POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

ACE - Autonomous Combat Environment: a Modular Reinforcement Learning Framework for Dogfighting

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Author: **Emanuele Greco**

Student ID: 10741757
Advisor: Prof. Nicola Gatti
Co-advisors: Piergiuseppe Pezzoli
Academic Year: 2024-25

Abstract

Autonomous within-visual-range (WVR) air combat combines fast nonlinear flight dynamics, continuous control, uncertainty, and adversarial interaction, making systematic validation and fair comparison of Deep Reinforcement Learning (DRL) methods difficult. This thesis addresses these issues by introducing ACE (Autonomous Combat Environment), a unified and highly modular simulation and benchmarking framework designed to reduce fragmentation and improve reproducibility in autonomous air-combat research.

A two-stage methodology is adopted to separate algorithmic prototyping from physics-based validation. First, a suite of Gymnasium toy-model environments is developed to enable rapid iteration on observation design, action parameterizations, reward shaping, and episode structure under controlled conditions. This stage supports standardized evaluation protocols and direct comparisons of widely used continuous-control algorithms (PPO, SAC, TD3), reducing confounding factors that often make conclusions ambiguous. Second, the same pipeline is transferred to a JSBSim-based 6-DoF setting with realistic aircraft dynamics, actuator limits, and safety-relevant constraints. In this stage, training focuses on PPO, SAC, and recurrent PPO (rPPO) to better handle temporal dependencies typical of flight and pursuit tasks. Across both stages, the framework is designed to be algorithm-agnostic and hot-swappable, with clear separation between environments, rewards, algorithms, and evaluation scripts. Experiments are made maximally repeatable by using fixed random seeds whenever possible to isolate algorithmic effects from stochastic variance, and by adopting a structured logging stack with Weights & Biases for experiment tracking and TacView for high-fidelity trajectory inspection.

Results show that, in toy-model environments, robust and interpretable learning is consistently achievable when rewards and episode design are carefully engineered, enabling transparent benchmarking of pursuit and tracking behaviors. In contrast, high-fidelity experiments highlight that generalization and stability remain key bottlenecks as realism increases: randomized objectives, increased scenario variability, partial observability, and adversarial dogfighting lead to stronger sensitivity to initialization and reward details, as well as more frequent oscillatory or suboptimal behaviors.

Overall, this thesis contributes a reproducible baseline and a modular experimental infrastructure that supports rigorous method evaluation in autonomous aerial combat, while also delineating concrete directions for improvement, including stronger curricula, more principled constraint handling, improved opponent sampling, and memory-aware policies for partially observable settings.

Keywords: Reinforcement Learning, Dogfighting, Fixed-wing Aircraft, Modular Simulation Framework

Abstract in lingua italiana

Il combattimento aereo autonomo entro il raggio visivo (WVR) combina dinamiche di volo non lineari e rapide, controllo continuo, incertezza e interazione avversaria, rendendo difficile una validazione sistematica e un confronto equo dei metodi di Deep Reinforcement Learning (DRL). Questa tesi affronta tali criticità introducendo ACE (Autonomous Combat Environment), un framework di simulazione e benchmarking unificato e altamente modulare, progettato per ridurre la frammentazione e migliorare la riproducibilità nella ricerca sul combattimento aereo autonomo.

Viene adottata una metodologia in due fasi per separare il prototyping algoritmico dalla validazione basata sulla fisica. In primo luogo, viene sviluppata una suite di ambienti toy-model in Gymnasium per consentire uniterazione rapida su progettazione delle osservazioni, parametrizzazioni delle azioni, reward shaping e struttura degli episodi in condizioni controllate. Questa fase supporta protocolli di valutazione standardizzati e confronti diretti di algoritmi di controllo continuo ampiamente utilizzati (PPO, SAC, TD3), riducendo i fattori confondenti che spesso rendono ambigue le conclusioni. In secondo luogo, la stessa pipeline viene trasferita in un contesto 6-DoF basato su JSBSim, con dinamiche del velivolo realistiche, limiti degli attuatori e vincoli rilevanti per la sicurezza. In questa fase, laddestramento si concentra su PPO, SAC e PPO ricorrente (rPPO) per gestire meglio le dipendenze temporali tipiche dei compiti di volo e inseguimento. In entrambe le fasi, il framework è progettato per essere agnostico rispetto allalgoritmo e facilmente sostituibile (hot-swappable), con una chiara separazione tra ambienti, reward, algoritmi e script di valutazione. Gli esperimenti sono resi il più possibile ripetibili utilizzando, ove possibile, seed casuali fissi per isolare gli effetti algoritmici dalla variabilità stocastica, e adottando uno stack di logging strutturato con Weights & Biases per il tracciamento degli esperimenti e TacView per lispezione ad alta fedeltà delle traiettorie.

I risultati mostrano che, negli ambienti toy-model, un apprendimento robusto e interpretabile è ottenibile in modo consistente quando reward e struttura degli episodi sono progettate con cura, permettendo un benchmarking trasparente dei comportamenti di inseguimento e tracking. Al contrario, gli esperimenti ad alta fedeltà evidenziano che la generalizzazione e la stabilità rimangono colli di bottiglia principali allaumentare del

realismo: obiettivi randomizzati, maggiore variabilità degli scenari, osservabilità parziale e dogfighting avversario portano a una maggiore sensibilità allinizializzazione e ai dettagli della reward, oltre che a comportamenti oscillatori o subottimali più frequenti.

Nel complesso, questa tesi fornisce una baseline riproducibile e uninfrastruttura sperimentale modulare che supporta una valutazione rigorosa dei metodi nel combattimento aereo autonomo, delineando al contempo direzioni concrete di miglioramento, tra cui curriculum più efficaci, una gestione dei vincoli più fondata, un campionamento degli avversari migliore e policy dotate di memoria per contesti a osservabilità parziale.

Parole chiave: Apprendimento per rinforzo, Combattimento aereo ravvicinato, Velivolo ad ala fissa, Architettura modulare di simulazione

Contents

1 | Introduction

1.1. Context & Motivation

Autonomous decision-making for air combat has become a central research topic because modern engagements demand extremely fast, high-quality decisions under uncertainty. The pilot (or human operator) must continuously fuse information from sensors, communications, and tactical cues while coping with tight time constraints and rapidly changing threat configurations. In realistic scenarios, the situation is further complicated by multiple simultaneous adversaries and the need for coordination.

This trend is accelerating as a new era of aerial warfare emerges, centered on sixth-generation fighter concepts that integrate advanced artificial intelligence, pervasive sensor fusion, autonomous UAV wingmen, and digital communications to enable cooperative operations. Compared to todays most sophisticated platforms, this vision emphasizes AI-assisted mission management, real-time data connectivity, and dynamic human-machine teaming across multi-domain battlespaces. Operationally, it includes the deployment of networked UAV swarms, collaborative mission execution, and resilient autonomous agents. As adversarial, multi-agent scenarios become the norm both within and beyond visual range, research attention has increasingly converged on (Deep) Reinforcement Learning (RL) as a key enabling technology to address the complexity and dynamism of next-generation missions.

Within this broader context, Within Visual Range (WVR) dogfighting is an especially demanding benchmark for RL. WVR engagements are characterized by highly dynamic, nonlinear interactions where small timing differences and subtle geometry changes can flip the outcome. Agents must act in continuous time and space, often relying on incomplete or noisy observations, while respecting hard flight-envelope and safety constraints. At the same time, WVR requires long-horizon tactical trade-offs, such as sacrificing short-term position to gain advantage, making it a compelling stress test for RL methods intended for autonomous air combat.

1.2. Problem Definition: WVR Dogfight

Within-visual-range (WVR) air combat, commonly referred to as *dogfighting*, is one of the most demanding problems in aerospace autonomy. The objective is to endow unmanned aerial vehicles (UAVs) with the capability to engage in close-range aerial combat without direct human intervention, in highly dynamic and adversarial conditions. Unlike beyond-visual-range (BVR) engagements, WVR encounters unfold at short distances and high angular rates, where the outcome is strongly influenced by relative geometry, rapid maneuvering, and continuous control of the aircraft dynamics. In this setting, an autonomous agent must simultaneously manage flight stability and envelope limits while pursuing tactical advantage against an opponent that actively reacts, deceives, and counters.

The strategic relevance of autonomous dogfighting is multiple. First, it can reduce risks to human pilots by enabling unmanned platforms to operate in hazardous missions, directly contributing to force protection. Second, autonomy can improve operational accuracy and responsiveness by leveraging the speed, precision, and adaptiveness of algorithmic decision-making under uncertainty. As modern conflict environments become increasingly contested, the ability to deploy capable autonomous air combat agents can provide a decisive tactical advantage, especially as traditional manned platforms face evolving threats and constraints.

1.3. Technical Challenges

Autonomous WVR dogfighting is challenging because it is an interactive control problem against a learning or adaptive adversary. The underlying aircraft dynamics are nonlinear and high-dimensional, and effective behavior requires reasoning over rapidly changing relative geometry, while respecting physical actuator constraints and safety limits. In practice, an agent must couple low-level continuous control (e.g., smooth, stable maneuver execution) with high-level tactical intent (e.g., pursuit, repositioning, or disengagement), often under partial observability and with delayed consequences of actions.

From a learning perspective, several obstacles make progress difficult. Reward design is particularly sensitive: dense shaping is often necessary to guide exploration, yet poorly structured rewards can induce stable but tactically irrelevant behaviors (e.g., safe orbiting or indecisive chasing) rather than decisive engagement. Generalization is another core difficulty: policies trained on limited initial conditions or restricted opponent behaviors can fail when target locations, trajectories, or adversarial strategies vary. Training stability

can further degrade as realism increases, because high-fidelity dynamics amplify the impact of small control errors and expose policies to oscillations, brittleness, and constraint violations. These challenges motivate a methodology that separates what can be learned reliably in simplified settings from what must be validated in physics-based simulation.

1.4. The Role of Reinforcement Learning

Traditional rule-based control and static optimization approaches struggle to address the combination of high-dimensional continuous dynamics, nonlinear constraints, and adversarial interaction that characterizes WVR dogfighting. Hand-engineered logic can be effective in narrowly defined scenarios, but tends to be brittle when facing rapidly changing contexts and unpredictable opponents. By contrast, Deep Reinforcement Learning (Deep RL) provides a data-driven framework in which policies are learned through trial-and-error interaction with an environment, allowing agents to discover effective maneuver strategies directly from experience.

In principle, RL is well-suited to this domain because it can learn closed-loop behaviors that integrate perception of complex state spaces with continuous control, adapting to unstructured scenarios and opponent responses. Recent milestones have demonstrated that RL agents can achieve super-human performance across a variety of tasks, suggesting strong potential for aerospace autonomy. However, deploying such methods in safety-critical defense contexts requires a higher level of trust, which depends on systematic evaluation, reproducibility, and robust performance under variation. For this reason, RL must be studied not only for peak performance in narrow benchmarks, but also for stability, generalization, and interpretability across more realistic environments.

1.5. Objectives & Contributions

This thesis aims to develop and evaluate autonomous WVR dogfighting agents based on Deep Reinforcement Learning, with an emphasis on reproducible benchmarking and a clear separation between algorithmic prototyping and physics-based validation. The research follows a two-stage methodology. First, a suite of simplified Toy Model environments is implemented using the Gymnasium toolkit to rapidly prototype observation spaces, action parameterizations, and reward shaping strategies. These modular environments serve as controlled sandboxes for benchmarking baseline RL algorithms on meaningful air-combat sub-tasks such as target interception, trajectory tracking, and adversarial pursuit, enabling reliable comparisons through standardized metrics and consistent

training protocols. Second, the study transitions to high-fidelity simulation with the JSBSim flight dynamics engine, integrated through a Gym-compatible interface, exposing agents to realistic six-degrees-of-freedom dynamics, actuator limits, and safety-critical constraints.

