An Autonomous Attack Decision-Making Method Based on Hierarchical Virtual Bayesian Reinforcement Learning

Dinghan Wang[1]
Northwestern Polytechnical University, Xi'an, 710072, China

Jiandong Zhang[1]
Northwestern Polytechnical University, Xi'an, 710072, China

Qiming Yang*
Northwestern Polytechnical University, Xi'an, 710072, China

Jieling Liu
Xi'an North Electro-optic science and technology defense Co. Lt, Xi'an, 710043, China

Guoqing Shi
Northwestern Polytechnical University, Xi'an, 710072, China

Yaozhong Zhang
Northwestern Polytechnical University, Xi'an, 710072, China

***Abstract*—In response to the challenges of estimating missile launch timing during close-range unmanned autonomous air combat in the future, this paper proposes an autonomous attack decision-making method based on hierarchical virtual Bayesian reinforcement learning (HVBRL). First, a six-degree-of-freedom (6-DOF) high-fidelity aircraft dynamics model along with missile dynamics and guidance rate models are constructed. Second, the HVBRL algorithm is introduced, where the low-level algorithm outputs control parameters and the high-level algorithm generates control commands. Given that the number of missile hits on a target under specific conditions follows a binomial distribution, a simple prior knowledge can be introduced through its conjugate prior, the Beta distribution, to avoid prolonged exploration of ineffective areas. Moreover, carrying only a limited number of missiles and predicting the number of hits by multiple virtual missiles in specific states through a neural network circumvents the computational complexity issue associated with carrying an excessive number of missiles. Finally, this paper presents the low-level training algorithm, the high-level training algorithm, and the high-level self-play training algorithm. Experimental results show that our method significantly reduces the simulation computational complexity. Compared with the Monte Carlo method carrying 1000 missiles, the simulation speed of the high-level training algorithm is increased by 32.75 times, and that of the high-level self-play algorithm is increased by 23 times. Moreover, the estimated missile hit probability with bias can effectively guide the timing of missile launches in close-range air combat, which has significant implications for intelligent autonomous air combat decision-making and operational analysis.**

***Index Terms*— 6-DOF, Bayesian, reinforcement learning, self-play.**

Manuscript received XXXXX 00, 0000; revised XXXXX 00, 0000; accepted XXXXX 00, 0000.
"This work was co-supported by the Natural Science Basic Research Program of Shaanxi (No. 2022JQ-593), the Key R&D Program of Shaanxi Provincial Department of Science and Technology (No. 2022GY-089) and the Aeronautical Science Foundation of China (No. 20220013053005)." *(Corresponding author: Qiming Yang)*. Dinghan Wang and Jiandong Zhang contributed equally to this work and should be considered co-first authors.

Dinghan Wang is with Northwestern Polytechnical University, Xi'an, 710072, China (e-mail: 2018301718@mail.nwpu.edu.cn). Jiandong Zhang is with Northwestern Polytechnical University, Xi'an, 710072, China (e-mail: jdzhang@nwpu.edu.cn). Qiming Yang is with Northwestern Polytechnical University, Xi'an, 710072, China (e-mail: yangqm@nwpu.edu.cn). Jieling Liu is with Xi'an North Electro-optic science and technology defense Co. Lt, Xi'an, 710043, China (e-mail: 784639311@qq.com). Guoqing Shi is with Northwestern Polytechnical University, Xi'an, 710072, China (e-mail: shiguoqing@nwpu.edu.cn). Yaozhong Zhang is with Northwestern Polytechnical University, Xi'an, 710072, China (e-mail: zhang_y_z@nwpu.edu.cn).

I. INTRODUCTION

IN the rapidly evolving field of aerial combat, the adoption of artificial intelligence and, more specifically, reinforcement learning techniques, has opened new vistas for research and development. The intricacies of air-to-air combat, encompassing both traditional engagements and close-range dogfights, demand sophisticated decision-making algorithms that can adapt to dynamic environments and execute complex maneuvers with high precision. This paper delves into the advancements in general reinforcement learning algorithms applied to traditional air combat and close-range engagements, as well as the exploration of hierarchical reinforcement learning (HRL) methods in aerial combat scenarios.

General reinforcement learning algorithms have shown significant promise in addressing the challenges of traditional air combat. Research in this domain has expanded, exploring various aspects of autonomous maneuver decision-making. For instance, Yang et al. [1] and Zhang et al. [2] have contributed to maneuver decision-making models using deep reinforcement learning techniques, albeit facing limitations such as computational demands and applicability in realistic simulations. Further, the exploration of algorithms like DDPG and PPO by Yang et al. [3] and Zhang et al. [4], respectively, highlights the ongoing efforts to improve maneuver decision effectiveness and address the continuous action space challenge in air combat. Notably, Wang et al. [5] and Li et al. [6], [7] have investigated the Pursuer-Evader challenge and autonomous maneuver decision models, showcasing the integration of deep learning techniques and expert knowledge. The research by Zhang et al. [8] and Zhu et al. [9] on missile launch modeling and curriculum learning-based algorithms, respectively, along with Jiang et al. [10] and Hu et al. [11] focusing on BVR combat and close-range engagements, underscore the diverse approaches

being tested in the field. Furthermore, Zhou et al. [12], Kong et al. [13], Wu et al. [14], and Lee et al. [15] have expanded the scope of research to include state-adversarial processes, supervised learning integration, and CUAV tasks, reflecting the breadth of methodologies being applied to enhance air combat strategies.

In the realm of close-range dogfights, the development of intelligent algorithms has revolutionized traditional combat tactics. A HRL method, as proposed by Pope et al. [16], offers a solution to high-dimensional control challenges, while tactical frameworks [17] and improvements in target tracking [18] enhance decision-making capabilities. The effectiveness of transfer learning [19], curriculum learning [20], and deep reinforcement learning models [21], [22] in maneuver control and tactical decision-making further illustrates the advancements in dogfight algorithms, showcasing the potential for AI-driven solutions to dominate this aspect of aerial combat.

The exploration of HRL in air combat represents a significant leap towards mastering the complexity of aerial engagements. By decomposing intricate tasks into simpler, manageable sub-tasks [23], [24], facilitate autonomous decision-making across various combat scenarios. The application of HRL in multi-agent settings by Kong et al. [25] and the development of hierarchical command and control systems by Zhou et al. [26] underscore the potential for enhanced teamwork and adaptability in air combat. Moreover, innovative strategies such as the hierarchical goal-oriented learning by Yuan et al. [27] and the integration of macro-actions and expert knowledge by Wang et al. [28] further augment the strategic decision-making capabilities of unmanned aerial vehicles. The contributions of Chai et al. [29] and Qian et al. [30] in merging HRL with advanced techniques like fictitious self-play and expert insight assimilation highlight the ongoing efforts to refine and optimize decision-making models for air combat, paving the way for future advancements in this challenging domain. Zhang et al. [31] proposed a hierarchical prior-based reinforcement learning approach for loyal wingman task execution for future aerial combat.

In summary, the exploration of reinforcement learning, from general algorithms to hierarchical models, in the context of air combat, showcases a dynamic and evolving field of study. However, existing technologies and methods still face numerous challenges. This paper aims to address several key issues in this field through innovative algorithms and methods:

1) Insufficient simulation fidelity.
2) The difficulty in training due to sparse rewards and limited missile load during missile launch training tasks, leading to convergence issues.
3) The time-space complexity issues brought about by high fidelity and carrying an excessive number of missiles.

To address these issues, this paper proposes a series of innovative strategies:

1) An autonomous attack decision-making method based on HVBRL is proposed, solving the difficulties in convergence as well as the time-space complexity issues.
2) Given that the number of missile hits on targets under specific conditions follows a binomial distribution, introducing simple prior knowledge through its conjugate prior, the Beta distribution, can avoid long periods of exploration in ineffective areas.
3) Furthermore, by carrying only a small number of missiles and predicting the hit count of multiple virtual missiles in specific states through neural networks, the computational complexity issue caused by carrying too many missiles is avoided.
4) Self-play training.

The structure of the article is organized as follows: Section II describes the problem and preparatory work, presenting the 6-DOF dynamic model of the aircraft, the missile dynamics, and the guidance rate model. Section III introduces the classic PPO-clip algorithm. Section IV presents the low-level algorithms as well as the construction methods for the high-level algorithm and the high-level self-play algorithm. Section V presents and analyzes the experimental results. Finally, the research conclusions of this paper are presented.

II. PROBLEM DESCRIPTION AND PRELIMINARY WORK

A. Aircraft Dynamics Model

When constructing a 6-DOF model, it is essential to integrate the structural characteristics of the aircraft, the propulsion system, aerodynamic properties, and control systems to accurately simulate its dynamic behavior. The dynamics analysis involves a comprehensive assessment of the forces acting on the aircraft, including the conversion of engine thrust changes into throttle control input C_a variations, and mapping the changes in aerodynamic forces to the three main control surfaces that govern the aircraft's attitude: the ailerons, elevators, and rudder, corresponding to control inputs C_e, C_r, and C_t, respectively. These control inputs jointly determine the aircraft's speed control, roll, pitch, and yaw behaviors, involving inertial properties, torque generated by rotation, engine thrust and its torque, as well as the effects of aerodynamic forces and moments. In this study, the dynamics equations are presented in a simplified variable form, and due to space limitations, only the basic form is shown here. For details, please refer to the JSBSim documentation [32]. The linear acceleration in the body-fixed coordinate system can be expressed by (1).

