preventing targets from escaping, which resulted in lower rewards. To prove the real performance of our algorithm, the average time of first pursuit and the average success of the pursuers in the last 1000 training episodes are recorded in Figure 10.

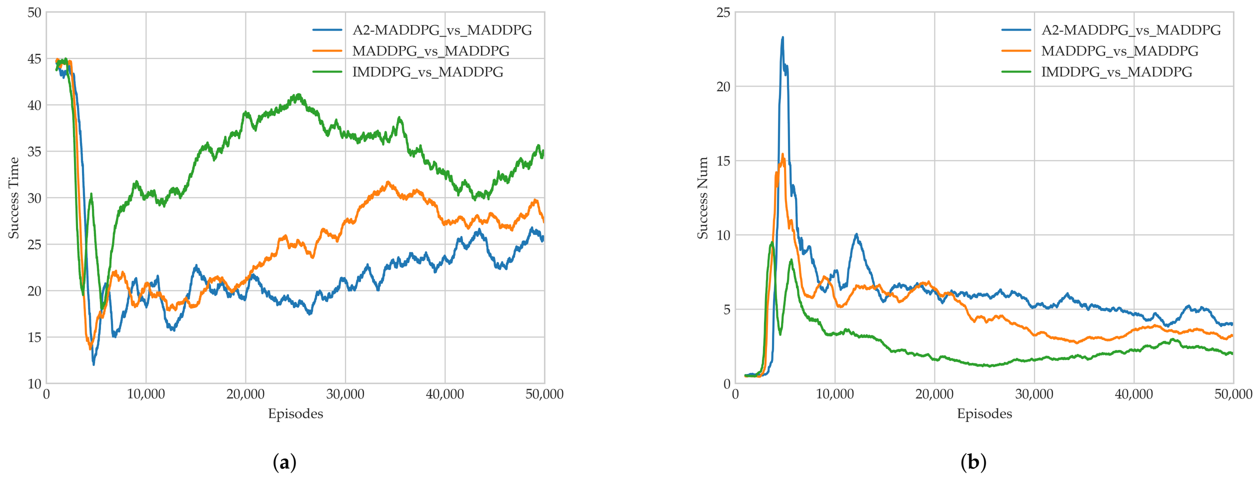


Figure 10. Algorithm performance in Experiment 1. (**a**) Earliest pursuit time and (**b**) successful number of pursuers.

Figure 10a shows that the MAS pursuers can complete their pursuit after being trained using the three DRL algorithms. As the training time increases, A2-MADDPG UAV formations can pursue evaders in a shorter time. This means that the A2-MADDPG model has more advanced co-adjutant siege capabilities. Figure 10b shows the variation of eligible pursuing times produced during training. The pursuers quickly completed a large number of pursuits in each round, but their success gradually decreased as two maneuvering objects were introduced, confronted one another, and stabilized. This means that MADDGP evaders could also make intelligent decisions autonomously to flee. Ultimately, the average eligible pursuing time of A2-MADDPG UAVs was about four per round, which is better than the other two algorithms.

To present the pros and cons of each algorithm in the steady state, the last 10,000 rounds (from 40,000 to 50,000 episodes in Figures 9 and 10) were analyzed. The results are presented in Table 3 and include the average return values for pursuers and evaders, the average and maximum first pursuit time, and the average and maximum successes.

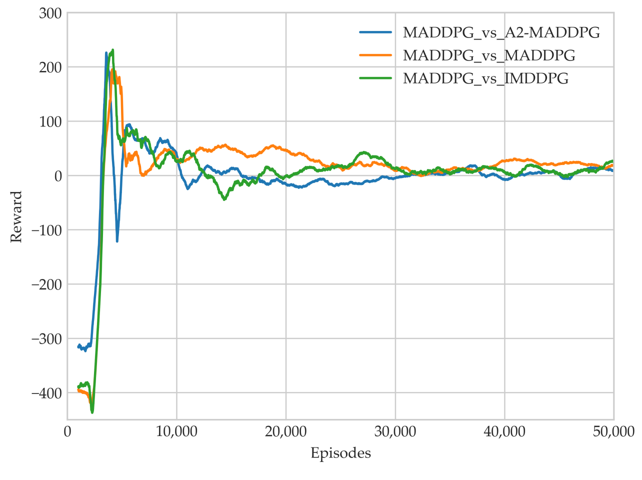
Table 3. Comparison of the training results (sampled from 40,000 to 50,000 episodes in Experiment 1).