The main contributions of this work address common limitations in prior literature, where results are often tied to a single algorithm or to customized, non-comparable environments that reduce cross-study insight and reproducibility. Specifically, this thesis:

- Develops a unified and modular simulation framework for autonomous aerial combat, spanning both toy-model and high-fidelity settings
- Allows higher reproducibility by making use of common interfaces, strong code baseline, hot-swappable algorithms and modular environments
- Systematically benchmarks multiple Deep RL algorithms across a curated set of aerial combat tasks using standardized evaluation metrics

Empirically, the toy-model stage demonstrates that, with properly shaped rewards and observation normalization, agents can learn pursuit and tracking behaviors with robust convergence and interpretable training dynamics, enabling debugging and ablation studies in a highly controlled setting. The subsequent JSBSim stage highlights a critical realism gap: while agents can still learn fundamental behaviors, tasks involving randomized trajectories and full adversarial dogfighting pose substantially greater challenges for stability and generalization, revealing strong sensitivity to reward design and a tendency toward oscillatory or suboptimal behaviors under variability. Together, these findings highlight the need to benchmark RL methods in both simplified and physically realistic settings, and they point to scalable approaches for future autonomous combat applications.

2 | Technical Background

This chapter explains the basic theory behind the methods used in this thesis. It gives a clear and short overview of Reinforcement Learning (RL), explains why Deep Neural Networks are useful for learning complex behaviors, and introduces the main Deep RL (DRL) algorithms (PPO, TD3, SAC, rPPO) used for autonomous air combat tasks.

2.1. Reinforcement Learning

Reinforcement Learning (RL) is a type of machine learning where an agent learns how to act in an environment by trying actions and getting feedback as rewards [64]. The main goal is to learn a policy that achieves the highest total reward over time. RL works well for problems that require a sequence of decisions, especially when it is hard to build an exact model of the system or to find a solution with traditional methods.

2.1.1. Markov Decision Processes

Formally, an RL problem can be described as a Markov Decision Process (MDP) [7], which provides a mathematical framework for decision-making in situations where outcomes depend on both the current state and the chosen action. An MDP is defined by a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where:

- $\mathcal{S}$: set of possible states describing the environment
- $\mathcal{A}$: set of possible actions
- $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition probability function, which describes the chance of moving from one state to the next when a specific action is taken
- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, which defines the immediate feedback the agent receives after moving from one state to another
- $\gamma \in [0, 1)$ is the discount factor, which controls how much the agent values future rewards compared to immediate rewards

Figure 2.1: Block diagram of a Markov Decision Process

A policy $\pi : \mathcal{S} \to \mathcal{A}$ (or, more generally, $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ for stochastic policies) tells the agent what action to take in each state, or gives a probability distribution over possible actions. At each timestep, the agent observes a state s, chooses an action a, receives a reward r and moves to a new state s'. The agent's goal is to choose actions over time to maximize the expected total return in the future.

2.1.2. Policies, Value Functions and Bellman Equations

For any policy, we can define value functions that estimate the expected return, meaning the sum of discounted rewards, starting from a given state (and possibly a given action). In particular:

- The state-value function is defined as:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s\right] \tag{2.1}$$

  where the expectation is taken over trajectories generated by following policy π, starting from the initial state $s_0 = s$

- The action-value function (Q-function) is defined as:

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a\right] \tag{2.2}$$

  which gives the expected return when starting in state s, taking action a, and then following policy π afterwards.

The main idea of dynamic programming in RL is expressed by the Bellman equations [6]. They break down the value functions in a recursive way:

$$V^*(s) = \max_{a \in \mathcal{A}} \mathbb{E}\left[r + \gamma V^*(s') \mid s, a\right] \tag{2.3}$$

$$Q^*(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q^*(s', a') \mid s, a\right] \tag{2.4}$$

where V^* and Q^* denote value functions under the optimal policy.

These equations are the starting point for most RL algorithms. They allow exact solutions in some simple, small cases, but in most real problems they are used to build learning updates based on sampled data and function approximation.

2.1.3. Why standard RL is not enough

Tabular RL methods, such as Q-learning [75] and SARSA [63], learn value functions and policies by storing a separate value for every state-action pair. This works well for small problems, but it becomes impractical in high-dimensional or continuous spaces because the number of state-action pairs grows very quickly (this phenomenon is known as "curse of dimensionality") [64]. Deep Reinforcement Learning (DRL) addresses this by using Neural Networks (NN) [49] to approximate value functions and policies. This makes it possible to handle large spaces and to generalize from situations the agent has already seen to new ones.

2.1.4. Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) uses Neural Networks (NN) as flexible function approximators. This allows policies and value functions to generalize to new situations and to work in high-dimensional settings. Moving from simpler, value-based RL methods to deep models made it possible to solve control problems with large state and actions spaces, strong non-linear dynamics, partial observability and complex reward signals.

A key idea in modern DRL is the actor-critic framework [5]. In this setup, the actor network chooses actions, while the critic network estimates how good those actions are (for example, by predicting a value or an advantage). This approach helps the agent learn which decisions led to good outcomes and often needs fewer samples, especially in continuous and high-dimensional tasks. Another important improvement is entropy regularization [77], which supports exploration by encouraging the policy to stay somewhat random, instead of becoming too deterministic too early. In addition, methods like expe-

rience replay [36] and target networks [41] are widely used to make training more stable and reduce sudden changes in learning.

2.1.5. Policy Gradient Methods

Policy gradient methods learn a parameterized policy π_θ by directly maximizing the expected return:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} [R(\tau)] \tag{2.5}$$

The policy gradient theorem states:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \, A^\pi(s_t, a_t) \right] \tag{2.6}$$

where:

- $\pi_\theta(a_t|s_t)$ is the probability of selecting action a_t in state s_t under policy π_θ
- $A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$ is the advantage function
- $\tau = (s_0, a_0, r_0, \ldots)$ is a trajectory generated by the policy

In practice, the policy network (actor) is updated using stochastic gradient ascent, with advantage estimates provided by the critic network. This combination is the basis of most modern actor-critic DRL algorithms.

Among modern actor-critic methods, four algorithms have shown strong performance across many challenging tasks:

- **Proximal Policy Optimization (PPO)** [53]: an on-policy method that uses a clipped objective to keep policy updates small, making training more stable and robust.
- **Twin Delayed Deep Deterministic Policy Gradient (TD3)** [20]: an off-policy method for continuous control. It uses two critic networks, delays the actor updates, and applies target policy smoothing to improve performance and reduce overestimation.
- **Soft Actor-Critic (SAC)** [67]: an off-policy method that aims to maximize both reward and entropy, encouraging better exploration. Is also uses two critic networks to make learning more stable and reliable.
- **Recurrent Proximal Policy Optimization (rPPO)** [43]: an extension of PPO

that augments the policy (and typically the value function) with recurrent memory, such as an LSTM or GRU. This allows the agent to integrate information over time and handle partial observability, where the current observation alone is insufficient to infer the underlying state.

Modern DRL research often uses simulation frameworks such as Gymnasium [19], which provide standard APIs for building environments and training agents. A major challenge, especially in air combat, is designing the state space, action space, and reward function carefully, because these choices strongly influence how the agent learns and how well the learned behavior transfers to real scenarios.

2.2. Neural Networks

Deep neural networks are at the core of modern reinforcement learning. They allow agents to approximate both policies (actors) and value functions (critics) effectively, even in continuous and high-dimensional settings. In this thesis, two main types of neural architectures were used:

- **MultiLayer Perceptrons (MLPs)**: In most tasks, the agent receives a low-dimensional vector that summarizes the environment's dynamic state. Because these state vectors are structured and fully observed, they are well suited to being encoded directly with fully connected MLPs [50]. The network processes the observation through several nonlinear hidden layers, producing a compact feature representation. The actor's output head then returns either continuous action values (TD3) or the parameters of an action distribution (SAC, PPO). The critic, which often has a similar architecture, outputs a scalar estimate such as a Q-value or a state-value. Overall, MLPs offer a flexible and computationally efficient base for policy learning in high-dimensional control, without requiring the data volume or architectural complexity typically associated with convolutional networks.
- **Recurrent Neural Networks (RNNs)**: In more realistic or partially observable settings, where state information may be missing, delayed or noisy, it is often beneficial to use recurrent architectures [17]. In this thesis, PPO agents were implemented with Long Short-Term Memory (LSTM) [25] layers placed after an initial MLP-based feature extractor. The LSTM maintains an internal memory and a hidden state that summarize recent information over time, allowing the agent to capture trajectory context and make better sequential decisions. This design helps the policy produce temporally consistent actions and remain robust

Convolutional Neural Networks (CNNs) [32] were not used because the state inputs were already concise and preprocessed, rather than raw high-dimensional sensory data. Likewise, attention mechanisms [4] and transformer-based models [69] were not adopted, since our environments did not involve complex spatial structure or variable-length sequences that would justify their added complexity. All architectures were designed to be extensible. The modular structure supports rapid prototyping, both with and without recurrent components, and can be adapted to more complex input settings if needed in future air-combat research.

2.3. Algorithms Used

Modern deep reinforcement learning algorithms stand out for their ability to handle complex, continuous environments efficiently and to perform reliably on demanding tasks such as autonomous air combat. In this section, we introduce four state-of-the-art actor-critic methods and discuss their main ideas, key implementation aspects, and why they are well suited for the control problems addressed in this thesis. It is worth noting that in reinforcement learning we distinguish methods between on and off policy [45]. On-Policy algorithms update their policy using data collected only from the current policy; this setup comes with strong theoretical foundations, but it limits how much collected experience can be reused. In contrast, Off-Policy methods can learn from data generated by older (or even different) policies by storing experience in a replay buffer. This typically leads to much higher sample efficiency and often supports more stable and parallel training. Overall, the choice between these two kinds of algorithms has a major impact on their design, sample complexity and practical deployment.

2.3.1. Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) [53] is a first-order, on-policy policy gradient method that is widely used because it is reliable, relatively simple to implement and robust across both discrete and continuous control tasks. It was introduced as a practical alternative to Trust Region Policy Optimization (TRPO) [52]. While TRPO relies on second-order optimization and explicitly constrains updates using the KL-divergence, PPO achieves similar behavior through simpler and more scalable regularization. The main contribution of PPO is its surrogate objective, which keeps each policy update within an implicit "trust region" so the new policy does not move too far from the previous one. In practice, this is enforced with a clipping mechanism that limits overly large policy changes. This design significantly improves training stability and reduces the risk of sudden performance

collapse, which is especially important in dynamic settings such as aerial dogfighting.

PPO alternates between collecting experience using the current policy and then updating both the policy and the value function on that data, which supports stable and efficient learning.

Clipped Surrogate Objective

The clipped surrogate objective introduced by PPO is defines as:

$$L^{CLIP}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) A_t \right) \right] \tag{2.7}$$

where:

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the probability ratio between new and old policies
- A_t is the (possibly normalized) advantage estimate
- ϵ is the clipping parameter (typically 0.2)

If the advantage is positive, the sampled action turned out better than what the current baseline expected, so the optimizer is encouraged to make that action more likely under the updated policy. However, PPO does not allow this probability to grow arbitrarily in a single update: once the probability ratio exceeds $1 + \epsilon$, the clipped term stops providing additional improvement, effectively capping the incentive to push the policy too far in one step. If the advantage is negative, the sampled action was worse than expected, so the optimizer tries to reduce its probability under the new policy. Again, PPO limits how much the policy can change at once: when the ratio drops below $1 - \epsilon$, the clipping mechanism prevents further gain from decreasing it, which avoids overly aggressive updates that could destabilize learning. Therefore, the clipping acts like a trust-region-style constraint: it preserves the direction of improvement suggested by the advantage, but restricts the magnitude of the policy update so training remains stable and robust.