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} u \\ v \\ w \end{bmatrix} \times \begin{bmatrix} p \\ q \\ r \end{bmatrix} + \frac{1}{m} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \cdot R_{gb} \quad (1)$$

where u, v, w represent the velocity components along the x, y, z axes of the aircraft body coordinate system; p, q, r are the angular velocities around the corresponding axes; X, Y, Z are the force components acting on the aircraft; m is the mass of the aircraft; g is the acceleration due to gravity; R_{gb} represents the rotation matrix from the ground coordinate system to the body coordinate system. The angular acceleration in the body coordinate system can be obtained from (2).

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = I^{-1} \left(- \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times I \begin{bmatrix} p \\ q \\ r \end{bmatrix} + \begin{bmatrix} L \\ M \\ N \end{bmatrix} \right) \tag{2}$$

where I represents the inertia tensor of the aircraft, and L, M, N are the torque components acting on the x, y, z axes of the aircraft body coordinate system, respectively. By integrating the above equations, the state parameters of the aircraft can be obtained.

B. Missile Dynamics and Guidance Rate Model

In the inertial reference frame, the motion equations of a missile can be described by (3).

$$\begin{cases} \dot{x}(t) = v(t)\cos\theta(t)\cos\phi(t) \\ \dot{y}(t) = v(t)\cos\theta(t)\sin\phi(t) \\ \dot{z}(t) = v(t)\sin\theta(t) \end{cases} \tag{3}$$

where (x, y, z) represent the missile's position coordinates in the inertial reference frame, and (v, θ, ϕ) represent the missile's speed, trajectory pitch angle, and trajectory yaw angle, respectively, all of which are functions of time t. In the ballistic reference frame, the missile's dynamics equations can be expressed by (4).

$$\begin{cases} \dot{v}(t) = g(n_x(t) - \sin\theta(t)) \\ \dot{\phi}(t) = \frac{g}{v(t)} n_y(t)\cos\theta(t) \\ \dot{\theta}(t) = \frac{g}{v(t)}(n_z(t) - \cos\theta(t)) \end{cases} \tag{4}$$

where $n_x(t), n_y(t), n_z(t)$ represent the missile's lateral control overloads in the velocity, yaw, and pitch directions, respectively, which can be calculated using the proportional navigation method; $m(t)$ is the mass of the missile, g is the acceleration due to gravity.

Missile guidance employs the proportional navigation method, and the lateral control overloads in the yaw and pitch directions are defined by (5).

$$\begin{cases} n_y = \frac{Kv}{g}\cos\theta\dot{\beta} \\ n_z = \frac{Kv}{g}\dot{\epsilon} + \cos\theta \end{cases} \tag{5}$$

where $\dot{\beta}$ and $\dot{\epsilon}$ are the rates of change of the line-of-sight angle and the line-of-sight elevation angle, respectively.

The line-of-sight vector, which is the relative position vector $\vec{r}$ between the target and the missile, is determined by the target's position (x_t, y_t, z_t) and the missile's position (x_m, y_m, z_m), with its magnitude defined as $R = \sqrt{r_x^2 + r_y^2 + r_z^2}$. The line-of-sight angle and elevation angle and their rates of change are given by (6).

$$\begin{cases} \dot{\beta} = \frac{\dot{r_y} r_x - r_y \dot{r_x}}{r_x^2 + r_y^2} \\ \dot{\epsilon} = \frac{(r_x^2 + r_y^2)\dot{r_z} - r_z(\dot{r_x} r_x + \dot{r_y} r_y)}{R^2 \sqrt{r_x^2 + r_y^2}} \end{cases} \tag{6}$$

III. PPO-CLIP ALGORITHM

The Proximal Policy Optimization algorithm (PPO) [33] is a reinforcement learning method based on the Actor-Critic framework, which has been widely applied in various scenarios. This algorithm is closely related to the policy gradient method, which employs neural networks to directly approximate the policy function. The main update rule of the policy gradient algorithm is as (7).

$$\nabla \hat{R}_\theta = \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[\mathcal{R}(\tau) \nabla \log \pi_\theta(\tau) \right] \tag{7}$$

where τ represents the trajectory sample of an agent within a cycle, following the probability distribution $\pi_\theta(\tau)$ defined by policy parameters θ. $\mathcal{R}(\tau)$ refers to the cumulative discounted reward of the trajectory. Our goal is to maximize the gradient of the policy parameters with respect to the expected cumulative discounted reward, which is achieved through gradient ascent.

Traditional policy gradient algorithms typically update the policy parameters only once per iteration using trajectory samples, whereas the PPO algorithm combines the advantages of A2C and TRPO algorithms, allowing trajectory samples collected under the old policy to be used for multiple iterations of policy parameter updates. This is achieved by introducing importance sampling and clipping techniques, the latter limiting the difference between the new and old policy functions, especially in terms of the probability ratio.

The importance sampling formula is as (8).

$$\nabla \hat{R}_\theta = \mathbb{E}_{\tau \sim \pi_{\theta_{\text{old}}}(\tau)} \left[\frac{\pi_\theta(\tau)}{\pi_{\theta_{\text{old}}}(\tau)} \mathcal{R}(\tau) \nabla \log \pi_\theta(\tau) \right] \tag{8}$$

where θ_{old} represents the old policy parameters, from which we plan to sample and use these samples to update the policy parameters θ.

Now, we focus on the state-action pairs (s, a) within the trajectory. To prevent inappropriately increasing the probability of choosing an action just because the return is positive, we introduce the state value function $V(s)$ as a baseline, thereby defining the advantage function (9).

$$A(s, a) = Q(s, a) - V(s) \tag{9}$$

This function measures the additional value of taking a certain action in a specific state compared to the average situation. For a more accurate estimation of the advantage, we employ Generalized Advantage Estimation (GAE), which combines Temporal Difference (TD) and Monte Carlo (MC) methods to balance bias and variance.

$$A(s, a) = \delta_t + (\gamma\lambda)\delta_{t+1} + \ldots + (\gamma\lambda)^{T-t+1}\delta_{T-1} \tag{10}$$

where $\delta_t = r_t+\gamma V(s_{t+1})-V(s_t)$ represents the TD error, T is the end of a cycle, and γ and λ are hyperparameters that adjust bias and variance.

By combining importance sampling with the advantage function, we obtain a new expectation formula (11).

$$\mathbb{E}_{(s,a)\sim\pi_{\theta_{\text{old}}}}\left[\frac{\pi_\theta(s|a)}{\pi_{\theta_{\text{old}}}(s|a)}A^{\theta_{\text{old}}}(s,a)\nabla\log\pi_\theta(a|s)\right] \tag{11}$$

In practical applications, we seek to optimize the following objective function (12).

$$L^{\theta_{\text{old}}}(\theta)=\mathbb{E}_{(s,a)\sim\pi_{\theta_{\text{old}}}}\left[\frac{\pi_\theta(s|a)}{\pi_{\theta_{\text{old}}}(s|a)}A^{\theta_{\text{old}}}(s,a)\right] \tag{12}$$

Finally, to ensure the stability of policy updates, we apply a clipping operation to limit the difference between the new and old policies, as (13).

$$L^{\theta_{\text{old}}}_{\text{clip}}(\theta)\approx\sum_{(s,a)}\min\left(\frac{\pi_\theta(s|a)}{\pi_{\theta_{\text{old}}}(s|a)}A^{\theta_{\text{old}}}(s,a),\right.$$
$$\left.\text{clip}\left(\frac{\pi_\theta(s|a)}{\pi_{\theta_{\text{old}}}(s|a)},1-\epsilon,1+\epsilon\right)A^{\theta_{\text{old}}}(s,a)\right) \tag{13}$$

where ϵ is a hyperparameter that controls the gap between the target policy and the baseline policy.

IV. HVBRL ALGORITHM

The HVBRL algorithm is based on the concept of hierarchy, designing both low and high-level algorithms to form a complete decision-making algorithm. The low-level algorithm is responsible for outputting four control quantities: throttle, ailerons, elevators, and rudders, to achieve the desired course, speed, and altitude of the aircraft. The high-level algorithm only needs to output the desired course, speed, altitude, and whether to launch missiles based on the input observational features. The low-level algorithm takes the actions output by the high-level algorithm, excluding the launch action, as part of its input observations to complete the low-level control. Fig. 1 presents the hierarchical network structure of this paper.

The architecture shown in Fig. 1 consists of a low-level processing module identified by the green box on the left and an high-level processing module identified by the yellow box on the right. Three serial modules composed of linear layers, ReLU activation functions, and layer normalization complete the feature extraction of the state space for both low and high-level algorithms. Additionally, the high level requires sequence memory feature processing through a GRU module and a layer normalization module. Subsequently, the features extracted by the high level pass through four linear layers, three of which output logits for partial observational quantities of the low level and sample actions from a categorical distribution to realize action sampling from a multi-discrete action space. The output of another linear layer is used for virtual missile hit number prediction and missile launch decision-making. This output is constrained within the (0,1) interval using the Sigmoid function and multiplied by the maximum virtual missile prediction number N to obtain the predicted number of missiles. Simple prior knowledge is then introduced to complete the prediction of hit probability, and this probability is used to sample from a Bernoulli distribution to determine whether a missile is launched. In addition to receiving part of the high-level actions as part of its own state input, the low level also obtains partial state information from the environment. After feature extraction, it also obtains logits through four linear layers and completes the selection of low-level actions by sampling from a categorical distribution.