Comparison Standard	IMDDPG	MADDPG	A2-MADDPG
Average return value of pursuers	42.15	72.87	103.20
Average return value of evader	−31.65	−50.47	−60.79
Average value of earliest pursuit time	32.17	27.71	24.61
Maximum value of earliest pursuit time	35.662	29.799	26.802
Average successful number of the pursuers	2.42	3.51	4.46
Maximum successful number of the pursuers	2.994	2.508	5.241

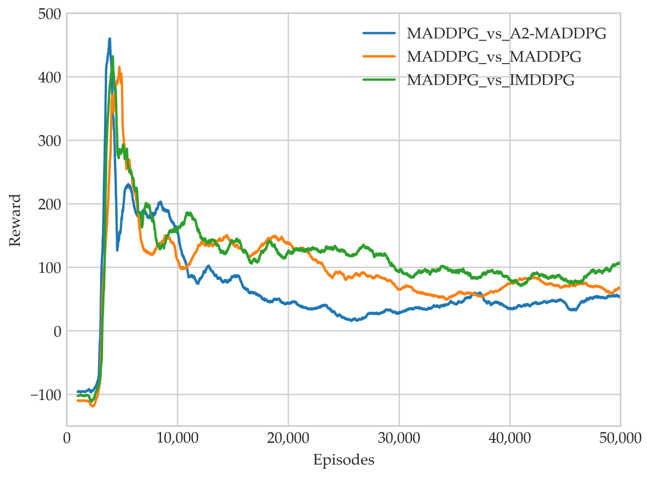
Table 3 shows the performance of each algorithm after stable convergence. The IMDDPG pursuers that used a distributed critic network had an average return value of 42.15, while the MADDPG pursuers had 72.87. Meanwhile, the A2-MADDPG algorithm improved the pursuers' performance, causing the average return value to rise to 103.20. Driven by the shaped reward function, A2-MADDPG pursuers developed efficient strategies, thereby obtaining a higher round reward. The earliest pursuit time indicator reflects the performance of time-efficient decisions that pursuers made. Table 3 shows that for the A2-MADDPG, the earliest pursuit time became shorter, and its average value was reduced from IMDDPG's, 32.17 to 24.61. The maximum earliest pursuit time was reduced from 35.662 to 26.802. The pursuer success number describes how many evaders were successful in each round. This indicator was better for the A2-MADDPG than for the other two algorithms, which indicates that the MAS pursuers based on it had better performance.

4.2.2. Performance of Evaders

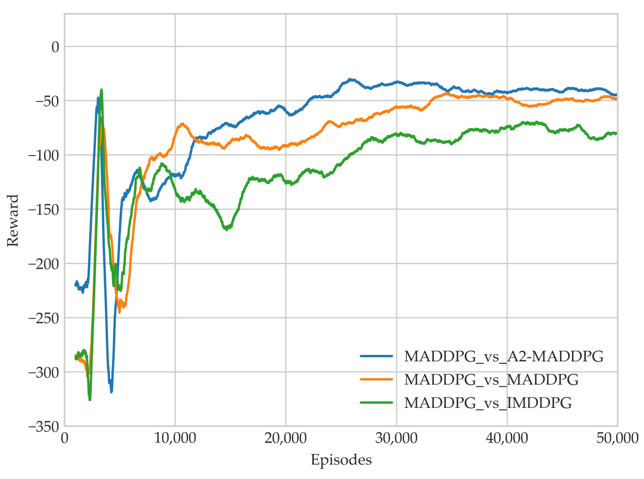
In Experiment 2, we trained evaders using the IMDDPG, MADDPG, and A2-MADDPG to challenge MADDPG pursuers. The average reward results for the last 1000 training episodes are presented in Figure 11.