Generalized Advantage Estimation (GAE)

Generalized Advantage Estimation computes the advantage by accumulating a discounted sequence of temporal-difference (TD) errors over multiple future steps. Instead of relying on a single-step TD error (which can be noisy) or a full Monte Carlo return (which can have high variance), GAE blends information across different horizons through the

parameter λ:

$$\hat{A}_t^{GAE(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \, \delta_{t+l}^V, \quad \text{where} \quad \delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{2.8}$$

Here λ controls how far into the future the advantage estimate looks. When λ is small, the estimate focuses more on short-term, bootstrapped predictions (lower variance but more bias). When λ is close to 1, it incorporates longer-term information and behaves more like Monte Carlo estimation (lower bias but higher variance). The result is a smoother, more reliable advantage signal that reduces update noise and improves training stability, which is particularly important in complex continuous-control settings where raw returns can fluctuate significantly from one trajectory to the next.

PPO Algorithm

Algorithm 2.1 Proximal Policy Optimization (PPO-Clip)

1: **Input:** initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment
4: Compute rewards-to-go $\hat{R}_t$
5: Compute advantage estimates $\hat{A}_t$ (using GAE) based on the current value function V_{ϕ_k}
6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_k}(s_t, a_t), \; g(\epsilon, A^{\pi_k}(s_t, a_t)) \right)$$

typically via stochastic gradient ascent with Adam
7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2$$

typically via gradient descent
8: **end for**

Advantages and Limitations

In practice, PPO benefits significantly from careful normalization. Normalizing observations, rewards, and especially the advantage estimates typically accelerates convergence and makes training more robust across different initial conditions and environment settings. Training stability also strongly depends on the update configuration: batch size and the number of epochs per update control how aggressively the policy is optimized on each data collection, and poor choices can lead to unstable learning dynamics or overfitting to the most recent rollouts. For a reliable assessment of performance, it is also important to train and evaluate across multiple random seeds and to use diverse environment resets, so results are reproducible and not driven by lucky initializations.

Despite its overall stability, PPO is not without drawbacks. Because its updates are intentionally conservative, the algorithm can sometimes plateau early and remain trapped in suboptimal behaviors, especially in complex control problems. At the same time, its flexibility and consistent empirical performance have made PPO one of the most widely used algorithms in reinforcement learning research and real world applications.

2.3.2. Twin Delayed Deep Deterministic Policy Gradient (TD3)

Twin Delayed Deep Deterministic Policy Gradient (TD3) [20] is an off-policy actor-critic algorithm introduced to improve the stability of deterministic policy gradient methods and to mitigate the well-known issue of Q-value overestimation. Its design builds on three main ideas. First, TD3 uses clipped double Q-learning: it learns two independent critic networks and computes the Bellman target using the smaller of the two estimates, which helps reduce optimistic value predictions. Second, it applies delayed policy updates, meaning that the actor and the target network are updated less frequently than the critics. This slows down rapid policy shifts and makes learning more stable. Finally, TD3 introduces target policy smoothing by adding small, clipped Gaussian noise to the target actions used in the critic update. This acts as a regularizer, making the value function less sensitive to narrow peaks and reducing the risk of the policy exploiting inaccurate sharp maxima in the learned Q-function.

TD3 Algorithm

Algorithm 2.2 Twin Delayed DDPG (TD3)

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$

Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$

repeat

Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, $\epsilon \sim \mathcal{N}(0, \sigma)$

Execute a in the environment

Observe next state s', reward r, and done signal d to indicate episode termination

Store (s, a, r, s', d) in replay buffer $\mathcal{D}$

if s' is terminal **then**

Reset environment state

end if

if it's time to update **then**

for j in range(however many updates) **do**

Randomly sample batch $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$

Compute target actions $a'(s') = \text{clip}(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High})$, $\epsilon \sim \mathcal{N}(0, \sigma)$

Compute targets $y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', a'(s'))$

Update Q-functions by gradient descent: $\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (Q_{\phi_i}(s, a) - y(r, s', d))^2$ for $i = 1, 2$

if j mod policy_delay $= 0$ **then**

Update policy by gradient ascent: $\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_{\phi_1}(s, \mu_\theta(s))$

Update target networks: $\phi_{\text{targ},i} \leftarrow \rho\phi_{\text{targ},i} + (1 - \rho)\phi_i$ (for $i = 1, 2$), $\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$

end if

end for

end if

until convergence

Advantages and Limitations

In practice, TD3 is often chosen because it directly targets two common failure modes in continuous control: Q-value overestimation and unstable learning dynamics. Target policy smoothing acts as a regularizer for the critic by making the Bellman targets less sensitive to sharp, potentially spurious peaks in the learned Q-function, which is particularly

helpful when actions are continuous. In addition, delaying the actor and target-network updates relative to the critic updates reduces the variance of value estimates and prevents the policy from changing too quickly based on still-immature critics, improving overall stability.

Exploration also requires attention: during data collection, it is common to inject extra zero-mean Gaussian noise into the actions executed in the environment, encouraging broader coverage of the action space and improving replay buffer diversity. Overall, TD3 delivers consistent and robust performance in high-dimensional continuous control problems.

2.3.3. Soft Actor-Critic (SAC)

Soft Actor-Critic (SAC) [67] is an off-policy actor-critic algorithm that combines high sample efficiency with an explicit mechanism to encourage exploration. Instead of optimizing only expected return, SAC adds an entropy term to the objective, which directly rewards stochastic policies. This encourages the agent to keep exploring and to avoid becoming overly confident too early. In air combat scenarios, where rewards can be sparse and the landscape is prone to local optima, this property is especially useful.

From an implementation perspective, SAC learns off-policy using an experience replay buffer. By reusing past transitions for many gradient updates, it typically achieves much better sample efficiency and often more stable training than on-policy methods. To reduce value overestimation, SAC also employs two critic networks and computes targets using the minimum of their Q-value estimates. Exploration is controlled through the temperature parameter α, which sets how strongly entropy is weighted in the objective: higher values promote broader exploration, while lower values shift the focus toward exploitation. In practice, SAC can tune α automatically during training, adjusting it to maintain a good balance between exploration and performance without requiring extensive manual tuning.

Entropy-Regularized Objective

In SAC, the agent is not trained to only chase high reward, but also to keep its action distribution spread out (high entropy). The entropy term rewards policies that remain stochastic, meaning they do not collapse too quickly to always taking the same action in a given state. This is crucial in continuous control and adversarial settings, where premature determinism can lock the agent into suboptimal maneuvers and reduce its ability to react to diverse situations.

$$J(\pi) = \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}\left[r(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))\right] \tag{2.9}$$

The parameter α sets how much the agent values exploration versus exploitation. A larger α makes the policy more exploratory (more randomness), while a smaller α allows the policy to become more deterministic and exploit what it has learned. With automatic tuning, α is adjusted during training to maintain a target entropy level: the policy stays sufficiently exploratory when uncertainty is high, and gradually becomes more focused as learning progresses. This mechanism often leads to more stable learning dynamics and better robustness to changing conditions compared to purely reward-driven objectives.

Soft Value Functions

In SAC, value estimation is built around soft Q-functions, meaning the critic does not evaluate actions purely by their expected discounted reward, but also accounts for the entropy-regularized nature of the policy. The soft Q-function still follows a Bellman-style recursion: it equals the immediate reward plus the discounted value of the next state.

$$Q^\pi(s_t,a_t) = r(s_t,a_t) + \gamma\mathbb{E}_{s_{t+1}}\left[V^\pi(s_{t+1})\right] \tag{2.10}$$

The key difference is that the value of a state is defined through the policy's action distribution, not through a hard max over actions.

$$V^\pi(s_t) = \mathbb{E}_{a_t\sim\pi}\left[Q^\pi(s_t,a_t) - \alpha\log\pi(a_t|s_t)\right] \tag{2.11}$$

As a result, SAC learns critics that are consistent with the objective of maximizing reward and entropy: actions are attractive not only when they yield high return, but also when they do not require the policy to become overly confident and collapse its distribution. Because off-policy methods can suffer from optimistic value estimates (overestimation bias), SAC adopts two separate critic networks and forms targets using the minimum of the two Q-values. Intuitively, if one critic becomes overly optimistic on some regions of the state-action space, the other critic is less likely to make the same error in the same way; taking the minimum provides a conservative target that reduces runaway feedback loops in value learning. This double-critic mechanism is one of the main reasons SAC tends to be stable and reliable in continuous-control tasks.

SAC Algorithm

Algorithm 2.3 Soft Actor-Critic (SAC)

Input: initial policy θ, Q-functions ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$
Set target parameters: $\phi_{targ,1} \leftarrow \phi_1$, $\phi_{targ,2} \leftarrow \phi_2$
repeat
  Observe state s and sample action $a \sim \pi_\theta(\cdot|s)$
  Execute a in the environment
  Observe next state s', reward r, and terminal signal d
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  if time to update **then**
    for each update step **do**
      Sample minibatch $\{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Sample next action $\tilde{a}' \sim \pi_\theta(\cdot|s')$
      Compute target: $y = r + \gamma(1-d)\left(\min_{i=1,2} Q_{\phi_{targ,i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right)$
      Update Q-functions: $\nabla_{\phi_i} \frac{1}{|B|} \sum (Q_{\phi_i}(s,a) - y)^2$ for $i = 1, 2$
      Update policy: $\nabla_\theta \frac{1}{|B|} \sum \left(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s)|s)\right)$
      Update target networks: $\phi_{targ,i} \leftarrow \rho\phi_{targ,i} + (1-\rho)\phi_i$ for $i = 1, 2$
    end for
  end if
until convergence

Advantages and Limitations

In practice, SAC requires careful tuning of a few key hyperparameters that strongly affect learning behavior. The entropy coefficient α plays a central role because it controls trade-off between exploration and exploitation: higher values encourage more stochasticity and broader exploration, while lower values lead to more deterministic, reward-driven behavior. For this reason, automatic temperature tuning is usually recommended, as it tends to make training more robust across different environments and reward scales. Training dynamics are also influenced by the replay buffer configuration. Both the buffer size and the minibatch size affect how efficiently experience is reused and how stable gradient updates are, especially in long training runs where the data distribution can drift. Finally, the target-network update settings matter: the soft-update coefficient ρ determines how quickly target networks track the learned networks, directly impacting the smoothness and stability of value estimates.

Overall, SAC is often preferred in complex, high-dimensional control problems where ex-

ploration is essential. Its entropy-regularized objective helps the agent avoid premature convergence, while off-policy learning with replay improves sample efficiency. These properties are particularly valuable in air combat tasks with sparse rewards and multiple local optima.

At the same time, SAC has notable drawbacks. Compared with simpler actor-critic methods, it is more sensitive to implementation details and hyperparameters, and training can become unstable if the entropy term, reward scale or target updates are poorly configured. SAC is also more computationally demanding than many alternatives, since it typically uses twin critics and performs multiple gradient updates per environment step. In addition, its performance can degrade when exploration noise is no longer beneficial, where the stochastic policy may slow down fine-tuning unless α is reduced appropriately.

2.3.4. Recurrent PPO (rPPO)

Recurrent Proximal Policy Optimization (rPPO) [43] extends the widely used on-policy algorithm PPO by adding a recurrent layer (typically an LSTM or GRU [12]) so that the policy can condition its decisions not only on the current observation, but also on an internal hidden state that summarizes past information. This makes rPPO particularly suited for partially observable settings (POMDPs) [2], where the instantaneous observation may be insufficient to infer the true state of the environment and memory becomes essential to act effectively.

From an implementation perspective, rPPO keeps PPO's clipped surrogate objective (with value loss and entropy bonus), but requires a careful redesign of the data pipeline and training procedure to support Truncated Backpropagation Through Time (TBPTT) [78]. Collected rollouts from multiple workers are split into episodes and, optionally, into fixed-length sequences. In addition, the hidden state of each sequence must be chosen consistently, and the model's forward pass must reshape data efficiently.