A. Low-Level Algorithm

1. *State Space and Action Space*

The construction of state features is crucial for training, as well-chosen features can accelerate training convergence and reduce the parameter space. Considering the complexity of the environment and the combat mission, we have designed a tuple of 10 state variables to represent the aircraft's state space.

$$[\Delta H,\Delta\psi,\Delta V,H_{abs},\phi,\theta,V_x,V_y,V_z,V] \tag{14}$$

where ΔH denotes the difference between the desired and current altitude; $\Delta\psi$ denotes the difference between the desired and current heading; ΔV denotes the difference between the desired and current speed; H_{abs} represents the current altitude of the aircraft; ϕ represents the roll angle; θ represents the pitch angle; V_x, V_y, V_z represent the aircraft's velocity along the x, y, z axes, respectively; V represents the total velocity of the aircraft. The units of ΔH, H_{abs} are in km; $\Delta\psi$, ϕ, θ are in rad; ΔV, V_x, V_y, V_z, V are in Ma.

The low-level action space consists of a tuple of four continuous control variables:

$$[C_a,C_e,C_r,C_t] \tag{15}$$

where C_a represents the throttle control variable, used to control the aircraft's speed; C_e represents the rudder control variable, used to control the aircraft's yaw angle; C_r represents the elevator control variable, used to control the aircraft's pitch angle; C_t represents the aileron control variable, used to control the aircraft's roll angle. The range of values for each control variable is shown in Table I. To balance computational complexity and simulation fidelity, this paper discretizes the continuous action space into a multi-discrete action space of size 50 for each continuous action.

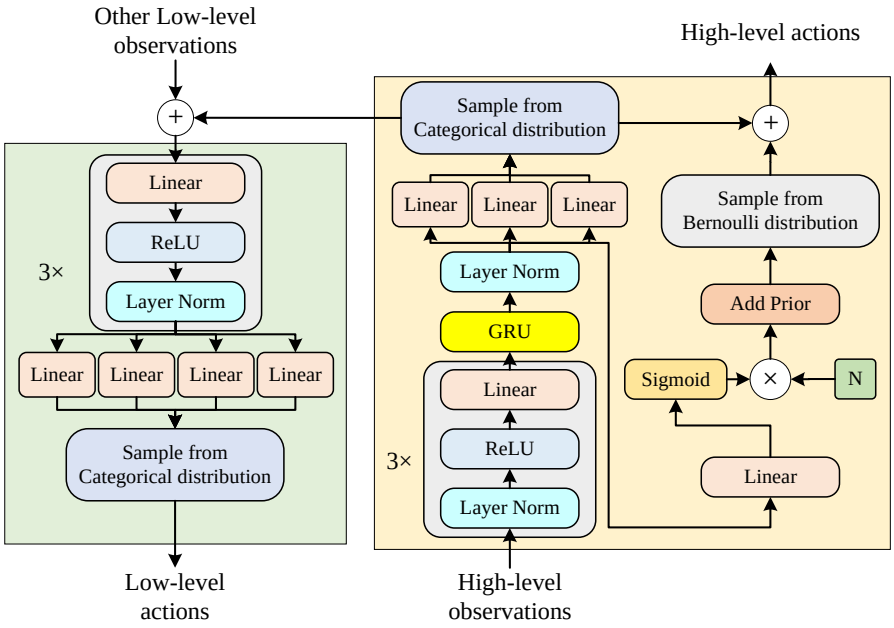

Fig. 1. Hierarchical network control structure.

TABLE I
Range of control variables.

Parameter	Range	Parameter	Range
C_a	[0, 1]	C_r	[-1, 1]
C_e	[-1, 1]	C_t	[-1, 1]

2. Reward Function and Termination Conditions

We expect the aircraft to fly according to the desired heading, altitude, and speed, and we limit the roll angle to avoid stalling and losing altitude due to excessive roll. Therefore, we defined the following four reward functions.

$$\begin{cases} R_\psi = e^{-(\frac{\Delta\psi}{5})^2} \\ R_H = e^{-(10\Delta H)^2} \\ R_V = e^{-(\frac{\Delta V}{20})^2} \\ R_\phi = e^{-(\frac{\phi}{0.3})^2} \end{cases} \tag{16}$$

These four reward functions are Gaussian, limiting the reward value to the (0,1] interval. The different standard deviations of the Gaussian reward functions represent different tolerances for different indicators; a smaller standard deviation indicates a lower tolerance and a higher precision requirement. To quickly meet the requirements for heading, altitude, speed, and roll angle during training, we use the geometric mean of these four rewards as the overall measure of these indicators.

$$R_C = (R_\psi \cdot R_H \cdot R_V \cdot R_\phi)^{\frac{1}{4}} \tag{17}$$

Additionally, if the aircraft descends below the dangerous altitude of 2km, a reward of $R_{PH} = -1$ is given; if the aircraft descends to the termination altitude of 1km, a reward of $R_{PH} = -2$ is given.

Based on the characteristics of the mission, we define the following seven conditions for ending an episode.

1) Failure to reach the desired heading, speed, and altitude within the specified simulation steps.
2) The aircraft descends to the termination altitude of 1km.
3) Operation exceeds the time limit: 10^3 simulation steps.
4) The aircraft's altitude exceeds 10^5 km.
5) The aircraft's p, q, r rotational angular velocities exceed 10^3 rad/s.
6) The aircraft's speed exceeds 100 Ma.
7) The aircraft's acceleration exceeds 20g.

B. High-level Algorithm

In Bayesian statistics, conjugate prior distributions refer to the scenario where the prior distribution and the likelihood function satisfy certain conditions, resulting in a posterior distribution that belongs to the same family as the prior distribution. Specifically, for a prior distribution $p(\theta)$ with parameter θ and a likelihood function $p(x|\theta)$, their product is given by (18).

$$p(\theta)p(x|\theta) = k(\theta) \cdot g(x) \tag{18}$$

where $k(\theta)$ is a function independent of x, and $g(x)$ is a function independent of θ. This implies that the product of the prior distribution and the likelihood function can be decomposed into a product of a function $k(\theta)$ that is independent of x and a function $g(x)$ that is independent of θ. When this condition is met, the posterior distribution $p(\theta|x)$ will belong to the same family as the prior distribution $p(\theta)$, making Bayesian inference more convenient. We can obtain the parameters of the

posterior distribution by updating the parameters of the prior distribution without having to recalculate the entire distribution.

The missile hit probability $\theta(s)$ can be estimated through live-fire tests; however, the cost of conducting field tests is substantial. This paper proposes a method for automatically simulating missile hit probabilities through simulation. Specifically, in a high-fidelity simulation environment, the probability of hitting m out of N missiles launched in a specific state s is given by (19).

$$B(m|N,\theta(s)) = \binom{N}{m}\theta(s)^m(1-\theta(s))^{N-m} \tag{19}$$

To satisfy the conjugate prior condition, the prior distribution of missile hit probability $\theta(s)$ should be of the same distribution family as the likelihood function. Therefore, the prior distribution of $\theta(s)$ is chosen to be a Beta distribution.

$$Beta(\theta(s)|a,E) = \frac{\Gamma(E)}{\Gamma(a)\Gamma(E-a)}\theta(s)^{a-1}(1-\theta(s))^{E-a-1} \tag{20}$$

where E, a represent the belief based on past experience that out of E launched missiles, a will hit. The coefficient $\frac{\Gamma(E)}{\Gamma(a)\Gamma(E-a)}$ satisfies the normalization requirement.

$$\int_0^1 Beta(\theta(s)|a,E)d\theta(s) = 1 \tag{21}$$

The expected value of the prior distribution of missile hit probability $\theta(s)$ is shown as (22).

$$\mathbb{E}[\theta(s)] = \frac{a}{E} \tag{22}$$

Multiplying the prior distribution with the likelihood function and normalizing, we obtain the posterior distribution of missile hit probability as (23).

$$Beta(\theta(s)|m,a,E,N) = \frac{\Gamma(N+E)}{\Gamma(m+a)\Gamma(N+E-m-a)}\cdot \theta(s)^{m+a-1}(1-\theta(s))^{N+E-m-a-1} \tag{23}$$

According to the Bayesian sum and product rules, the distribution of $\theta(s)$ adopts the posterior distribution.

$$p(x=1|\mathcal{X}) = \int_0^1 p(x=1|\theta(s))p(\theta(s)|\mathcal{X})d\theta(s) = \mathbb{E}[\theta(s)|\mathcal{X}] \tag{24}$$

where x follows a Bernoulli distribution, taking the value 1 when hitting the target, and $\mathcal{X}$ represents the data sample. Thus, the Bayesian estimate of $\theta(s)$, $\hat{\theta(s)}$, is shown as (25).

$$\hat{\theta(s)} = \frac{m+a}{N+E} \tag{25}$$

By repeatedly updating $\hat{\theta(s)}$ through multiple simulation firings, it will converge to the true missile hit probability in state s.