(**a**)



(**b**)



(**c**)

Figure 11. Average rewards in each episode during Training Experiment 2 of (**a**) all agents, (**b**) pursuers, and (**c**) evaders.

As illustrated in Figure 11c, the evaders achieved the highest round average using the A2-MADDPG, followed by MADDPG, and finally, IMDDPG. This means that A2-MADDPG evaders had a larger advantage during confrontations, which helped them avoid being attacked by pursuers more often.

We recorded the earliest completion time and the eligible tracking time of the pursuers in each round to verify the algorithm's performance, as shown in Figure 12. In Figure 12a, observe that the blue curve has the highest values when the experiment stabilized, that is the first time of pursuing task completion during confrontations between MADDPG pursuers and A2-MADDPG evaders was the longest. This means that A2-MADDPG evaders made efficient decisions when avoiding predators. Figure 12b shows the value of the eligible pursuing time for 50,000 training episodes, from which we observed that the average eligible pursuing time under the confrontation between MADDPG pursuers and A2-MADDPG evaders was the smallest. A2-MADDPG evaders demonstrated better escape strategies. In addition, experimental data from the last 10,000 rounds were analyzed, as presented in Table 4, to control the sequence. The analysis included the average return value of the pursuing UAV formation and evader, the average value of the first time of pursuing task completion, the minimum value of the first time of pursuing task completion, the average eligible pursuing time, and the maximum eligible pursuing time.

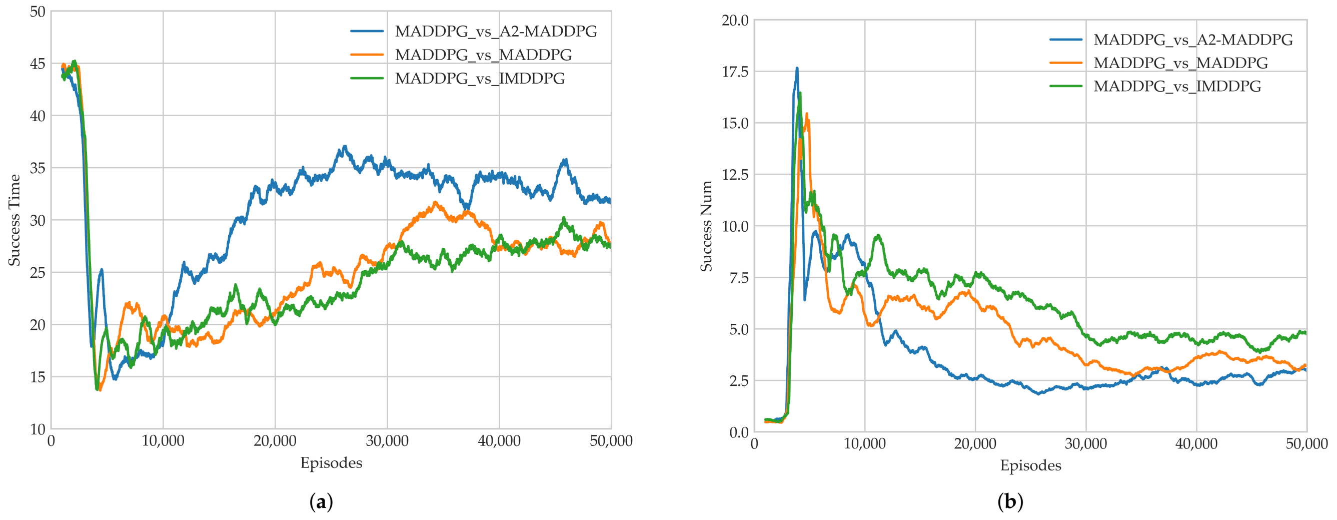


Figure 12. Algorithm performance in Experiment 2. (**a**) Earliest pursuit time and (**b**) number of successful pursuers.