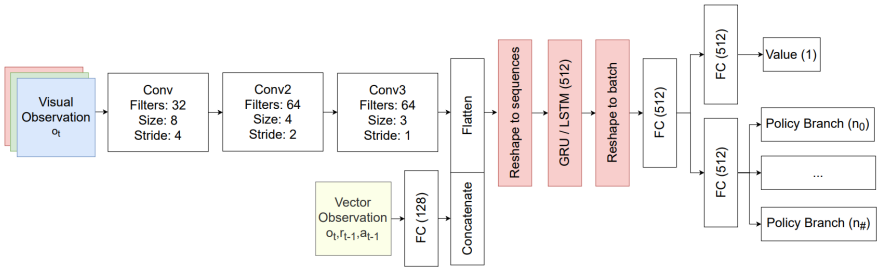


Figure 2.2: Visual representation of a Feed-Forward Convolutional Recurrent Neural Network used by rPPO [43]

Recurrent PPO Algorithm

Algorithm 2.4 Recurrent PPO (rPPO)

for iteration $= 1, 2, \ldots$ **do**
  for worker $= 1, 2, \ldots, W$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
  end for
  Compute advantage estimates $\hat{A}_{1,1}, \ldots, \hat{A}_{T,W}$
  Split trajectories into episodes
  if fixed sequence length **then**
    Split episodes into sequences of fixed length
  else
    sequences $\leftarrow$ *episodes*
  end if
  Zero-pad sequences that are too short
  Select initial hidden states of each sequence
  for epoch $= 1, 2, \ldots$ **do**
    for minibatch $= 1, 2, \ldots, MB$ with size $M \leq WT$ **do**
      Optimize $L^{\text{mask}}(\theta)$
    end for
    if epoch > 0 **then**
      Recompute hidden states and advantages w.r.t. θ
    end if
    $\theta_{\text{old}} \leftarrow \theta$
  end for
end for

Advantages and Limitations

Recurrent PPO (rPPO) inherits PPO's practical appeal (simple on-policy training with a clipped surrogate objective) while adding an explicit memory mechanism through a recurrent layer (e.g., LSTM/GRU). This allows the policy to condition actions on both the current observation and a hidden state that summarizes relevant past information, which is exactly what is needed in an air-combat scenario where critical cues may be missing from the instantaneous observation and capturing long-term dependencies (i.e., how certain actions modify the flight path). This means that the agent is capable of remembering which commands have already been executed, therefore has the ability to learn memory-

dependent behavior. Moreover, rPPO's generalization can improve substantially as the training diversity increases, meaning that the recurrent policy can learn a more robust solution rather than overfitting to a narrow set of solutions.

At the same time, rPPO's benefits come with non-trivial implementation complexity, as even small tuning errors may lead to misleadingly decent results on small problems, which is particularly dangerous when validating an implementation. From a computational standpoint, rPPO is typically slower and less scalable. Rercurrent networks introduce temporal dependencies and TBPTT require processing data in sequential chunks, thus reducing parallelism. As a result, we pay more time per update, and tuning the trade-off between sequence length and batch size becomes a constraint. Optimization is also harder, as even with LSTMs training can be more unstable than in the feed-forward case due to vanishing/exploding gradients, and because truncation limits how far gradients can propagate. If the task requires very long-term dependencies, a truncated horizon can prevent recurrent state from learning the right summaries, leading to policies that either ignore memory or develop brittle heuristics. In practice, this makes rPPO more sensitive to hyperparameters than standard PPO. Finally, recurrence introduces new debugging challenges: a recurrent policy can overfit by using its hidden state to "memorize" specific episode patterns rather than learning robust strategies. When the test distribution shifts, the hidden state can cause systematic errors that are hard to diagnose.

3 | Literature Review

This chapter surveys Deep Reinforcement Learning (DRL) research in autonomous air combat, tracing the evolution from early work on autonomous fixed-wing flight control to more recent, high-fidelity and adversarial dogfighting settings. It outlines how the problem has been formulated over time (control, maneuver decision-making, and multi-agent tactics), briefly summarizing typical state/action representations and training setups. The chapter also highlights key limitations, such as reward sparsity and realism gaps in simulators, that still constrain performance.

3.1. Evolution of Autonomous Air Control & Combat

The use of DRL in air-combat research has advanced markedly over the past decade, as studies have moved from relatively constrained settings to increasingly complex challenges involving sequential decision-making, tactical situational awareness, and adversarial interactions.

3.1.1. Foundations: control and early learning for Air Combat

McGrew et al. [39] address autonomous air-combat maneuvering by formulating a one-on-one engagement problem (level flight and fixed velocity) and solving for a real-time strategy using Approximate Dynamic Programming. The work is widely cited as an early demonstration that, with suitable function approximation and carefully designed features, ADP can yield policies that are competitive with human decision making in real time.

Ure et al. [68] propose a multimodal control and flight-planning framework designed to enable autonomous aggressive maneuvering across a UCAV's full flight envelope. Their key idea is maneuver decomposition: splitting complex aggressive behaviors into structured modes that simplify both planning and control, while improving robustness and feasibility under nonlinear aerodynamics and envelope limits.

3.1.2. DRL for fixed-wing flight control

Clarke and Hwang [13] explore DRL for direct aerobatic maneuvering control of agile fixed-wing aircraft, using a Normalized Advantage Function (NAF) formulation to handle continuous control. The key conceptual step is shifting from traditional controller structures (e.g., PID or model-based designs) toward a policy that maps state directly to control surface commands in a way that can capture highly nonlinear behavior during aggressive maneuvers. This connects to the broader question of where RL should sit in the autonomy stack: low-level control (surface/actuator commands), mid-level guidance (attitude/energy targets), or high-level tactics (maneuver selection). In practice, the work motivates careful handling of safety constraints (envelope protection, saturation, termination logic) and highlights the need for robust simulation settings to avoid brittle policies when transferring across conditions.

Zhang et al. [84] tackle DRL-based control for full 6-DOF fixed-wing flight, reflecting the move from simplified 3-DOF engagement kinematics to more realistic nonlinear aircraft motion. De Marco and D'Onza [14] propose a DRL control approach for high-performance aircraft and report results in a nonlinear dynamics setting, further supporting the feasibility of learning-based control in complex regimes. Collectively, these works motivate architectural decompositions where robust low-level control is either learned directly or combined with classical inner loops to stabilize training and execution. Richter et al. [48] provide a broad review of reinforcement learning for fixed-wing aircraft control tasks, organizing the landscape of RL applications beyond a single maneuver or scenario.

3.1.3. DRL for Maneuver Decision-Making in Autonomous Air Combat

A common approach to Within Visual Range (WVR) air-combat decision making is to define a maneuver/action set (continuous or discretized) and learn policies that maximize tactical advantage under crafted situation metrics.

Fan et al. [18] propose an A3C-based decision method for air-combat maneuver selection, highlighting asynchronous learning as a practical mechanism to accelerate training in complex tasks.

Li et al. [35] introduce a Multi-Step Double DQN (MS-DDQN) method for UCAV short-range combat decision-making, building a 6-DOF UCAV simulation with situation assessment functions and a set of basic actions; the multi-step return is used to improve training efficiency and convergence, and they report improved performance over DQN baselines.

Zhang et al. [82] propose FRE-PPO for maneuver decision-making, combining reward estimation with Proximal Policy Optimization (PPO) and an explicit training framework for air-combat agents. These contributions show that PPO-style architectures offer stability in continuous/high-dimensional problems.

Jing et al. [28] propose NC-DDPG, a modification of Deep Deterministic Policy Gradient (DDPG) where a node-clustering mechanism is used to reduce redundancy in the replay buffer and retain only representative transitions. The core idea is to improve learning efficiency for complex maneuvering strategies by filtering the stored experience while preserving the most informative samples. The approach is validated on a typical short-range air-combat maneuver decision task, showing that the proposed replay-management strategy can accelerate the learning process by reducing the computational burden without discarding critical experience.

3.1.4. Hierarchical RL and Self-Play for tactics

Hierarchical and adversarial training paradigms have become increasingly prominent. Pope et al. [44] describe a hierarchical architecture combined with maximum-entropy RL and reward shaping, developed in the context of DARPAs AlphaDogfight Trials; their approach stresses modular policy selection conditioned on engagement context and reports strong competitive performance.

Sun et al. [62] propose a Multi-Agent Hierarchical Policy Gradient (MAHPG) algorithm trained via Adversarial Self-Play to reduce reliance on expert knowledge and to handle hybrid (discrete/continuous) action spaces with a hierarchical decision network; they report that self-play can lead to emergent strategies that surpass expert-designed tactics.

Chai et al. [10] motivate hierarchical DRL specifically for 6-DOF UCAV air-to-air combat, arguing that direct end-to-end learning on full 6-DOF dynamics is substantially harder than on simplified 3-DOF models and thus benefits from hierarchical decompositions.

Selmonaj et al. [54] propose hierarchical multi-agent reinforcement learning for air-combat maneuvering, explicitly combining hierarchy with multi-agent learning to support both tactical decomposition and coordinated action selection. Their approach exemplifies the direction toward structured MARL solutions in which higher-level coordination can be learned jointly with lower-level maneuver generation.

Kong et al. [29] develop a hierarchical multi-agent reinforcement learning method for multi-aircraft close-range air combat, similarly targeting scalability to multiple aircraft and emphasizing hierarchical organization to make learning feasible in larger engagements.

Sun et al. [61] argue that many air-combat RL pipelines over-focus on optimizing the ego policy while under-modeling the opponent, despite the intrinsically game-theoretic nature of dogfights. They propose a two-stage approach that explicitly refines the adversary strategy from existing air-combat data using a Limited Imitation Offline RL (LIORL) method, then trains an ego agent in a simulator constructed around this refined (and potentially stronger) enemy behavior. LIORL is described as blending elements of imitation learning, maximum-entropy RL, and offline RL to extract a more reliable adversary policy from static datasets. In simulation, they report that the learned agent improves its win rate and can anticipate the refined opponents trajectory, with training curves converging to a high win rate. A key takeaway is that opponent-model fidelity can materially change what the ego policy learns; however, this direction introduces dependencies on dataset coverage/quality, and it raises the risk that errors or biases in the refined opponent model propagate into the ego training environment.

3.1.5. Multi-Agent Learning for Coordination

Kallstrom et al. [30] study multi-agent DRL as a way to construct coordinated synthetic pilots for air combat simulation, motivated by the difficulty of hand-crafting high-quality, human-like opponent behaviors. They empirically evaluate multiple approaches in two air-combat-related scenarios and report that curriculum learning helps address the high-dimensional state space and sparse rewards, while multi-objective learning can produce diverse agent characteristics useful for training applications.

Wang et al. [72] propose an evolutionary multi-agent reinforcement learning algorithm for multi-UAV air combat, incorporating evolutionary ideas to address exploration and population-level improvement in complex adversarial settings. This provides an alternative to pure gradient-based MARL, potentially improving robustness and diversity of tactics in scenarios where gradient methods struggle with local optima or brittle policies.

Yang et al. [79] study multi-UAV confrontation as a partially observable Markov game and propose DP-MADDPG, a variant of MADDPG that targets two common MARL failure modes in adversarial settings: slow/unstable convergence and collapse to a single dominant (possibly brittle) policy. The approach integrates a decomposed critic design with local and global dual critics to better balance individual and team-level signals, and Prioritized Experience Replay (PER) to improve sampling efficiency and accelerate learning. Experiments on the Multi-agent Combat Arena (MaCA) benchmark compare against standard MADDPG and independent-learning DDPG baselines, showing faster convergence and substantially higher reported win rates in the tested scenarios. The

paper highlights that practical multi-agent air-combat decision-making depends on both effective coordination and real-time action selection, but the broader challenge remains scaling these methods to richer sensing/communication constraints and ensuring robustness against non-stationary opponents beyond the training distribution.

3.1.6. Towards realism: high-fidelity environments, hybrid learning and richer objectives

A recurring criticism of early air-combat RL work is limited realism (simplified dynamics, low-fidelity sensors, narrow maneuver sets). Recent work increasingly targets more realistic environments and hybridizes learning signals. Bae et al. [3] study DRL-based air-to-air combat maneuver generation in a realistic environment, focusing on generating maneuvers rather than selecting from a small discrete library. This direction supports continuous maneuver synthesis and can reduce dependence on handcrafted maneuver catalogs, which is particularly relevant when the final goal is deployment in higher-fidelity simulators.