However, due to limited computing resources, carrying too many missiles in the simulation process drastically increases the computational time complexity and space complexity.

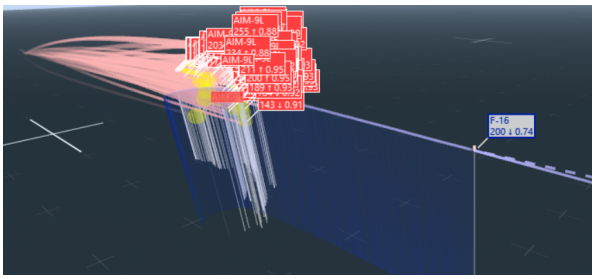

Fig. 2. Simulation with 1000 missiles.

Moreover, strictly controlling repeated experiments in the same state can affect the progress of combat operations. Therefore, we use a neural network to predict the number of missile hits in state s.

$$\hat{m} = f_\phi(s) \tag{26}$$

where ϕ denotes network parameters. $\hat{\theta(s)}$ becomes $\hat{\theta_\phi(s)}$.

$$\hat{\theta_\phi(s)} = \frac{\hat{m}+a}{N+E} \tag{27}$$

where N is the assumed total number of missiles launched, while the actual number of missiles carried in the simulation $n \ll N$. It is assumed that the missiles are independent of each other. Including past experience and the missile firing action a predicted by the neural network.

$$a \sim Bern(\hat{\theta_\phi(s)}) \tag{28}$$

Sampling from this action distribution and interacting with the environment allows for the update of the network through reinforcement learning algorithms to improve its prediction accuracy.

However, using a policy network to directly output missile hit probability

$$\theta(s) = \pi_\phi(s) \tag{29}$$

leads to excessive exploration of meaningless areas in the early stages, such as areas beyond the missile attack range, causing sparse rewards and slow training convergence.

1. *State Space and Action Space*

The construction of state features is crucial for training, as well-defined features can accelerate training convergence and reduce the parameter size. Considering the complexity of the environment and combat missions, we have designed a tuple consisting of 17 state variables to represent the state space.

$$[\Delta H_m, \Delta H_r, \Delta V_m, \Delta V_r, H_{abs}, D_r, D_m, AO_r, TA_r, AO_m, TA_m, \phi, \theta, V_x, V_y, V_z, V] \tag{30}$$

where ΔH_m represents the altitude difference between the missile and the aircraft; ΔH_r represents the altitude difference between opposing aircraft; ΔV_m represents

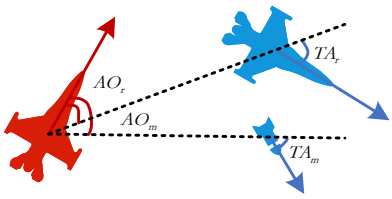

Fig. 3. Schematic diagram of AO&TA.

TABLE II
Range of discrete control variables.

Parameter	Value Range
ΔH_d	{-0.2, -0.1, 0, 0.1, 0.2}
$\Delta\psi_d$	$\{-\frac{\pi}{6}, -\frac{\pi}{12}, 0, \frac{\pi}{12}, \frac{\pi}{6}\}$
ΔV_d	{-0.2, -0.1, 0, 0.1, 0.2}
F	{True, False}

the velocity difference between the aircraft's axial direction and the missile's axial direction; ΔV_r represents the velocity difference in the axial direction between opposing aircraft; D_r represents the distance between opposing aircraft; D_m represents the distance between the aircraft and the missile; AO_r, TA_r, AO_m, TA_m represent the angles between opposing aircraft and the missile, as shown in Fig. 3; ϕ represents the aircraft's roll angle; θ represents the aircraft's pitch angle. The dimensions of $\Delta H_m, \Delta H_r, D_r, D_m$ are in kilometers; AO_r, TA_r, AO_m, TA_m, ϕ, θ are in radians; and ΔV_m, ΔV_r are in Ma.

The action space of the high-level control model consists of a tuple of discrete values controlling the aircraft's heading, velocity, altitude, and missile launch.

$$[\Delta H_d, \Delta\psi_d, \Delta V_d, F] \tag{31}$$

where ΔH_d represents the altitude error; $\Delta\psi_d$ represents the heading error; ΔV_d represents the velocity error; F indicates whether a missile is launched. The range of values for each control variable is shown in Table II.

2. Reward Function and Termination Conditions

The relative situation reward for opposing aircraft is mainly characterized by three key variables: AO_r, TA_r, R_r. The situational reward related to angles AO_r, TA_r is defined as (32).

$$\begin{aligned} f(AO_r, TA_r) = 1 + \frac{1}{13.05 \times |AO_r| + 2} + \\ \min\left(\frac{\tan^{-1}\left(1 - \max(0.64 \times |TA_r|, 10^{-4})\right)}{2\pi}, 0\right) \end{aligned} \tag{32}$$

where the influence of AO_r is adjusted through a denominator term, making the reward more sensitive to its changes.

The situational reward related to the distance R_r is defined as (33).

$$\begin{aligned} f(R_r) =& 1 \cdot (R_r < 4) + (R_r \geq 4) \cdot \\ & \text{clip}\left(-0.03 \cdot {R_r}^2 + 0.28 \cdot R_r + 0.42, 0, 1\right) + \\ & \text{clip}\left(\exp\left(-0.14 \cdot R_r\right), 0, 0.4\right) \end{aligned} \tag{33}$$

This reward is a piecewise function that provides a fixed reward when R_r is less than 4km and calculates the reward based on a quadratic and exponential function of R_r when R_r is greater than or equal to 4, aiming to reward closer distances in line with close-range combat requirements. The $clip$ function limits the first term to an interval with the second and third terms as the minimum and maximum values, respectively.

In addition to situational rewards, when conducting high-level self-play training, if there is a missile warning, we aim to guide the aircraft to autonomously avoid the missile through rewards. Empirically, when the missile has sufficient energy and is head-on with the aircraft, it will be difficult for the aircraft to evade. Therefore, the reward function is constructed based on the decrease in missile velocity and the angle between the missile and aircraft velocity vectors. A negative reward is given if the missile's velocity decreases and the angle with the aircraft is large; a positive reward is given if the angle is small. Continuous rewards guide the aircraft to avoid missiles.

Let the current missile velocity be V_m, the missile velocity at the previous moment be V'_m, and the angle between the missile velocity vector and the aircraft velocity vector be α. When $\cos\alpha < 0$, the reward is

$$R_m = \frac{\alpha}{max(V_m - V'_m, 0) + 1} \tag{34}$$

When $\cos\alpha \geq 0$, the reward is

$$R_m = \alpha \times \max(V_m - V'_m, 0) \tag{35}$$

Furthermore, we define a shooting reward of -10 to avoid high-frequency shooting

$$R_{shoot} = -10 \tag{36}$$

Aircraft crash/being shot down reward

$$R_{crash} = -200 \tag{37}$$

and missile hitting the target reward

$$R_{hit} = 200 \tag{38}$$

According to the characteristics of the mission, in addition to the low-level termination conditions, the round should end when the aircraft is shot down or the missile hits the target.

C. Task Training Algorithms

1. Low-Level Training Algorithm

The low-level training algorithm is related to the aircraft's basic control variables and is responsible for directly controlling the flight to achieve desired headings,

speeds, and altitudes. In Algorithm 1, Φ_{ego} represents the parameters of the actor network, and Θ represents the parameters of the critic network. Both are initialized orthogonally, and the actor network is configured with a smaller gain to enhance exploration of the action space. $\mathcal{D}$ denotes the experience buffer, storing experience data generated by policy rollouts. $\mathcal{T}$ denotes the total number of steps required for the entire training. $\mathcal{B}$ denotes the batch size, and b denotes the mini-batch update size. $\mathcal{N}$ represents the number of vector environments. h, ψ, and v respectively denote altitude, heading, and speed. s_t, a_t, $\log \pi_{\Phi_{ego}}(a_t|s_t)$, $v_\Theta(s_t)$, d_t, and r_{t+1} respectively denote the state, action, logarithm of action probability, state value, termination status, and reward at time $t+1$. A and G respectively represent advantage and return. The general definition of the return G is the sum of discounted rewards, which is,

$$G = r_t + \gamma \times r_{t+1} + \gamma^2 \times r_{t+2} + ... + \gamma^{T-t} \times r_T \quad (39)$$

where T is the termination step. When we use GAE, the return G is usually calculated at the end of an episode, after calculating the advantage A for each step. The return is then obtained by adding the advantage to the state value, i.e.,

$$G(t) = V(t) + A(t) \quad (40)$$

To fully utilize the experience data, the network is updated for $\mathcal{E}$ epochs during updates, and the experience data are shuffled during batch updates.