Table 4 gives the parameters of each algorithm after stable convergence. Compared with the IMDDPG, which uses a distributed critic network, centralized MADDPG evaders had higher average return values, longer times being pursued, and lower occurrences of being caught. A2-MADDPG evaders, with superior maneuverability, could generate effective actions to flee from capture by pursuers. That means the proposed A2-MADDPG also optimized the evaders' strategies.

Table 4. Comparison of training results (sampled from 40,000 to 50,000 episodes in Experiment 2).

Comparison Standard	IMDDPG	MADDPG	A2-MADDPG
Average return value of pursuers	85.97	72.87	45.13
Average return value of evader	−76.93	−50.47	−40.53
Average value of earliest pursuit time	27.98	27.70	33.07
Maximum value of earliest pursuit time	30.250	29.799	35.819
Average success number of the pursuers	4.45	3.51	2.66
Maximum success number of the pursuers	4.898	3.928	3.115

4.3. *Experiment on the Effectiveness of the A2-MADDPG*

4.3.1. Effectiveness of Pursuing

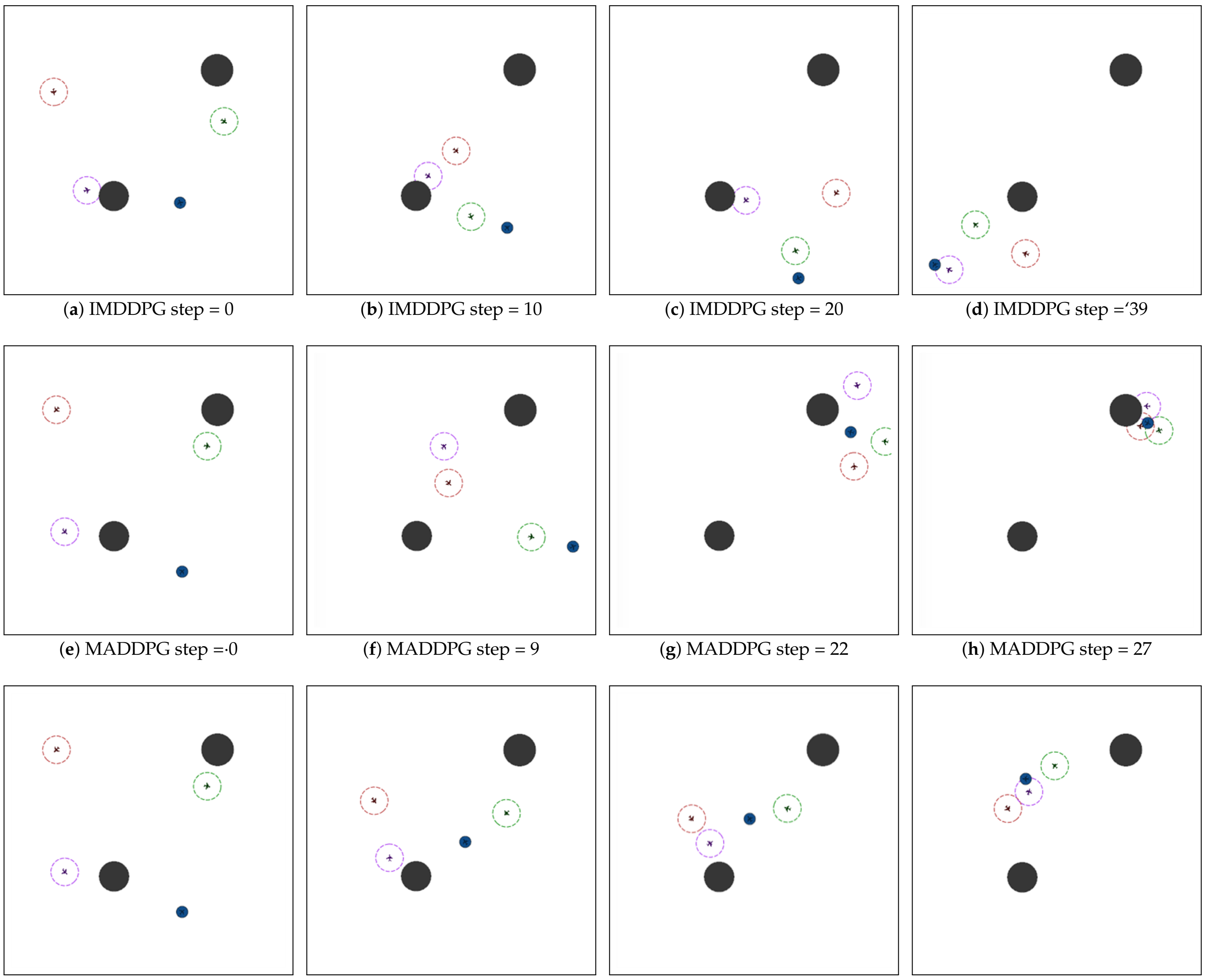
In this section, the algorithm for a specific 3V1 pursuit and evasion confrontation was simulated and analyzed. Table 5 provides the initial positions of the UAVs and obstacles in Experiment 3.