Zhao and Liu [86] introduce a physics-informed DRL approach for aircraft conflict resolution, illustrating a complementary direction where domain constraints and physical structure are incorporated to guide learning in safety-critical airspace problems. In addition, Yoo et al. [81] present a DRL-based intelligent agent for autonomous air combat evaluated in a realistic flight simulation environment (DCS), supporting the broader push toward higher-fidelity evaluation beyond toy simulators.

Li et al. [34] propose an Imitative Reinforcement Learning framework for autonomous dogfight, explicitly combining imitation signals with reinforcement learning objectives. This hybrid design is useful when expert demonstrations exist and when pure self-play RL would otherwise require prohibitive training time or would converge to unrealistic behaviors without strong priors.

Li et al. [33] study within-visual-range (WVR) air combat under complex nonlinear 6-DOF aircraft and missile dynamics using a hierarchical two-layer framework. To improve early-stage learning stability, they propose a cross-coordination mechanism between behavior cloning (BC) and PPO, alternating updates around the latest strategy and using additional regularization. The stated goal is to combine the fast warm-start properties of imitation with the asymptotic improvement of RL, yielding better sample efficiency and stronger game performance than baselines in their 6-DOF setting.

Mei et al. [40] propose a layered decision framework for 6-DOF WVR air-combat maneuver

generation, where the upper-level combat policy issues maneuvering instructions and the lower-level controller converts them into direct actuator commands. They model the control layer as an MDP and the combat layer as a POMDP, and emphasize training structure: expert-shaped rewards for stable low-level control, and a self-play curriculum at the combat level that plays against historical policies to progressively improve robustness. They report an operational success rate of 85.7% against a game-theory baseline and a reduction in training time compared to an RL baseline, supporting the practical value of combining hierarchy with curriculum/self-play in high-dimensional 6-DOF scenarios.

Chen et al. [11] propose the Missile Hit Probability Enhanced Actor-Critic (MHPAC) method for autonomous decision-making in air-to-air confrontation. The key mechanism is integrating a missile hit probability (MHP) neural predictor into both reward shaping and exploration, aiming to address sparse/delayed rewards typical of missile launch and defense. Their action set explicitly includes maneuvering commands plus a missile launch decision, and the reward is event-based. The paper reports that MHPAC improves strategy quality through curriculum learning and self-play, achieving a relative win ratio above 65% against different strategies, while maintaining real-time inference latency.

Yang et al. [80] propose an autonomous evasive maneuver method for a UCAV in air combat with multiple targets, shifting the focus from purely offensive maneuver optimization to survivability-oriented behavior under multi-threat conditions. This is relevant for modern engagement models where the agent must trade off pursuit, evasion, and threat management under limited time and information.

Seong and Shim [55] propose TempFuser, a policy architecture designed to capture both long-horizon tactical evolution and short-horizon aerodynamic transitions in one-versus-one dogfight settings. Their key idea is a long-short-term temporal fusion transformer: two distinct temporal transition embeddings are extracted (via separate LSTM units) and fused through a transformer encoder to model global context and opponent-dependent dynamics, enabling end-to-end flight command generation rather than selecting from a discrete maneuver library. The work emphasizes training and validation in a high-fidelity simulator (DCS) and introduces an energy-embedded reward intended to encourage acrobatic and tactically effective behaviors without relying on prior maneuver heuristics; the authors report improved performance versus baseline policies across multiple opponent aircraft types and challenging regimes. Main limitations in this line of work include the computational cost and sim-to-real gap of high-fidelity training, and the sensitivity of emergent tactics to reward shaping and opponent diversity during training.

3.1.7. Reward shaping, interpretability, and robustness

Wang et al. [74] address two recurring issues in DRL air-combat policies: limited interpretability and weak transferability. They propose an intent-driven framework that first categorizes pilot tactical intents into attack, defense, and evasion, then designs a mapping from intent to behavior and constructs corresponding reward models. A central theoretical motivation is that mixing sparse terminal rewards with dense shaping rewards can distort the reward-to-policy mapping and induce suboptimal convergence; the paper uses this to justify a more structured reward design. To improve stability and exploration in large state-action spaces, the authors devise a dueling-noisy-multi-step DQN variant and compare it against multiple DQN baselines/variants, reporting improved stability and convergence speed, as well as qualitatively interpretable intent-consistent trajectories.

Zhang et al. [85] focus on one of the central bottlenecks in WVR DRL training: reward design and the associated weight tuning in long-horizon, large-state environments. They propose a reward-design paradigm combining three components: sparse rule-based rewards, sub-goal rewards structured around transitions among four basic combat situations (pursuit, escape, back-to-back, face-to-face), and shaping rewards derived from a dynamic situation assessor (DSA). The DSA is built using an improved entropy weight method to increase objectivity and contrast in situation evaluation, while an improved sparrow search algorithm (SSA) is introduced as a parameter optimizer to automate tuning of PPO-related parameters and reward weights. Validation on a high-fidelity simulation platform (JSBSim) indicates improved training efficiency and competitive effectiveness versus state-of-the-art baselines in their experimental setup.

Zhang et al. [83] address a complementary challenge: adaptability to engagement conditions not seen during training. They propose an improved deep meta-reinforcement learning framework, where meta-training tasks are constructed from typical engagement situations to learn reusable maneuvering skills, and reward shaping is designed using potential-function ideas to stabilize learning across tasks. In meta-evaluation, the agent is reported to adapt quickly and reach about an 80% success rate on unseen tasks, suggesting that meta-RL can help reduce brittleness and improve generalization when the operational space cannot be exhaustively covered in training.

Jiang et al. [27] address common bottlenecks in UAV control by combining Meta-Learning with Generative Adversarial Imitation Learning (GAIL). The aim is to extract task-common structure during meta-training and to learn behavior from expert demonstrations without requiring carefully shaped reward signals, while also applying state normalization for stability. Experiments are conducted in simulated 2D settings, where the combined

"GAIL-enhanced Reptile" approach is reported to improve training efficiency over single-method baselines. However, the paper highlights a critical dependency: performance degrades substantially when expert trajectories are few or low-quality, underscoring that demonstration availability and fidelity can be the dominant practical constraint.

Wang et al. [73] focus explicitly on robustness of autonomous maneuver decision (AMD) policies when disturbances cause performance degradation after transferring from a training (source) environment to a disturbed (target) environment. Their core idea is to model the reachable set of trajectories under bounded disturbances as a tube and use its (normalized) size as an interpretable robustness indicator: smaller tubes correspond to more robust, but typically more conservative, actions. They incorporate this tube-size term into the reward with a tunable laziness factor that trades aggressiveness for robustness, and propose offline regression plus a tube library to avoid expensive online reachable-set computation during training. In simulation, they report improved robustness relative to selected robust RL baselines, using transfer-focused metrics such as changes in draw rate and shifts in capture-time distributions. A practical limitation noted by the authors is that the approach assumes access to information about the opponents future states, which restricts direct applicability in more realistic settings.

Han et al. [22] explicitly target interpretability by separating the control stack into three layers: a low-level four-channel control law, a mid-level library of eight basic fighter maneuvers (BFMs), and a high-level decision module that selects BFMs using Double DQN (DDQN). The opponent used during training is constructed with a decision tree (DT), and the authors report an 85.75% win rate against this DT baseline, with positive outcomes against several unseen opponents. The paper emphasizes post-hoc behavioral interpretation, highlighting emergent tactics such as yo-yo maneuvers to adjust turn rate and a "Dive and Chase" pattern that exploits opponent behavior.

While not an RL maneuver policy, Dong et al. [15] present a complementary direction: using historical engagement data to provide real-time decision support for basic fighter maneuvers (BFMs) in within-visual-range gun engagements. Using the Air Combat Engagement Database (ACED), they train a Transformer classifier to map time-series state features to BFM choices, reporting higher test accuracy than an LSTM baseline and extremely low inference latency. They add SHAP-based interpretability to explain feature contributions and introduce a situational assessment framework with multiple performance metrics. In 1vs1 experiments, they report large gains in win rate and increases in tactical/situational advantage-time measures, while noting that pilot data are sourced from commercial simulation software, which may limit direct applicability to real combat settings.

3.2. Gaps & Challenges

Despite rapid progress, the literature surveyed in this chapter reveals a set of persistent gaps that limit the reliability, comparability and operational relevance of DRL-based autonomous air combat systems. These challenges are not isolated: they interact, making end-to-end solutions difficult to validate and to transfer

3.2.1. Realism and the "simulator-to-operational" gap

A recurring limitation across the evolution from early 3-DOF settings to higher fidelity simulators is the remaining mismatch between training environments and operational conditions.

- **Dynamics and actuator realism**: Even when moving to 6-DOF non-linear aircraft models, many pipelines still simplify aerodynamics, actuator limits or failure modes, risking policies that exploit simulator artifacts rather than robust flight strategies.
- **High-fidelity simulators are expensive**: Training directly in platforms such as DCS [16], MFS [1] or X-Plane [31] can improve evaluation realism, but dramatically increases computational cost and iteration time, constraining ablation studies, hyperparameter search and opponent diversification.
- **Sensor, noise and latency modeling**: Several works implicitly assume near-perfect state information, while real air combat is dominated by partial observability, measurement noise and intermittent detection. Policies trained with privileged state features may not survive realistic sensing constraints.

3.2.2. Reward design

Reward design remains a central bottleneck, particularly for long-horizon WVR engagements.

- **Sparse and delayed signals**: Projectile hits, missile launches and terminal outcomes yield event-based sparse rewards, motivating auxiliary predictors but also leading to biased learning toward proxy objectives.
- **Shaping brittleness and weight tuning**: Many approaches rely on weighted mixtures of dense shaping and sparse rule-based rewards. Tuning these weights is often dependent on the faced scenario.

- **Unrealistic tactics**: In adversarial settings, policies may discover "reward hacks" or implausible behaviors unless constraints are carefully enforced.
- **Unclear objective**: Win-rate may hide undesirable behaviors, like unsafe maneuvers or near-collision regimes, unless multi-objective criteria are explicitly modeled.

3.2.3. Opponent modeling

Dogfighting is intrinsically game-theoretic, yet many pipelines still treat the opponent as a static or weak baseline.

- **Non-stationarity and instability**: As opponents evolve (self-play), the learning target changes, producing instability, catastrophic forgetting or cycling behaviors unless stabilization mechanisms (policy pools) are used.
- **Overfitting**: Self-play improves robustness, but policies can still over-specialize to the training population, leading to brittle performance against novel tactics.

3.2.4. Partial observability and memory

Real WVR decision making is a POMDP, so the agent must infer intent, hidden states and future threats.

- **State representations often remain "too Markov"**: Many studies use engineered global features or direct access to opponent kinematics that may not be observable in realistic settings.
- **Long-horizon temporal structure**: Capturing both short-term aerodynamic transients and long-term tactical evolution remains difficult. LSTMs or Transformers [69] are promising, but their generalization across aircraft types and engagement regimes is still underexplored.

3.2.5. Scalability to multi-aircraft

Moving from 1v1 to multi-aircraft engagements introduces coordination and communication challenges that current benchmarks only partially capture.

- **Growth of interactions**: Multi-agent settings suffer from exploding joint action/state spaces, exacerbating sample complexity and convergence issues.
- **Communication limits**: Most MARL formulations do not model intermittent communications, which make robust coordination challenging.

3.2.6. Safety and constraint satisfaction

Air combat is safety-critical even in simulation, and policies must remain within feasible flight envelopes.

- **Robustness to disturbances**: Domain shift (wind, sensor noise) can degrade performance; tube-based or disturbance-aware methods are promising but may require unrealistic information (opponent future states) or induce overly conservative behavior.
- **Constraints are external patches**: Termination logic, envelope protection and saturation handling are often implemented as wrappers rather than being integrated into the learning objective, raising questions about optimality and transferability.

3.2.7. Evaluation methodology

A major gap is the lack of standardized evaluation protocols.