Algorithm 1: Low-level algorithm

Initialize parameters of actor Φ_{ego} and critic Θ orthogonally;
Initialize the buffer $\mathcal{D}$;
Reset the vector environment;
for $n = 1$ **to** $\mathcal{T}/\mathcal{B}$ **do**
  for $t = 1$ **to** $\mathcal{B}/\mathcal{N}$ **do**
    Change desired h, ψ, v randomly at fixed time intervals;
    Get s_t, d_t;
    Get a_t, $\log \pi_{\Phi_{ego}}(a_t|s_t)$, $v_\Theta(s_t)$ from actor and critic network;
    Execute one step in the vector environment;
    Store $\{s_t, a_t, \log \pi_{\Phi_{ego}}(a_t|s_t), v_\Theta(s_t), r_{t+1}, d_t\}$ into $\mathcal{D}$;
  Calculate advantages A and returns G and store them into $\mathcal{D}$;
  for $e = 1$ **to** $\mathcal{E}$ **do**
    Shuffle $\mathcal{D}$;
    for $i = 1$ **to** $\mathcal{B}/b$ **do**
      Calculate policy loss, value loss and entropy loss;
      Update Φ_{ego}, Θ;

2. High-Level Training Algorithm

The high-level training algorithm is responsible for issuing desired commands to the low level and also handles missile control. When executing the high-level training algorithm, the low-level model $\mathcal{M}_{low}$ should be set to evaluation mode, i.e., the parameters of the low-level model are not changed, only the high-level network is trained. Unlike the low-level model, the high-level algorithm uses a GRU module to learn sequence information, hence the experiences need to be stored in order and cannot be shuffled. The execution process is as shown in Algorithm 2.

Algorithm 2: High-level algorithm

 # Initialize parameters and reset envs
Load low-level model $\mathcal{M}_{low}$ and set it to eval mode;
for $n = 1$ **to** $\mathcal{T}/\mathcal{B}$ **do**
  for $t = 1$ **to** $\mathcal{B}/\mathcal{N}$ **do**
    # Get high-level actions
    # Transfer actions into low-level states
    $\mathcal{M}_{low}$ takes one step to update the vector env;
    # Store high-level experience
  Store A and G into $\mathcal{D}$;
  for $e = 1$ **to** $\mathcal{E}$ **do**
    for $i = 1$ **to** $\mathcal{B}/b$ **do**
      # Update parameters of high-level models

3. High-Level Self-Play Training Algorithm

The high-level self-play training algorithm utilizes self-play techniques to simulate the complex adversarial process of aerial combat under realistic conditions. This study introduces a policy pool $\mathcal{P}$ for storing historical policy models. Strategies are uniformly sampled from $\mathcal{P}$ to approximate the average strategy, with the latest policy model serving as the best response strategy. The execution process is as shown in Algorithm 3.

V. EXPERIMENTAL RESULTS ANALYSIS

A. Experimental Parameter Settings

The experimental hyperparameters are presented in Table III.

The hardware and software platforms are presented in Table IV, V.

The other experimental parameters are set as follows.

1) By default, 13 missiles are expected to hit 3 targets.
2) When the distance is less than 12,000m, it is anticipated that 6 missiles will hit.
3) When the distance is less than 8,000m, it is anticipated that 10 missiles will hit.

Algorithm 3: High-level self-play algorithm

Load low-level model $\mathcal{M}_{low}$ and set it to eval mode;
Initialize parameters of high-level actors Φ_{ego} , Φ_{enm} and critic Θ orthogonally;
Initialize the buffer $\mathcal{D}$ and policy pool $\mathcal{P}$;
Reset the vector environment;
for $n = 1$ **to** $\mathcal{T}/\mathcal{B}$ **do**
  for $t = 1$ **to** $\mathcal{B}/\mathcal{N}$ **do**
    # Get high-level actions
    # Transfer actions into low-level states
    $\mathcal{M}_{low}$ takes one step to update the vector env;
    # Store high-level experience
  Store A and G into $\mathcal{D}$;
  for $e = 1$ **to** $\mathcal{E}$ **do**
    for $i = 1$ **to** $\mathcal{B}/b$ **do**
      # Update parameters of high-level models
  Save critic model;
  Save actor model with parameters Φ_{ego} as best response;
  Sample enemy strategy uniformly from $\mathcal{P}$ as average strategy, and assign its parameters to Φ_{enm};

TABLE III
Hyperparameters configuration.

Parameter	Value	Parameter	Value
$\mathcal{E}$	4	seed	1024
$\mathcal{B}$	3000	clip_param	0.2
λ	0.95	γ	0.99
$\mathcal{N}$	4	$\mathcal{T}$	5×10^7
b	300	N	200
gain_actor	0.01	gain_critic	1
mlp_hidden_size	128×128	entropy_coef	10^{-3}
gru_hidden_size	128	max_grad_norm	2
launch_threshold	0.5	explosion radius	300

TABLE IV
Hardware platform.

Hardware	Specification
CPU	Intel i7-10700KF
GPU	Nvidia RTX 3080
Memory	16GB
GPU Memory	8GB

TABLE V
Software platform.

Work	Platform
Algorithm Implementation	Python run in PyCharm
Experiment Visualization	Tacview

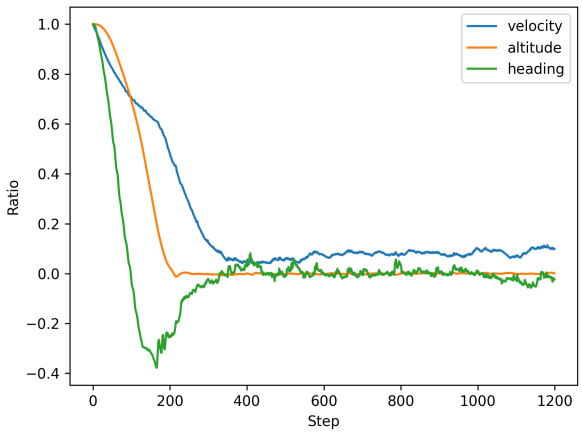

Fig. 4. Performance of low-level algorithm.

4) When the angle is less than 50 degrees, it is anticipated that 7 missiles will hit.
5) When the angle is less than 25 degrees, it is anticipated that 10 missiles will hit.
6) The maximum attack angle is 50°.
7) The maximum attack distance is 15,000m.
8) The minimum attack interval is 120 steps.

B. Performance of the Low-Level Algorithm

Our experiments set expected values for three fundamental control indicators: heading, speed, and altitude. The performance of these three control indicators is evaluated simultaneously in the same experimental episode. Specifically, the difference between the expected heading and the initial heading is set to 60°, the difference between the expected speed and the initial speed is set to 218 m/s, and the difference between the expected altitude and the initial altitude is set to 1904 m. Since the three indicators have different units and magnitudes, in order to clearly display the control performance of the three indicators of the underlying model in one image, we use the difference between the expected indicators and the actual indicators at the initial moment as the baseline, and normalize by dividing subsequent differences by this baseline difference. The results under the experiment with a total of 1200 steps are shown in Fig. 4. As can be seen, even when the difference between the expected value and the initial value is large, the control performance of the underlying model is still quite excellent.

C. Performance of the High-Level Algorithm in Three Specific Scenarios

To evaluate the performance and generalizability of the high-level algorithm in fixed scenarios (red curve in Fig. 5), during the training process, the opponent (Blue aircraft) strategy was set to not carry missiles, to perform tail-chase escape maneuvers immediately upon missile warning, and to initially face towards us (Red

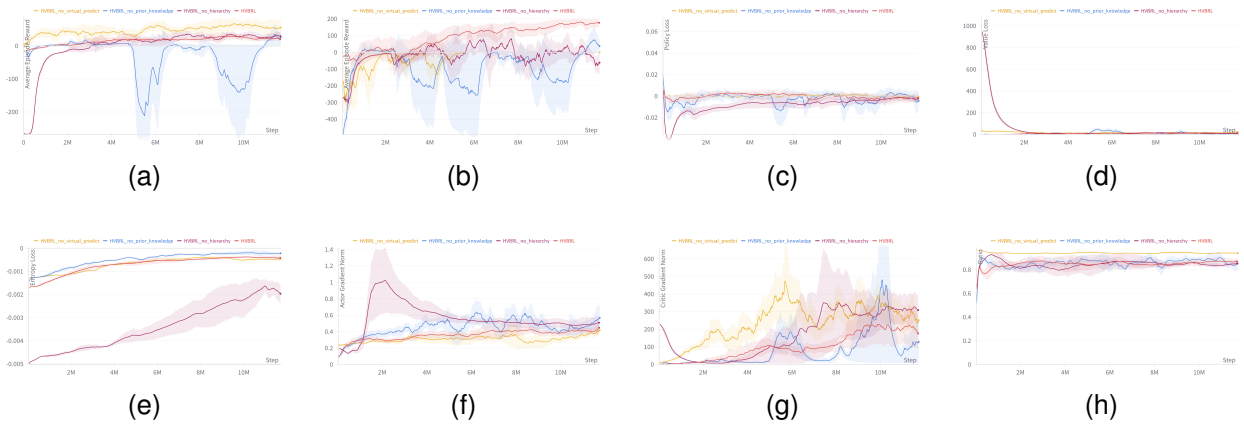

Fig. 5. Comparison training curves of the four algorithms, the red curve represents the HVBRL algorithm, the blue curve represents the HVBRL algorithm without simple prior knowledge, the yellow curve represents the HVBRL algorithm without virtual missile prediction, and the purple curve represents the non-hierarchical HVBRL algorithm. (a) training reward. (b) evaluation reward. (c) policy loss. (d) value loss. (e) entropy loss. (f) actor gradient norm. (g) critic gradient norm. (h) ratio.

aircraft). The high-level algorithm was trained over 12M steps in this fixed scenario, where Fig. 5a shows the curve of average rewards per episode during training; Fig. 5b shows the curve of average rewards per episode during evaluation; Figs. 5c, 5d, 5e display the curves of policy loss, value loss, and entropy loss during training; Figs. 5f and 5g exhibit the norm curves of the action network gradients and value network gradients during training; Fig. 5h presents the ratio curve during training. Due to a certain probability of executing missile launch actions during training, the average reward curve exhibits significant fluctuations. Therefore, observing only the curve in Fig. 5a does not indicate convergence of the training process. To assess whether the training process has converged, at the end of each episode, the latest training model is evaluated against a fixed strategy of the Blue aircraft, collecting reward information to plot in Fig. 5b. Moreover, to avoid low-probability sampling of missile launch actions leading to missile depletion, an hl"launch threshold" was introduced in the evaluation combat simulation process and set to 0.5; a missile launch action is taken when the predicted probability of missile hit exceeds this threshold. During training, to fully explore the action space, no launch threshold is set. As seen in Fig. 5b, the episode rewards gradually converge to around 200, close to the reward for hitting the target. Additionally, the loss curves shown in Figs. 5c, 5d, 5e and the gradient norm curves shown in Figs. 5f and 5g, reflect the convergence and stability of the training process. The ratio curve shown in Fig. 5h reflects the correct execution of training updates.