Table 5. Initial positions of UAVs and obstacles in Experiment 3.

Elements	Position
Pursuer 1 (red UAV)	(−7.3, 7.7)
Pursuer 2 (green UAV)	(4.2, 4.0)
Pursuer 3 (purple UAV)	(−8.0, −2.9)
Evader (blue UAV)	(2.0, 8.0)
Obstacle 1 (black circle)	(6.0, 7.0)
Obstacle 2 (black circle)	(3.0, 4.0)

Figure 13 presents the confrontation process of the mission starting from the same initial state. The IMDDPG pursuer formation successfully hit the evader after 39 steps. The

MADDPG pursuer formation generated better encirclement strategies, and it took more time to hunt down the target (27 steps). The A2-MADDPG formation adjusted the direction and speed of each UAV to more effectively reach the more maneuverable evader. It took 13 steps for the pursuers to reach the escape target.


(**a**) IMDDPG step = 0 (**b**) IMDDPG step = 10 (**c**) IMDDPG step = 20 (**d**) IMDDPG step =‘39

(**e**) MADDPG step =·0 (**f**) MADDPG step = 9 (**g**) MADDPG step = 22 (**h**) MADDPG step = 27

(**i**) A2-MADDPG step = 0 (**j**) A2-MADDPG step = 5 (**k**) A2-MADDPG step = 7 (**l**) A2-MADDPG step = 13

Figure 13. Experiment 3: pursuit–evasion game with one MADDPG-based evader and three groups of pursuers driven by different algorithms. The IMDDPG took 39 steps, MADDPG 27 steps, and A2-MADDPG 13 steps to reach the escape target for the first time.

The algorithms were examined in a random test environment with stochastic initial states and environments in Experiment 4. Based on the chase game trained in Section 4.2.1, the results regarding task completion for 1000 test episodes are shown in Table 6. The improved A2-MADDPG had a higher mission success rate of 88.9% for pursuers, which was higher than the success rate of the MADDPG's 75.3% and the IMDDPG's 70.4%. Moreover, the A2-MADDPG pursuers were able to catch the evader in less time. Compared with the MADDPG and IMDDPG, the average value of first time of pursuing completion was 23.698 for the A2-MADDPG, which shows that the A2-MADDPG pursuers can complete pursuits in less time and the effectiveness of their pursuits was enhanced.

Table 6. Effectiveness of pursuing in Experiment 4.

Comparison Standard	**A2-MADDPG**	**MADDPG**	**IMDDPG**
Critic framework of pursuers	Centralized based on AL	Centralized	Distributed
Critic framework of evader	Centralized	Centralized	Centralized
Success rate of pursuers (%)	88.9	75.2	70.4
Average value of the earliest time of pursuing completion	23.698	28.181	32.219

4.3.2. Effectiveness of Escaping

To test the effectiveness of escaping, 1000 random environments were generated in Experiment 5 to compare with the IMDDPG, MADDPG, and A2-MADDPG. The results are presented in Table 7. IMDDPG evaders had a success rate of 17.9%, and A2-MADDPG evaders created better evasion strategies, which resulted in an increased success rate of 31.4%. The maximum value of 30.692 for the average value of the first time of being pursued also proved the effectiveness of the A2-MADDPG for evaders. This means that A2-MADDPG evaders specified more effective escape strategies.