- **Win-rate is insufficient**: Many works report win-rate against selected baselines, but omit calibrated difficulty levels, uncertainty or broader metrics.
- **Benchmark fragmentation**: Results are spread across custom simulators, proprietary environments and different aircraft models, making direct comparison difficult.
- **Opponent selection bias**: Performance often depends strongly on the opponent set; without multiple opponents or cross-aircraft testing, claims of "strong tactics" may not generalize.
- **Limited reproducibility**: Even when Gym-style environments are released, full reproducibility requires consistent wrappers, reward definitions and evaluation scripts, which are often incompletely specified or missing.

3.3. Contributions

In this thesis, the gaps identified above are addressed by operating on a modular simulation suite for DRL in autonomous air combat, which also provides efficient benchmarking capabilities. The infrastructure provides:

- **Modular design**: Enable fast prototyping of new environments, reward designs and agent architectures to encourage community-driven research.
- **Benchmarking capabilities**: Provides a rich and curated collection of scenar-

ios, such as maneuvering, target and adversarial engagement, with standardized evaluation metrics and reference baselines.

- **Reproducibility**: Unlike much of the existing literature, this thesis provides clean, well-structured and fully documented code baselines that can be installed in a containerized environment and executed exactly as reported. This makes results easy to verify, simplifies fair comparisons and lowers the barrier for others to improve, extend or adapt the work.
- **Advanced logging**: This work adopts a detailed, end-to-end logging strategy for both training and evaluation. It integrates advanced experiment tracking with Weights & Biases [9], complemented by rich command-line logs, structured text outputs and systematic checkpointing of model weights. For evaluation, the pipeline adds high-fidelity trajectory recording and visualization in TacView [26], enabling fine-grained inspection of maneuvers and emergent behaviors. Together, these tools improve debugging, support rigorous comparisons across runs and make performance analysis transparent and repeatable.

By delivering this infrastructure and a systematic analysis of DRL methods for air combat, this thesis supports stronger scientific rigor, improved reproducibility and greater operational trust in autonomous control systems.

4 | Gymnasium Toy Model

In this chapter, a set of simplified Gymnasium environments developed as *toy models* for aerial combat tasks are presented, arranged from the simplest to the most complex. These environments were used to prototype and validate our approach, thanks to their faster simulation speed and their adjustable level of complexity.

4.1. Introduction

In the early stages of developing autonomous air combat agents, it is important to follow a structured, step-by-step approach. Real-world scenarios are complex, high-dimensional and expensive to simulate. To better handle these challenges, we first built a set of simplified environments, each designed to focus on a specific aspect of the air combat problem.

These environments have been implemented as custom classes, all inheriting from `Gym.Env` [19], and both training and evaluation have been performed by the use of separate, dedicated scripts for each environment, in order to keep a solid and reproducible codebase. The goal of this preliminary phase is to validate our RL framework and algorithms employed on simple tasks, and to test reward shaping, training stability and emergent behaviors in a controlled setting. Starting from these reduced scenarios makes it easier to spot issues early and refine the approach step-by-step before switching to more realistic simulations.

4.2. FollowPoint

4.2.1. Description

FollowPoint defines a lightweight 3D point-tracking environment that we use a first benchmark for Deep RL in spatial control. The agent must reach a fixed target position in space using a compact observation vector and a shaped, continuous reward. The environment uses a simplified kinematic update: at each step, the agent chooses a change in heading $\Delta\theta$ and a vertical displacement Δz. This setup avoids full flight dynamics and inertial

effects, while still preserving sequential decision-making through state integration. It enables fast iteration on reward scaling, normalization and training stability before moving to higher-fidelity environments.

It is worth noting that two different scenarios have been analyzed:

- **Fixed point**: using always the same fixed point throughout the episodes
- **Random fixed point**: using a randomly generated fixed point for each episode

4.2.2. Observation Space

The observation state is a eight-dimensional vector:

$$\texttt{obs} = [x_a, y_a, z_a, v_a, \theta_a, x_t, y_t, z_t]$$

where:

- x_a, y_a, z_a represent the agent's position
- v_a, θ_a represent the agent's velocity and heading angle
- x_t, y_t, z_t represent the target's position

4.2.3. Action Space

The action is a two-dimensional continuous vector a:

$$a = [\delta\theta, \delta z], \qquad \delta\theta \in [-0.5, 0.5], \quad \delta z \in [-0.25, 0.25]$$

representing the agent's variation in heading and altitude. This allows for a simple and fast control system to simulate.

4.2.4. Motion Equation

Once the action has been selected, the heading angle is normalized between $[-\pi, \pi]$:

$$\theta_{t+1} = ((\theta_t + \Delta\theta_t + \pi) \bmod 2\pi) - \pi$$

The agent's position is updated at each step with the following equations:

$$\texttt{state} = \begin{cases} x_{t+1} = x_t + v_t \cos(\theta_{t+1})\,\Delta t \\ y_{t+1} = y_t + v_t \sin(\theta_{t+1})\,\Delta t \\ z_{t+1} = z_t + \Delta z_t \end{cases}$$

The target's position remains unaltered throughout the whole episode.

4.2.5. Reward Design

In this environment, the reward shaping adopts a continuous feedback to smoothly direct the agent towards the target:

$$r_t = \begin{cases} +\texttt{improvement}, & \text{if } \texttt{improvement} > 0 \\ -0.5 \cdot |\texttt{improvement}|, & \text{otherwise} \end{cases} \tag{4.1}$$

$$r_{success} = \begin{cases} +50, & \text{if } d_{curr} < 0.5 \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

where:

- $d_{curr} = p_{curr} - p_{target}$, representing the current distance from the target
- $d_{prev} = p_{prev} - p_{target}$, representing the distance from the target at the previous step
- $\texttt{improvement} = d_{prev} - d_{curr}$, representing how much the distance between agent and target has changed

4.2.6. Discussion

FollowPoint is intentionally designed as a minimal yet non-trivial benchmark for spatial control, where learning performance is dominated by reward formulation and scaling, rather than by complex dynamics. By replacing full flight mechanics with a simpler kinematic integration, the environment preserves the sequential nature of decision making while remaining lightweight enough to enable rapid iteration and controlled ablation studies before transitioning to higher-fidelity simulators.

A central aspect of FollowPoint is the use of an improvement-based shaping signal, computed as the step-to-step reduction in Euclidean distance to the target. This choice provides dense feedback aligned with the task objective: the agent is rewarded when it

makes measurable progress and penalized when it regresses. In practice, this formulation reduces the tendency to exploit local behaviors, because the reward is tied to net progress rather than to absolute proximity at a single timestep. The asymmetric penalty further biases optimization toward conservative, monotonic approaches: moving away from the goal is discouraged, but without introducing excessively large negative returns that could destabilize early exploration. The success bonus complements the shaping term by making task completion unambiguous. Although the improvement reward already encourages convergence, the sparse terminal bonus creates a clear separation between approaching the target and actually reaching it within the acceptance radius.

The environment also allows studying generalization through two scenarios: a fixed target across episodes and a randomly sampled target per episode. The fixed-point setting primarily tests whether the agent can discover and stabilize a reliable control strategy under a stationary objective, making it suitable for debugging reward scaling and training stability. The random fixed-point variant, instead, stresses robustness and prevents memorization of a single trajectory, requiring the policy to condition its actions on the relative geometry encoded in the observation vector.

4.3. FollowTrajectory

4.3.1. Description

FollowTrajectory is the next step on the toy model roadmap, as it introduces a target that now moves dynamically on a randomly-generated 3D Bézier curve [42]. Compared to FollowPoint, this task uses a moving reference: the agent must follow a trajectory that unfolds over time. This means it has to learn not only how to get close in space, but also how to keep up with the motion and anticipate where the reference is going next.

At the beginning of each episode, we generate a new Bézier curve by sampling a fixed number of control points inside a define 3D space. By changing the path at every reset, the agent is exposed to many different shapes, which encourages the policy to generalize and remain robust across a wide range of geometric configurations.

4.3.2. Observation Space

The observation state is a seventeen-dimensional vector:

$$\texttt{obs} = [x_{rel}, y_{rel}, z_{rel}, d, \alpha, x_a, y_a, z_a, x_t, y_t, z_t, vel, dir] \tag{4.3}$$

where:

- $x_{rel}, y_{rel}, z_{rel}$ represent the relative position between agent and target
- d represents the distance value between agent and target
- $\alpha = \arctan(y_{rel}, x_{rel})$
- x_a, y_a, z_a represent the current agent position
- x_t, y_t, z_t represent the current target position
- vel, dir represent the agent's velocity and direction vector

4.3.3. Action Space

The action is a three-dimensional continuous vector a:

$$a = [\delta v, \delta\psi, \delta\theta], \qquad \delta v \in [-0.1, 0.1], \quad \delta\psi, \delta\theta \in [-0.4, 0.4]$$

representing the agent's variation in velocity, yaw angle and pitch angle. This new action vector allows for smoother and improved movement in the 3D space, moving one step closer to the actual dynamics of a fixed-wing aircraft.

4.3.4. Motion Equation

Once the action has been selected, the yaw angle, pitch angle and velocity are all updated and normalized:

$$v_{t+1} = v_t + \Delta v_t, \quad v_t \in [0.1, 1.5] \tag{4.4}$$

$$\psi_{t+1} = ((\psi_t + \Delta\psi_t + \pi) \bmod 2\pi) - \pi, \quad \psi_t \in [-\pi, \pi] \tag{4.5}$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t, \quad \theta_t \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right] \tag{4.6}$$

therefore, the new agent position is computed as:

$$\texttt{state} = \begin{cases} x_{t+1} = x_t + v_{t+1}\cos(\theta_{t+1})\cos(\psi_{t+1})\Delta_t \\ y_{t+1} = y_t + v_{t+1}\cos(\theta_{t+1})\sin(\psi_{t+1})\Delta_t \\ z_{t+1} = z_t + v_{t+1}\sin(\theta_{t+1})\Delta_t \end{cases} \tag{4.7}$$

The target's position is updated to the next point along the precomputed trajectory.

4.3.5. Reward Design

$$r_{dist} = \exp\left(-0.5 \cdot \left(\frac{d}{10}\right)^1\right), \quad r_{dist} \in [0,1] \tag{4.8}$$

$$r_{heading} = \frac{1 - \cos\left(dir_{agent} \times dir_{target}\right)}{2}, \quad \cos\left(dir_{agent} \times dir_{target}\right) \in [-1,1], \quad r_{heading} \in [0,1] \tag{4.9}$$

$$r_{align} = \frac{1 + \cos\left(pos_{agent} \times dir_{target}\right)}{2}, \quad \cos\left(pos_{agent} \times dir_{target}\right) \in [-1,1], \quad r_{align} \in [0,1] \tag{4.10}$$

$$r_t = \alpha\ r_{dist} + \beta\ r_{heading} + \gamma\ r_{align} - 1, \quad r_t \in [-1,0] \tag{4.11}$$

where:

- d is the distance value between the agent and the target
- $dir_{agent}, dir_{target}$ represent the agent and target direction vectors
- pos_{agent} represents the agent's position vector, used to check if it is aligned with the target
- $\alpha, \beta, \gamma \in (0,1]$ are weights. In this case, we used $\alpha = 0.5, \beta = 0.3, \gamma = 0.2$

The reward used praises the agent if it reduces distance, heads the same direction as the target and if it aligns correctly with the current heading of the target, to put itself right behind it.

4.3.6. Discussion

FollowTrajectory extends the FollowPoint benchmark by introducing a moving reference that evolves along a randomly generated 3D Bézier curve. This shift changes the nature of the control problem: instead of converging to a static goal, the agent must continually regulate its motion to remain close to a target whose position and direction vary over time. As a result, the task implicitly tests tracking and anticipation capabilities, since minimizing instantaneous distance alone is insufficient when the reference keeps advancing along the path. The episodic resampling of the curve's control points further prevents memorization of a single route and encourages the policy to generalize across a wide set of geometries, from smooth arcs to sharper turns and altitude changes. The observation design reflects this transition by combining absolute state information with relative

geometry. Including the relative displacement $(x_{rel}, y_{rel}, z_{rel})$, the scalar distance d and the relative bearing α, provides a compact representation of "where the target is" in the agent's local frame, while the explicit agent/target positions and the agent's direction stabilize learning and reduce partial observability effects.