Additionally, three more situations are compared, the first one (yellow curve) is to only use simple prior knowledge without using the virtual prediction method; the second one (blue curve) is to only use virtual prediction without adding prior knowledge; the last one (purple curve) is to abandon hierarchy structure. Training evaluation reward in Fig. 5b shows that the loss of any one method will lead to worse performance in the training results. Since the comparative algorithm performed poorly in fixed scenarios, we did not further extend it to self-play scenario. What's more, evaluation rewards are much higher than training rewards at some steps, the reason being that we trained in parallel across four environments and The average round rewards have been averaged. Due to the presence of randomness (a small probability of sampling a missile launch action, causing the enemy aircraft to make evasive maneuvers in advance, resulting in subsequent inability to hit the enemy), training includes exploration factors. This leads to situations where, even if the model learns the appropriate timing to launch a missile, some environments still exhaust their ammunition before they should fire. During evaluation, we removed the random exploration factor (by setting a launch threshold). All curves use exponential smoothing with a smoothing coefficient of 0.88.

Figs. 6a, 6b, and 6c respectively show the predicted number of virtual missile hits by the Red aircraft when the Blue aircraft adopts head-on, fleeing, and tail-chase escape maneuvers in three strategy scenarios, with the maximum number of virtual missiles set to 200 during the experiment. As seen in Fig. 6a, there is a higher predicted hit probability near 200 steps when the Blue aircraft performs head-on maneuvers, ending the simulation round earlier, which reflects a preference for close-range attacks. From Fig. 6b, it can be seen that when the Blue aircraft adopts fleeing maneuvers, there is an upward trend in the predicted number of virtual missile hits around 600 steps, which aligns with common sense. As seen in Fig. 6c, when the Blue aircraft performs tail-chase escape maneuvers, there is a higher tendency to shoot around 150 steps. The early termination of the simulation in all three scenarios is due to effective prediction and shooting.

Figs. 7a and 7b show the top and side views of the flight trajectories of both sides during the experiment in Fig. 6a, from which it can be seen that the Red aircraft

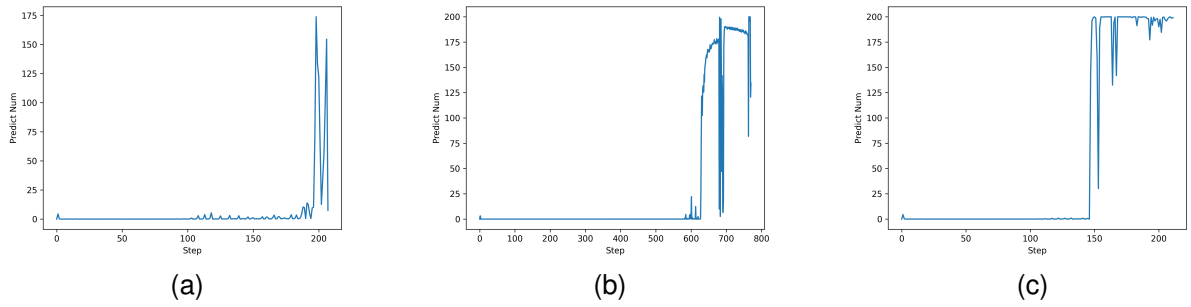

Fig. 6. Predicted number of virtual missile hits. (a) head-on. (b) fleeing. (c) tail-chase escape.

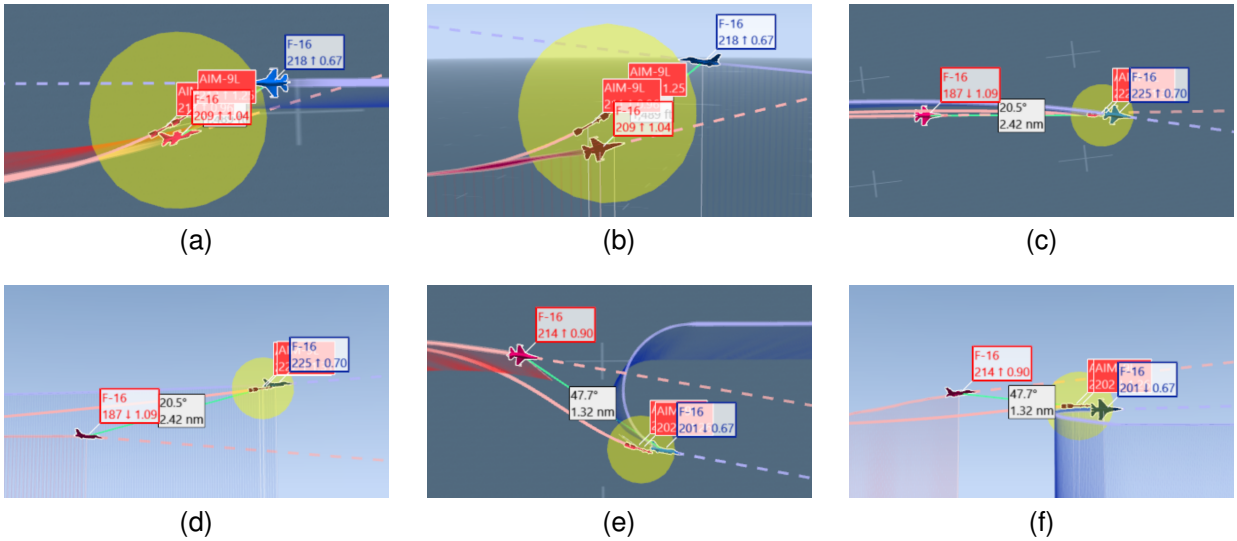

Fig. 7. Flying trajectory for high-level algorithm. (a) head-on (top view) (b) head-on (side view). (c) fleeing (top view). (d) fleeing (side view). (e) tail-chase escape (top view). (f) tail-chase escape (side view).

launches missiles and successfully hits the Blue aircraft at a close distance, with the Red aircraft striking from the flank, holding a tactical advantage. Figs. 7c and 7d display the flight trajectories of both sides during the experiment in Fig. 6b, where it can be seen that after guiding for a certain distance, the Red aircraft launches missiles and successfully hits the Blue aircraft. Throughout the process, the Red aircraft consistently tracks the Blue aircraft, accelerating its approach by lowering altitude to increase speed. Figs. 7e and 7f show the flight trajectories of both sides during the experiment in Fig. 6c, where it can be seen that the Red aircraft launches missiles at a close distance, and the Blue aircraft subsequently turns to escape but is ultimately hit due to being too close to evade the missile tracking. The dashed line represents the heading of the aircraft. Whether the missile hits the target can be determined by the missile explosion marker (yellow spherical area) in the figure and whether the blue aircraft comes into contact with this spherical area.

The Monte Carlo method is a statistical technique that allows for numerical solutions to complex problems through the use of random sampling. In a typical Monte Carlo simulation, a large number of random variations are input into a mathematical model of the problem, and the resulting outcomes are averaged to obtain an approximation. The accuracy of the solution improves as more random samples are included in the simulation. Since the total simulation steps in a round is 1000, carrying 1000 missiles can meet the maximum missile demand. Unlike using neural networks for virtual missile hit number prediction, the Monte Carlo method gives the hit situation by actually launching missiles in a round; it estimates the hit probability by simulating multiple rounds with adjusted opponent models and fixed self-motion trajectories. Opponent models need to be constrained to move within a spherical region, so that we can estimate the probability of our missiles hitting the target within the sphere at different distances and orientations from the center of the sphere. The smaller the radius of the sphere, the more accurate the estimated hit probability, the fewer the number of simulations required, but the weaker the generalization. The sphere radius is 1km in our experiment.

Finally, compared to the Monte Carlo method with a simulation of 1000 missiles (because this is the maximum number of steps in a round, which can be satisfied even if

TABLE VI
Evaluation indicators in three fix scenarios.