Table 7. Effectiveness of escaping in Experiment 5.

Comparison Standard	**A2-MADDPG**	**MADDPG**	**IMDDPG**
Critic framework of pursuers	Centralized	Centralized	Centralized
Critic framework of evader	Centralized based on AL	Centralized	Distributed
Success rate of pursuers (%)	31.4	24.7	17.9
Average value of the earliest time of being pursued	30.692	28.181	32.219

In summary, compared with the IMDDPG and MADDPG, the evaluation indicators of the A2-MADDPG were significantly better under the same hyperparameter and training environment settings; in the same test environment, the pursuit and escape strategies trained by the A2-MADDPG were obviously more robust and more efficient than those trained by the other two algorithms. Therefore, the A2-MADDPG had a superior performance in the experiments.

5. Conclusions

In this paper, deep reinforcement learning was applied to multi-agent pursuit–evasion decision-making without building a complicated control system, as is commonly performed in traditional approaches. An elaborate MADDPG-based framework was constructed for providing online decision-making schemes and determining the co-adjutant control of multi-agent systems. By introducing adversarial disturbances, an improved A2-MADDPG was proposed that effectively reduced the influence of errors between models and real scenarios. Introducing an adversarial attack trick optimized the robustness of the multi-agent intelligent control model by incorporating adversarial attacks from all agents. An adversarial learning technique was incorporated into our algorithm to overcome the vulnerability of responding to the changes introduced by other agents. This was performed by processing data in the input layer of a critic network. Experimental results showed that the proposed algorithm improved the performance of both types of players in pursuit–evasion games and that the trained agents could devise effective strategies autonomously in confrontational missions.

We intend to expand the pursuit–evasion missions by changing the number of pursuers and evaders in the future and increasing the number of obstacles to make the environment more complex, so as to evaluate the performance, efficiency, and robustness of our algorithms in a more realistic and dynamic space. In addition, we would like to apply the trained robust strategies to drones or unmanned vehicles, so that they can make decisions based on the environmental information obtained by the cameras with an authentic range.

This will accelerate the conversion of this work from virtual digital simulations to real multi-agent systems.

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

Funding: This work was supported by the National Natural Science Foundation of China (Grant No. 62003267), the Natural Science Foundation of Shaanxi Province (Grant No. 2020JQ-220), and the Open Project of Science and Technology on Electronic Information Control Laboratory (Grant No. JS20201100339).