From a control standpoint, the expanded continuous action space is a crucial step toward realistic 3D maneuvering. Allowing the agent to modulate both speed and orientation enables smoother pursuit policies compared to the previous control equations of Follow-Point. At the same time, the bounded updates and normalization in the motion equations constrain the system to a stable operational envelope, which is important for keeping the learning problem well-conditioned.

The reward function is shaped to promote persistent tracking rather than short-term exploitation. The distance component r_{dist} provides dense feedback and decays smoothly with increasing separation, creating a graded incentive to stay near the target. The heading agreement term rewards matching the target's direction of travel, allowing the agent to satisfy an important constraint of future air-combat tasks: air dominance. In a close-range air combat scenario, an attacker is considered "in advantage" if it is placed right behind it's target, keeping itself at a certain distance. The alignment term further biases the agent toward positioning itself consistently with the target's forward direction, which shapes behavior toward a trailing configuration rather than later oscillations. Finally, the negative constant shift makes the per-step reward strictly non-positive, turning the problem into minimizing a cost-like signal: the agent is incentivized to maximize the shaped components as much as possible at every timestep, while preventing unwanted return accumulation from loitering behaviors.

4.4. Tag Environment

4.4.1. Description

This environments marks the transition from a single to a multi-agent setting. It is a simple two-agent 3D "tag" scenario, named after the famous game Tag [76]. Two agents move in continuous 3D space, one acts as the chaser and the other as the evader, but both are controlled in the same way. It is important to note that, unlike previous environments, these agents are "warm started", meaning they start to learn from an already learned policy; in this case, the agents start from the trained FollowTrajectory policy.

At the start of each episode, the agents are placed in one of several random initial setups (e.g., facing each other, one starting behind the other, etc.). The goal of this environment

is to see whether the capabilities learned during FollowTrajectory can be further enhanced, allowing the agent to follow a "sentient" target rather than a fixed, pre-computed trajectory.

Self-Play

Self-Play [51] is a training paradigm in which an agent improves by repeatedly playing (or interacting) against itself (either a clone of its current policy or a pool of past snapshots) so the opponent automatically adapts as the learner gets stronger, creating an implicit curriculum. Common variants include:

- **Pure Self-Play** [57]: the agent plays against an identical copy of its current policy. Simple but often unstable because the opponent changes as fast as the learner and can lead to "mirror" behaviors or cycling in non-transitive games
- **Frozen-Opponent Self-Play** [23]: the agent plays against past versions, so the opponent is sampled from a buffer of frozen checkpoints. This reduces non-stationarity and catastrophic forgetting, and it is the typical choice when the aim is to have steady progress
- **League-based Self-Play** [70]: it is a generalization of the previos case, where a pool of diverse agents in maintained and a skill-based matchmaking is used to pick informative opponents. It improves robustness and reduces overfitting at the cost of extra complexity and compute
- **Fictitious Self-Play** [24]: it is a game theoretic extension where the agent learns best responses against an empirical distribution of opponents, which helps mitigate cycling and pushes the system toward more equilibrium-like, less exploitable strategies
- **Prioritized Fictitious Self-Play** [70]: instead of sampling opponents uniformly, the training pipeline prioritizes opponents that are neither too easy nor-impossible, so the agent spends more time on matchups that maximize learning signal while still preserving diversity through the broader league

In this thesis, we employ a “freeze & swap” self-play regime with two co-evolving agents. Both agents are initialized from the same starting policy, ensuring a symmetric baseline at the beginning of training. Learning then proceeds in alternating phases: we train Agent 0 while keeping Agent 1 frozen as a fixed opponent, and after a predefined number of episodes we swap roles by training Agent 1 against a frozen snapshot of Agent 0. Importantly, at each swap the frozen opponent is updated to the most recent checkpoint

of the other agent, so each learner is consistently challenged by the latest available behavior while avoiding the instability that typically arises when both policies are updated simultaneously.

4.4.2. Observation Space

Since the agents are warm-started with the policy obtained from FollowTrajectory, the observation space must be coherent. Therefore, the observation state is the same seventeen-dimensional vector:

$$\texttt{obs} = [x_{rel}, y_{rel}, z_{rel}, d, \alpha, x_a, y_a, z_a, x_t, y_t, z_t, vel, dir]$$

where:

- $x_{rel}, y_{rel}, z_{rel}$ represent the relative position between agents
- d represents the distance value between agents
- $\alpha = \arctan(y_{rel}, x_{rel})$
- x_a, y_a, z_a represent the current agent position
- x_t, y_t, z_t represent the current target (opponent agent) position
- vel, dir represent the agent's velocity and direction vector

4.4.3. Action Space

The action is again a three-dimensional continuous vector a:

$$a = [\delta v, \delta\psi, \delta\theta], \qquad \delta v \in [-0.1, 0.1], \quad \delta\psi, \delta\theta \in [-0.4, 0.4]$$

representing the agent's variation in velocity, yaw angle and pitch angle.

4.4.4. Motion Equation

Just like in FollowTrajectory, once the action has been selected for each agent, the yaw angle, pitch angle and velocity are all updated and normalized as seen in equations (4.4) to (4.7). The update is the same for both the agents in the environment.

4.4.5. Reward Design

The reward is the same as seen in (4.11), with the addition of the following two components:

$$penalty_{\text{align}} = dir_{\text{agent}_0} \times dir_{\text{agent}_1}, \quad penalty_{\text{align}} \in [-1, 1]$$

$$progress = \begin{cases} dist_{curr} - dist_{prev}, & \texttt{if } behind_{\text{agent}_0} > behind_{\text{agent}_1} \\ dist_{prev} - dist_{curr}, & \texttt{otherwise} \end{cases}$$

where:

- $dir_{\text{agent}_0}, dir_{\text{agent}_1}$ represent the direction vectors of the agents
- $dist_{curr}, dist_{prev}$ represent the current and previous distance between the agents
- $behind_{\text{agent}_0}, behind_{\text{agent}_1}$ represent how long (in steps) each agent has been trailing the other

Therefore, the final reward is:

$$r = \texttt{base reward} - w_{\text{align}} \cdot penalty_{\text{align}} + w_{progress} \cdot progress \tag{4.12}$$

where:

- `base reward` is the FollowTrajectory reward (4.11)
- $w_{\text{align}}, w_{progress}$ are the weights for the additional components

4.4.6. Discussion

The Tag environment marks a deliberate shift from single-agent tracking to an adversarial two-agent setting, while preserving the same low-level kinematics and observation structure used in FollowTrajectory. This continuity is intentional: by warm-starting both agents from a previously trained trajectory-following policy, the environment isolates the multi-agent difficulty without simultaneously changing state representations or control interfaces. In other words, Tag tests whether a policy that can reliably track a scripted reference can transfer its pursuit skills to a sentient target whose behavior actively reacts and counter-optimizes. The observation design further supports transfer and stability. Keeping the same vector as FollowTrajectory preserves compatibility with the warm-started policy and encourages reuse of learned geometric features such as relative displacement, distance and bearing. At the same time, in Tag these features acquire a richer meaning: the target it no longer a point on a pre-computed path but another agent whose future

motion depends on the interaction history. This effectively increases partial observability at the policy level, making the inclusion of both relative and absolute terms beneficial for robust behavior across diverse initial configuration (head-on, tail-chase).

Reward shaping build directly on the FollowTrajectory base reward to retain the core tracking objective, while adding components that address typical failure modes in symmetric pursuit-evasion games. The alignment penalty discourages degenerate solutions where agents collapse into synchronized motion without meaningful chasing dynamics, which can happen when both agents exploit the same local optimum (e.g., flying parallel at a fixed distance). The progress term, defined using the change in inter-agent distance and modulated by which agent has been trailing longer, introduces an explicit temporal notion of advantage: it rewards behaviors that improve the chasing state over time rather than merely maintaining a stable geometry. Together, these additions aim to promote sustained pursuit behavior, where one agent actively closes distance and positions itself behind the other.

4.5. Fight Environment

4.5.1. Description

Fight Environment represent the final phase of the toy model phase. It is built on a solid motion equation, proven to be realistic enough and effective, with a high-dimensional and complete observation space, in order to tackle the most challenging of the tasks. Both agents are randomly initialized in one of the four main configurations for dogfight (head-to-head, tail-to-tail, advantage-disadvantage), while receiving complete observations of each other's position. Training involves "freeze & swap" self-play once again, so each policy is cyclically updated and faces the latest opponent policy.

4.5.2. Tactical Geometry of a WVR Dogfight

Tactical Tracking Angles

To design an effective reward shaping for this task, we focused on standard air-combat angles that describe the relative geometry between the two aircraft and the related tactical advantage:

- **Antenna Train Angle (ATA)**: the angle between the agent's velocity (or heading) vector and the line-of-sight (LOS) vector to the target. ATA is a key quantity in intercept geometry and weapon employment, since it represents how much the

attacker must turn to point directly at target. Keeping ATA small is important to maintain stable tracking and to achieve weapon lock [56, 71].

- **Aspect Angle (AA)**: the angle measured from the target aircraft's tail to the LOS vector connecting the attacker to the target. AA indicates how close the attacker is to the target's six o-clock position (directly behind). In particular, AA = 0° = corresponds to an ideal position of pure pursuit (behind target), while AA = 180° = corresponds to a disadvantage position (target behind us) [56, 71].
- **Heading Crossing Angle (HCA)**: the difference between the attacker's heading and the target's heading, computed as the smallest rotation needed to make the two headings parallel. HCA captures fuselage alignment and pursuit geometry: HCA = 0° = means the aircraft fly parallel in the same direction, while HCA = 180° = indicates a head-on engagement [56, 71].

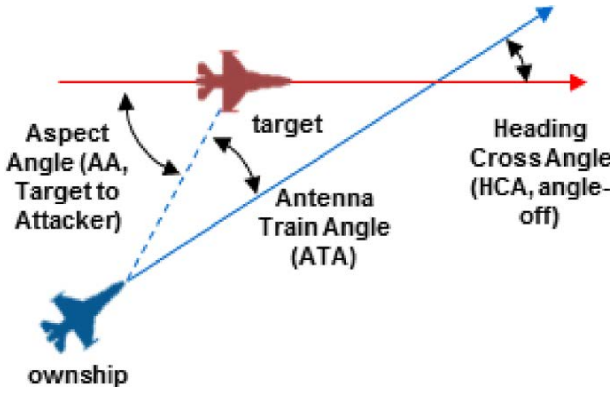


Figure 4.1: Visualization of tactical tracking angles in air-combat geometry

Weapon Engagement Zone (WEZ)

A distinctive aspect of this environments is the explicit modeling of tactical air combat geometry. In real aerial engagements, the Weapon Engagement Zone (WEZ) is the region of space where a platform meets the kinematic and line-of-sight conditions needed to launch a weapon with a high probability of success [56]. We capture this idea through a WEZ cone: at each timestep, the target is considered engaged when it lies inside a solid-angle sector centered on the agent's velocity vector and withing a feasible distance range. This formulation allows the reward function to explicitly encourage the agent to maneuver into advantageous firing positions, promoting tactical behaviors that are consistent with a real-world scenario, while also enabling the possibility of dealing a measurable amount of damage to the target.

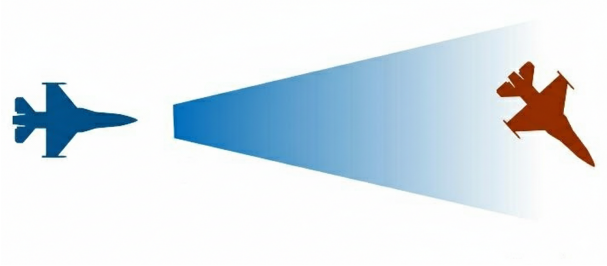


Figure 4.2: Weapon Engagement Zone (WEZ)

4.5.3. Observation Space

Like in the previous environment, the agents are warm-started, so the observation state is, again, the seventeen-dimensional vector (4.3).