	HVBRL	HVBRL_ no_prior_knowledge	HVBRL_no_virtual_predict	MC
FPS avg	135	130	139	4
FPS max	140	138	148	5
FPS min	126	120	127	2
Missile num	2	2	2	1000
Success scenarios	3	1	0	1

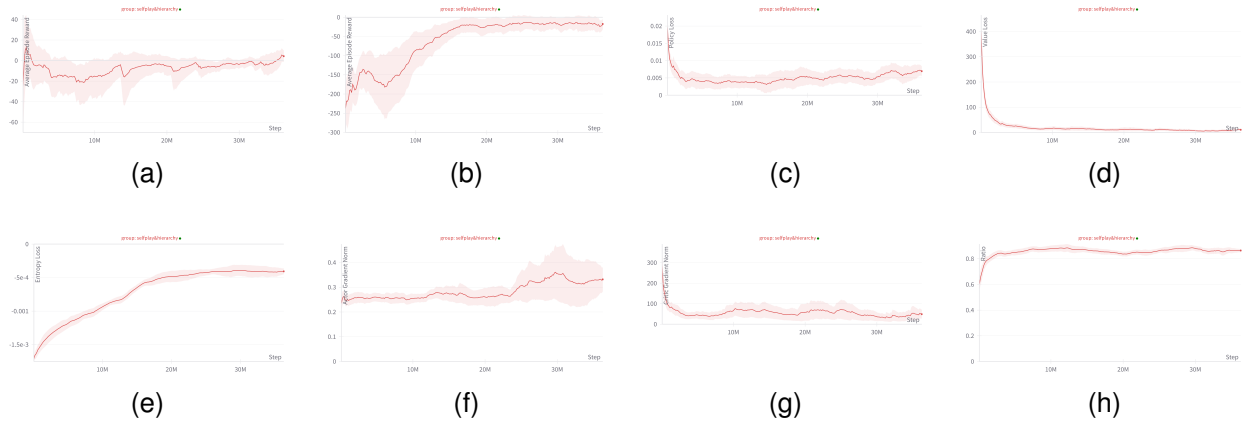

Fig. 8. Training curve for high-level self-play algorithm. (a) training reward. (b) evaluation reward. (c) policy loss. (d) value loss. (e) entropy loss. (f) actor gradient norm. (g) critic gradient norm. (h) ratio.

a missile is launched at every step), which has an average simulation frequency of only 4 steps per second due to the need to simulate multiple missiles in parallel, the self-play algorithm of the high-level model achieves an average simulation frequency of 135 steps per second, thus enhancing the simulation rate by nearly 32.75 times. A sufficient simulation rate also guarantees the learning update rate of the network. The HVBRL algorithm wins all three scenarios while the MC algorithm only wins one scenario which is head-on setting. More evaluation indicators are shown in Table VII.

D. Performance of the High-Level Self-Play Algorithm in Game Scenarios

The high-level self-play algorithm underwent a training process lasting 35 million steps in the game scenario. Figs. 8a-8h respectively illustrate the average reward per episode during the training process, the curve of average reward per episode during evaluation, policy loss, value loss, entropy loss, the gradient norm of the action network, the gradient norm of the value network, and the ratio curve. To stabilize the evaluation of the model's performance during training, this study uses the most recent training model as the red aircraft model and randomly selects a historical model as the blue aircraft model for a round of simulation combat, collecting rewards as evaluation rewards. Similarly, during the training and evaluation process, this algorithm introduces a "launch threshold" set to 0.5 because of the random selection of opponents and a low probability of shooting. The evaluation curve in Fig. 8b shows that the episode rewards gradually converge near zero, indicating that the game process is gradually converging to a Nash equilibrium. The smoothing method is consistent with the previous section.

Figs. 9a and 9b display the predicted missile hit counts by the red and blue sides through the high-level self-play algorithm, with the maximum number of virtual missiles set to 200 during the experiment. Fig. 9a shows the prediction during the game simulation process of the best response strategy (red aircraft strategy) versus the historical strategy (blue aircraft strategy), from which it can be seen that the blue aircraft launches missiles slightly earlier than the red aircraft, and the blue aircraft believes there is a very high missile hit probability within nearly 200 consecutive steps. However, the red aircraft's prediction shows a lot of fluctuation, reflecting a relatively balanced offensive and defensive strategy. Fig. 9b shows the prediction when both sides adopt the best response strategy, showing that the red and blue predictions are basically consistent, with the same missile launch timing, and the prediction results fluctuate.

Figs. 10a and 10b present the top view and side view of the flight trajectories of both sides during the experiment depicted in Fig. 9a, where it is visible that the red aircraft's missiles successfully hit the blue aircraft, while the red aircraft successfully evades the blue aircraft's missiles. From Fig. 10a, it is clear that both sides adopt a similar S-maneuver turning evasion maneuver in response to the enemy missiles, but the red aircraft has a smaller turning radius, thereby successfully avoiding the missiles. Figs. 10c and 10d show the top view and side view of the flight trajectories of both sides during the experiment

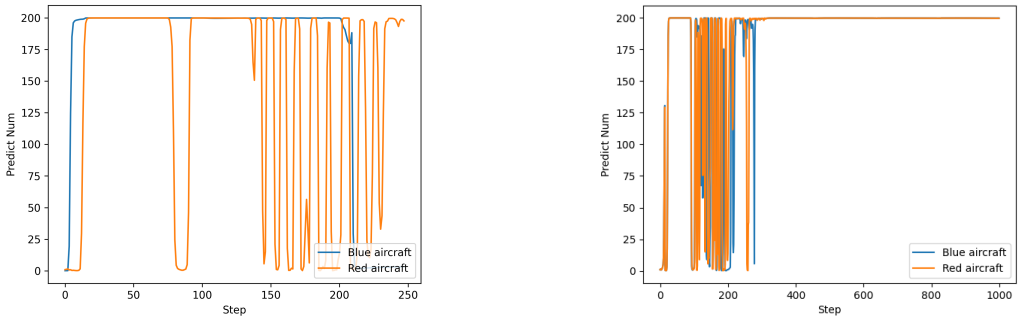

Fig. 9. Predicted number of virtual missile hits. (a) VS historical enemy. (b) VS self.

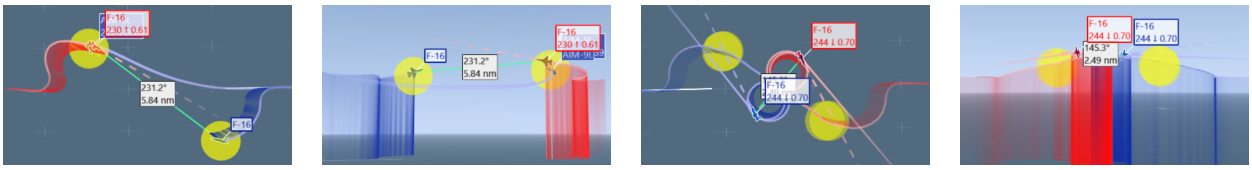

Fig. 10. Flying trajectory for high-level self-play algorithm. (a) historical enemy (top view) (b) historical enemy (side view). (c) self (top view). (d) self (side view).

TABLE VII
Evaluation indicators in self-play scenario.

	HVBRL (selfplay)	MC
FPS avg	96	4
FPS max	102	5
FPS min	82	2
Missile num	2	1000
Success rate	90.4%	4.2%

depicted in Fig. 9b, where it is visible that both sides reach an equilibrium state, avoiding each other's missiles and engaging in close-range circling after the missiles are exhausted, consistent with the common knowledge of real-life close-range aerial combat.

Finally, compared to the Monte Carlo method simulation carrying 1000 missiles, the average simulation frequency of the high-level self-play algorithm is 96 steps per second, which is a 23-fold increase in simulation rate. The success rate of HVBRL algorithm in self-play scenario is 90.4% while MC algorithm is 4.2% out of 1000 times episodic simulation. More evaluation indicators are shown in Table VII.

VI. CONCLUSION

In this paper, we address the multitude of challenges faced in close-range autonomous air combat scenarios. Based on a dual-layer network architecture, we propose an autonomous attack decision algorithm using HVBRL, encompassing both fixed strategy confrontation and self-play. Extensive experimental results demonstrate that the proposed method effectively accomplishes intelligent missile launching, situational dominance, and evasion of attacks. It resolves the issue of inefficiency caused by carrying a large number of missiles, providing significant implications for AI in prop-based games and combat simulation research. Future work will aim to expand the problem scale and explore autonomous decision-making in multi-agent systems.

REFERENCES

[1] Yang, Q., Zhang, J., Shi, G., Hu, J., Wu, Y., 2020. Maneuver decision of uav in short-range air combat based on deep reinforcement learning. IEEE Access 8, 363–378. doi:10.1109/ACCESS.2019.2961426.

[2] Zhang, J., Yu, Y., Zheng, L., 2023. Situational continuity-based air combat autonomous maneuvering decision-making. Defence Technology 29, 66–79.

[3] Yang, Q., Zhu, Y., Zhang, J., Qiao, S., Liu, J., 2019. Uav air combat autonomous maneuver decision based on ddpg algorithm, in: 2019 IEEE 15th International Conference on Control and Automation (ICCA), pp. 37–42. doi:10.1109/ICCA.2019.8899703.