Acknowledgments: The authors would like to acknowledge the National Natural Science Foundation of China (Grant No. 62003267), the Natural Science Foundation of Shaanxi Province (Grant No. 2020JQ-220), and the Open Project of Science and Technology on Electronic Information Control Laboratory (Grant No. JS20201100339) for providing the funding to conduct these experiments. We thank LetPub (www.letpub.com, accessed on 25 July 2021) for its linguistic assistance during the preparation of this manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Chen, Y.; De Luca, G. Technologies Supporting Artificial Intelligence and Robotics Application Development. *J. Artif. Intell. Technol.* **2021**, *1*, 1–8. [CrossRef]
2. Wu, D.; Wan, K.; Gao, X.; Hu, Z. Multiagent Motion Planning Based on Deep Reinforcement Learning in Complex Environments. In Proceedings of the 2021 6th International Conference on Control and Robotics Engineering (ICCRE), Beijing, China, 16–18 April 2021; pp. 123–128. [CrossRef]
3. Czap, H. *Self-Organization and Autonomic Informatics (I)*; IOS Press: Amsterdam, The Netherlands, 2005; Volume 1.
4. Folino, G.; Forestiero, A.; Spezzano, G. A Jxta Based Asynchronous Peer-to-Peer Implementation of Genetic Programming. *J. Softw.* **2006**, *1*, 12–23. [CrossRef]
5. Forestiero, A.; Mastroianni, C.; Papuzzo, G.; Spezzano, G. A proximity-based self-organizing framework for service composition and discovery. In Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, Melbourne, VIC, Australia, 17–20 May 2010; pp. 428–437.
6. Lopez, V.G.; Lewis, F.L.; Wan, Y.; Sanchez, E.N.; Fan, L. Solutions for Multiagent Pursuit-Evasion Games on Communication Graphs: Finite-Time Capture and Asymptotic Behaviors. *IEEE Trans. Autom. Control* **2019**, *65*, 1911–1923. [CrossRef]
7. Zhou, Z.; Xu, H. Mean Field Game and Decentralized Intelligent Adaptive Pursuit Evasion Strategy for Massive Multi-Agent System under Uncertain Environment with Detailed Proof. In Proceedings of the Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications II, Online, 27 April–8 May 2020.
8. Liu, K.; Jia, B.; Chen, G.; Pham, K.; Blasch, E. A real-time orbit SATellites Uncertainty propagation and visualization system using graphics computing unit and multi-threading processing. In Proceedings of the IEEE/AIAA Digital Avionics Systems Conference, Prague, Czech Republic, 13–17 September 2015.
9. Differential Games. A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization. *Math. Gaz.* **1967**, *51*, 80.
10. Unification of differential games, generalized solutions of the Hamilton-Jacobi equations, and a stochastic guide. *Differ. Equ.* **2009**, *45*, 1653–1668. [CrossRef]
11. Chen, J.; Zha, W.; Peng, Z.; Gu, D. Multi-player pursuit–evasion games with one superior evader. *Automatica* **2016**, *71*, 24–32. [CrossRef]
12. Hao, W.; Cheng, L.; Fang, B. An alliance generation algorithm based on modified particle swarm optimization for multiple emotional robots pursuit-evader problem. In Proceedings of the 2014 11th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), Xiamen, China, 19–21 August 2014.
13. Lecun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436. [CrossRef]
14. Botvinick, M.; Ritter, S.; Wang, J.X.; Kurth-Nelson, Z.; Blundell, C.; Hassabis, D. Reinforcement Learning, Fast and Slow. *Trends Cogn. Sci.* **2019**, *23*, 408–422. [CrossRef]
15. Wang, Y.; Dong, L.; Sun, C. Cooperative control for multi-player pursuit–evasion games with reinforcement learning. *Neurocomputing* **2020**, *412*, 101–114. [CrossRef]
16. Xu, G.; Zhao, Y.; Liu, H. Pursuit and evasion game between UVAs based on multi-agent reinforcement learning. In Proceedings of the 2019 Chinese Automation Congress (CAC), Hangzhou, China, 22–24 November 2019; pp. 1261–1266.
17. Park, J.; Lee, J.; Kim, T.; Ahn, I.; Park, J. Co-Evolution of Predator-Prey Ecosystems by Reinforcement Learning Agents. *Entropy* **2021**, *23*, 461. [CrossRef] [PubMed]

18. Gu, S.; Geng, M.; Lan, L. Attention-Based Fault-Tolerant Approach for Multi-Agent Reinforcement Learning Systems. *Entropy* **2021**, *23*, 1133. [CrossRef]
19. Sewak, M. Deep q network (dqn), double dqn, and dueling dqn. In *Deep Reinforcement Learning*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 95–108.
20. Liu, B.; Ye, X.; Dong, X.; Ni, L. Branching improved Deep Q Networks for solving pursuit–evasion strategy solution of spacecraft. *J. Ind. Manag. Optim.* **2017**, *13*.
21. Qadir, M.Z.; Piao, S.; Jiang, H.; Souidi, M.E.H. A novel approach for multi-agent cooperative pursuit to capture grouped evaders. *J. Supercomput.* **2020**, *76*, 3416–3426. [CrossRef]
22. Singh, G.; Lofaro, D.M.; Sofge, D. Pursuit-evasion with Decentralized Robotic Swarm in Continuous State Space and Action Space via Deep Reinforcement Learning. In Proceedings of the 12th International Conference on Agents and Artificial Intelligence, Valletta, Malta, 22–24 February 2020; pp. 226–233.
23. Wang, X.; Xuan, S.; Ke, L. Cooperatively pursuing a target unmanned aerial vehicle by multiple unmanned aerial vehicles based on multiagent reinforcement learning. *Adv. Control Appl. Eng. Ind. Syst.* **2020**, *2*, e27. [CrossRef]
24. Pang, C.; Xu, G.G.; Shan, G.L.; Zhang, Y.P. A new energy efficient management approach for wireless sensor networks in target tracking. *Def. Technol.* **2021**, *17*, 932–947. [CrossRef]
25. Di, K.; Yang, S.; Wang, W.; Yan, F.; Xing, H.; Jiang, J.; Jiang, Y. Optimizing evasive strategies for an evader with imperfect vision capacity. *J. Intell. Robot. Syst.* **2019**, *96*, 419–437. [CrossRef]
26. Lowe, R.; Wu, Y.; Tamar, A.; Harb, J.; Abbeel, P.; Mordatch, I. Multi-agent actor–critic for mixed cooperative-competitive environments. *arXiv* **2017**, arXiv:1706.02275.
27. Zhang, B.H.; Lemoine, B.; Mitchell, M. Mitigating unwanted biases with adversarial learning. In Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society, New Orleans, LA, USA, 2–3 February 2018; pp. 335–340.
28. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. *arXiv* **2015**, arXiv:1509.02971.
29. Li, B.; Gan, Z.; Chen, D.; Sergey Aleksandrovich, D. UAV Maneuvering Target Tracking in Uncertain Environments Based on Deep Reinforcement Learning and Meta-Learning. *Remote Sens.* **2020**, *12*, 3789. [CrossRef]
30. Li, B.; Yang, Z.P.; Chen, D.Q.; Liang, S.Y.; Ma, H. Maneuvering target tracking of UAV based on MN-DDPG and transfer learning. *Def. Technol.* **2021**, *17*, 457–466. [CrossRef]
31. Li, S.; Wu, Y.; Cui, X.; Dong, H.; Fang, F.; Russell, S. Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; Volume 33, pp. 4213–4220.
32. Goodfellow, I.J.; Shlens, J.; Szegedy, C. Explaining and harnessing adversarial examples. *arXiv* **2014**, arXiv:1412.6572.
33. Liu, Y.; Mao, S.; Mei, X.; Yang, T.; Zhao, X. Sensitivity of Adversarial Perturbation in Fast Gradient Sign Method. In Proceedings of the 2019 IEEE Symposium Series on Computational Intelligence (SSCI), Xiamen, China, 6–9 December 2019; pp. 433–436.
34. Papadimitriou, C.H.; Tsitsiklis, J.N. The complexity of Markov decision processes. *Math. Oper. Res.* **1987**, *12*, 441–450. [CrossRef]
35. Guo, H.; Liu, T.; Wang, Y.; Chen, F.; Fan, J. Research on actor–critic reinforcement learning in RoboCup. In Proceedings of the 2006 6th World Congress on Intelligent Control and Automation, Dalian, China, 21–23 June 2006; Volume 2, pp. 9212–9216.
36. Wan, K.; Gao, X.; Hu, Z.; Wu, G. Robust motion control for UAV in dynamic uncertain environments using deep reinforcement learning. *Remote Sens.* **2020**, *12*, 640. [CrossRef]
37. Hernandez-Leal, P.; Kartal, B.; Taylor, M.E. A survey and critique of multiagent deep reinforcement learning. *Auton. Agents Multi-Agent Syst.* **2019**, *33*, 750–797. [CrossRef]
38. Vivek, B.; Babu, R.V. Single-step adversarial training with dropout scheduling. In Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 13–19 June 2020; pp. 947–956.