[4] Zhang, H., Wei, Y., Zhou, H., Huang, C., 2022. Maneuver decision-making for autonomous air combat based on fre-ppo. Applied Sciences 12, 10230.

[5] Wang, Z., Li, H., Wu, H., Wu, Z., 2020. Improving maneuver strategy in air combat by alternate freeze games with a deep reinforcement learning algorithm. Mathematical Problems in Engineering 2020, 1–17.

[6] Li, B., Huang, J., Bai, S., Gan, Z., Liang, S., Evgeny, N., Yao, S., 2023b. Autonomous air combat decision-making of uav based on parallel self-play reinforcement learning. CAAI Transactions on Intelligence Technology 8, 64–81.

[7] Li, B., Bai, S., Liang, S., Ma, R., Neretin, E., Huang, J., 2023a. Manoeuvre decision-making of unmanned aerial vehicles in

air combat based on an expert actor-based soft actor critic algorithm. CAAI Transactions on Intelligence Technology.

[8] Zhang, Y., Yang, Z., Chai, S., He, Y., Wang, X., Zhou, D., 2023. Maneuver and attack strategy generation method for autonomous air combat in hybrid action space based on proximal policy optimization, in: 2023 42nd Chinese Control Conference (CCC), IEEE. pp. 3946–3953.

[9] Zhu, J., Kuang, M., Zhou, W., Shi, H., Zhu, J., Zhang, H., Li, X., Han, X., 2023. Mastering air combat game with deep reinforcement learning.

[10] Jiang, Y., Yu, J., Li, Q., 2022. A novel decision-making algorithm for beyond visual range air combat based on deep reinforcement learning, in: 2022 37th Youth Academic Annual Conference of Chinese Association of Automation (YAC), IEEE. pp. 516–521.

[11] Hu, T., Hu, J., Zhao, C., Pan, Q., 2022. Autonomous decision making of uav in short-range air combat based on dqn aided by expert knowledge, in: International Conference on Autonomous Unmanned Systems, Springer. pp. 1661–1670.

[12] Xiaoyu, Z., Jiangtao, H., Zhe, Z., Sheng, Z., Pan, Z., 2022. Intelligent air combat maneuvering decision based on td3 algorithm, in: International Conference on Autonomous Unmanned Systems, Springer. pp. 1082–1094.

[13] Kong, W., Zhou, D., Yang, Z., Zhao, Y., Zhang, K., 2020a. Uav autonomous aerial combat maneuver strategy generation with observation error based on state-adversarial deep deterministic policy gradient and inverse reinforcement learning. Electronics 9, 1121.

[14] Wu, Y., Lei, Y., Zhu, Z., Wang, Y., 2022b. Decision modeling and simulation of fighter air-to-ground combat based on reinforcement learning, in: Proceedings of the 4th International Conference on Image Processing and Machine Vision, pp. 102–109.

[15] Lee, G.T., Kim, C.O., 2020. Autonomous control of combat unmanned aerial vehicles to evade surface-to-air missiles using deep reinforcement learning. IEEE Access 8, 226724–226736.

[16] Pope, A.P. and Ide, J.S., 2022. Hierarchical reinforcement learning for air combat at darpa's alphadogfight trials. IEEE Transactions on Artificial Intelligence.

[17] Xu,J.andZhang,J.,2022.Autonomousdecision-makingfordogfights based on a tactical pursuit point approach. Aerospace Science and Technology, 129, 107857.

[18] Yoo, J. and Seong, H., 2022. Deep reinforcement learning-based intelligent agent for autonomous air combat, in: 2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC), IEEE. pp. 1–9.

[19] Ruan, W. and Duan, H., 2022. Autonomous maneuver decisions via transfer learning pigeon-inspired optimization for ucavs in dog- fight engage- ments. IEEE/CAA Journal of Automatica Sinica, 9, 1639–1657.

[20] Kallstrom, J. and Heintz, F., 2020. Agent coordination in air combat simulation using multi-agent deep reinforcement learning, in: 2020 ieee international conference on systems, man, and cybernetics (smc), IEEE. pp. 2157–2164.

[21] Wang, Y. and Long, Q., 2023. Research on maneuvering control algorithm of short-range uav air combat based on deep reinforcement learning, in: 2023 2nd International Conference on Machine Learn- ing, Cloud Computing and Intelligent Mining (MLCCIM), IEEE. pp. 83–90.

[22] Teng, T.H. and Tan, A.H., 2012. Self-organizing neural networks for learning air combat maneuvers, in: The 2012 international joint conference on neural networks (IJCNN), IEEE. pp. 1–8.

[23] Pope, A.P., Ide, J.S., Micovic, D., Diaz, H., Rosenbluth, D., Ritholtz, L., Twedt, J.C., Walker, T.T., Alcedo, K., Javorsek, D., 2021. Hierarchical reinforcement learning for air-to-air combat, in: 2021 International Conference on Unmanned Aircraft Systems (ICUAS), pp. 275–284. doi:10.1109/ICUAS51884.2021.9476700.

[24] Jiandong, Z., Dinghan, W., Qiming, Y., Guoqing, S., Yi, L., Yaozhong, Z., 2023. Multi-dimensional decision-making for uav air combat based on hierarchical reinforcement learning. Acta Armamentarii 44, 1547.

[25] Kong, W.r., Zhou, D.y., Du, Y.j., Zhou, Y., Zhao, Y.y., 2023. Hierarchical multi-agent reinforcement learning for multi-aircraft close-range air combat. IET Control Theory & Applications 17, 1840–1862.

[26] Zhou, W.J., Subagdja, B., Tan, A.H., Ong, D.W.S., 2021. Hierarchical control of multi-agent reinforcement learning team in real-time strategy (rts) games. Expert Systems with Applications 186, 115707.

[27] Yuan, Y., Yang, J., Yu, Z.L., Cheng, Y., Jiao, P., Hua, L., 2023. Hierarchical goal-guided learning for the evasive maneuver of fixed-wing uavs based on deep reinforcement learning. Journal of Intelligent & Robotic Systems 109, 43.

[28] Wang, B., Li, S., Gao, X., Xie, T., 2021. Uav swarm confrontation using hierarchical multiagent reinforcement learning. International Journal of Aerospace Engineering 2021, 1–12.

[29] Chai, J., Chen, W., Zhu, Y., Yao, Z.X., Zhao, D., 2023. A hierarchical deep reinforcement learning framework for 6-dof ucav air-to-air combat. IEEE Transactions on Systems, Man, and Cybernetics: Systems 53, 5417–5429. doi:10.1109/TSMC.2023.3270444.

[30] Qian, C., Zhang, X., Li, L., Zhao, M., Fang, Y., 2023. H3e: Learning air combat with a three-level hierarchical framework embedding expert knowledge. Expert Systems with Applications, 123084.

[31] Berndt, Jon. JSBSim: An Open Source Flight Dynamics Model. doi: 10.2514/6.2004-4923.

[32] Zhang, J.D., Wang, D.H., Yang, Q.M. Loyal wingman task execution for future aerial combat: A hierarchical prior-based reinforcement learning approach. doi: 10.1016/j.cja.2024.03.009

[33] Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms[J]. arXiv preprint arXiv:1707.06347, 2017.

Dinghan Wang was born in Heilongjiang Province, China in 2000. He received his bachelor's degree in Detection Guidance and Control Technology (Electronics) from Northwestern Polytechnical University. He is currently pursuing the master degree in System Engineering at Northwestern Polytechnical University, China. From 2018 to 2022, he was pursuing a bachelor's degree in Detection Guidance and Control Technology (Electronics). Since 2022, He entered Northwest Polytechnical University to study for a master's degree in System Engineering.

Jiandong Zhang obtained his bachelor's degree in fire control from Northwestern Polytechnical University in July 1997, a master's degree in system engineering from Northwestern Polytechnical University in March 2000, and a doctor's degree in system engi- neering from Northwestern Polytechnical University in 2005. From January 2005 to October 2007, he was engaged in research in the post doctoral mobile station of information and communication engineering of the university. In April 2007, he was promoted to associate professor of system engineering discipline of Northwest Polytechnical University. Since 2007, he has served as the tutor of master's degree. Member of China System Simulation Society.

Qiming Yang obtained his bachelor's degree in Detection Guidance and Control Technology from Northwestern Polytechnical University in July 2010, a master's degree in system engineering from Northwestern Polytechnical University in March 2013, and a doctor's degree in electronic science and technology from Northwestern Polytechnical University in 2020.

Jieling Liu was born in Shaanxi Province, China in 1973. She obtained her master degree from Northwestern Polytechnical University. Now she is working as a senior engineer in optoelectronic system overall technology.

Guoqing Shi was born in Shaanxi Province, China in 2001. He received his bachelor's degree in electronic information engineering from Xi'an Jiaotong University City College. He is currently pursuing the master degree in electronic science and technology at Northwestern Polytechnical University, China. From 2018 to 2022, he was pursuing a bachelor's degree in electronic information engineering. Since 2022, He entered Northwest Polytechnical University to study for a master's degree in electronic science and technology.

Yaozhong Zhang obtained his bachelor's degree in Electronic Engineering from Northwestern Polytechnical University in 1997, a master's degree in Information and Communication Engineering from Northwestern Polytechnical University in 2000, and a doctor's degree in Information and Communication Engineering from Northwestern Polytechnical University in 2006.