4.5.4. Action Space

Once again, the action is a three-dimensional vector a:

$$a = [\delta v, \delta\psi, \delta\theta], \qquad \delta v \in [-0.1, 0.1], \quad \delta\psi, \delta\theta \in [-0.4, 0.4]$$

representing the agent's variation in velocity, yaw angle and pitch angle.

4.5.5. Motion Equation

We used the same, well tested equations showed previously in (4.4) to (4.7).

4.5.6. Reward Design

This is the most complex reward developed within the toy model environments. It is used for both the agents and it aims at having an aggressive fighting scenario. The reward components added on top of the previously used reward (4.12) are:

$$r_{wez-step} = \begin{cases} 0.05, & \text{if enemy in WEZ} \\ 0, & \text{otherwise} \end{cases} \tag{4.13}$$

$$r_{hit} = \begin{cases} 0.1, & \text{if target hit} \\ 0, & \text{otherwise} \end{cases} \tag{4.14}$$

$$r_{win} = \begin{cases} 20, & \text{if target destroyed} \\ 0, & \text{otherwise} \end{cases} \tag{4.15}$$

$$p_{hit} = \begin{cases} -5, & \text{if agent is hit} \\ 0, & \text{otherwise} \end{cases} \tag{4.16}$$

$$p_{lose} = \begin{cases} -20, & \text{if agent is destroyed} \\ 0, & \text{otherwise} \end{cases} \tag{4.17}$$

The final reward can be written as:

$$r = \texttt{tag reward} + r_{wez-step} + r_{hit} + r_{win} + p_{hit} + p_{lose} \tag{4.18}$$

4.5.7. Discussion

Fight represents the culmination of the toy-model roadmap by combining the most complete state information available in the simplified setting with a reward structure explicitly grounded in within-visual-range (WVR) air-combat doctrine. While the underlying kinematic update remains intentionally lightweight, the task difficulty increases substantially because the objective is no longer "tracking" per se, but tactical dominance under an adversarial, non-stationary opponent. Random initialization across canonical dogfight geometries further broadens the distribution of encounter states and forces the learned policy to handle both offensive and defensive regimes within a single training curriculum.

A key design choice is the use of tactical tracking angles to make the reward sensitive to relative geometry in a way that is consistent with real engagement concepts. These angles provide a compact description of whether the agent is effectively pointing at the opponent, whether it is approaching a favorable position, and whether the two aircraft are aligned or crossing. The Weapon Engagement Zone (WEZ) model plays a complementary role by translating tactical positioning into an explicit notion of actionable advantage. Instead of rewarding proximity in isolation, the WEZ term incentivizes the agent to maneuver into a cone-and-range region where a weapon shot is feasible, thereby connecting control actions to an interpretable combat objective. The small per-step WEZ reward encourages persistent maintenance of a firing solution, which is important because momentary alignment can occur by chance during aggressive maneuvering. At the same time, the discrete hit and win bonuses provide sparse but high-value signals that anchor the reward scale to outcomes that matter operationally: landing shots and ultimately destroying the opponent. The corresponding penalties for being hit or destroyed introduce a clear risk

trade-off, discouraging reckless pursuit strategies that might yield occasional offensive opportunities at the cost of repeated vulnerability.

Training remains based on self-play, which is essential in this setting because the optimal behavior depends strongly on the opponent's skill and style. By cyclically updating policies against the latest opponent checkpoint, learning naturally progresses through an implicit curriculum: as one agent improves in pursuit and WEZ maintenance, the other is forced to discover stronger defensive maneuvers, which in turn raises the difficulty for offensive policy updates. Warm-starting from earlier environments further stabilizes this process, since the agents begin with sensible tracking behaviors and can allocate learning capacity to higher-level tactical refinements rather than rediscovering basic 3D control.

5 | Realistic JSBSim Simulator

5.1. Introduction

Having validated the approach to the problem in a variety of simplified kinematic environments, it is possible to move to a higher fidelity setting by integrating our RL agents with JSBSim [8, 65]. JSBSim provides a physically accurate six-degree-of-freedom (6-DoF) aircraft simulation. In this work, the F-16 Fighting Falcon is used as the platform for all subsequent reinforcement learning experiments.

To support systematic research and scalable benchmarking, a modular simulation framework, which links JSBSim to the Gymnasium [19] or PettingZoo [66] API and keeps responsibilities clearly separated, was developed. JSBSim handles the aircraft physics by running the F-16 Flight Data Model (FDM) with realistic, nonlinear dynamics. The task logic is implemented through environment wrappers, which define the objectives and reward functions for each scenario. The agent interface defines normalized observation and action spaces and converts agent outputs into JSBSim control surface and throttle commands; at the same time, after JSBSim completes a simulation step, the interface converts JSBSim outputs into agent observation.

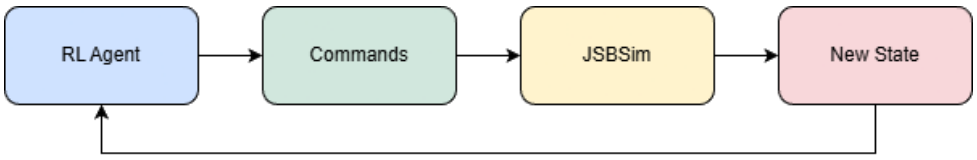


Figure 5.1: Simplified diagram showing the data pipeline between Python RL agents and the JSBSim simulator

Each experimental instance is therefore defined by a JSBSim flight model combined with a task overlay and exposed through the Gymnasium/PettingZoo API. This modular structure makes it straightforward to change tasks, algorithms, or reward shaping. In this work, all training with the real aircraft model was carried out only with three state-of-the-art continuous-control algorithms: Proximal Policy Optimization (PPO) from Chapter 2.3.1, Soft Actor-Critic (SAC) from Chapter 2.3.3 and Recurrent Proximal Policy

Optimization (rPPO) from Chapter 2.3.4, using the respective Stable Baseline3 implementations [46, 47].

Moving from simplified kinematic environments to high-fidelity JSBSim dynamics introduces several additional difficulties, including aerodynamic lag, more complex relationships between states and controls, and a stronger need for anticipatory decision-making. The shift toward rPPO was almost necessary, as its recurrent structure allows the agent to implicitly capture short-term flight history and, as a result, produce decisions that depend on recent context rather than only on the current observation. This kind of temporal memory is especially important in pursuit and tracking tasks, where the agent must account not only for current position and velocity but also for the motion trend and the likely near-future evolution of the trajectory.

As stated in Chapter 2 and 3.3, the implementation of the environment logic is completely algorithm-agnostic, which makes it possible to compare different algorithms under the same physical model and task setup. Also, all training and evaluation metrics were tracked on Weights & Biases (WandB), enabling real time visualization, performance monitoring and reliable experiment tracking. After training, TacView was used for qualitative inspection and visualization of trajectories and agent behavior during testing and interpretation. Fig 5.2 shows how the framework is implemented.

5.2. Realistic Flight Dynamics

JSBSim numerically integrates the nonlinear six-degree-of-freedom (6-DoF) equations of motion for a rigid-body aircraft, enabling high-fidelity simulation with realistic modeling of aerodynamics, propulsion, landing gear and control surface effects [8, 65].

5.2.1. Dynamics model of a 6-DoF fixed-wing aircraft

A fixed-wing fighter jet's motion can be described by combining translation (how the center of gravity moves in space) and rotation (how the aircraft changes its attitude). Together, these are the six degrees of freedom (6-DoF) of a rigid body: three for position and three for orientation.

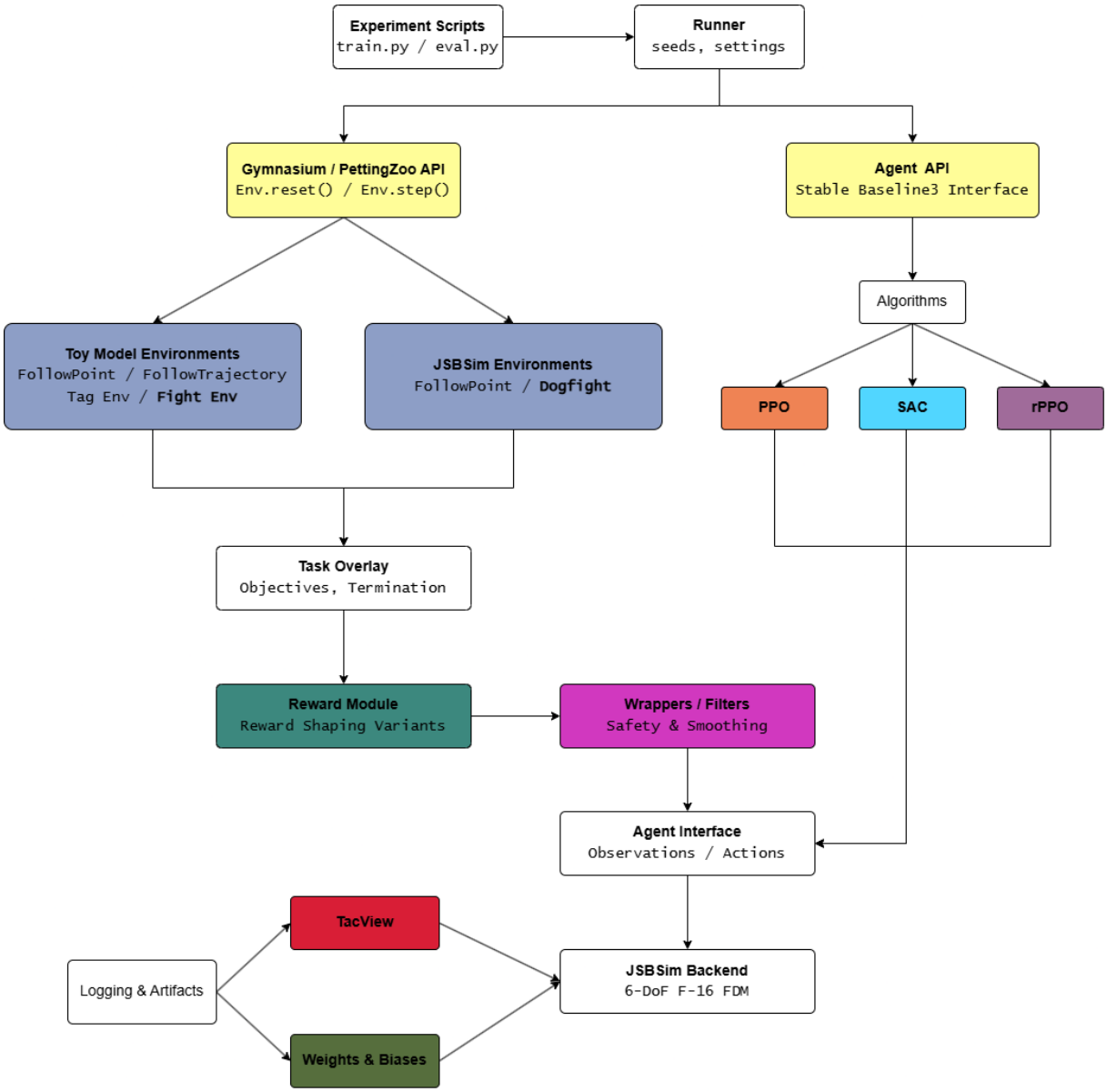


Figure 5.2: Block diagram showing the full modular framework. Colored blocks represent modular components.

Center of Gravity

The point marked G in Fig. 5.3 is the aircraft center of gravity. The dashed curve labeled "trajectory of G" represents the path followed by this point in space. The aircraft does not simply "move forward": it can climb, descend and turn, so the trajectory is generally curved and changes over time depending on speed, forces and control inputs. In the inertial (Earth) reference, the position of G is described with axes like x_E, y_E, z_E. In the figure, these axes define a fixed frame tired to the Earth (often treated as inertial for flight

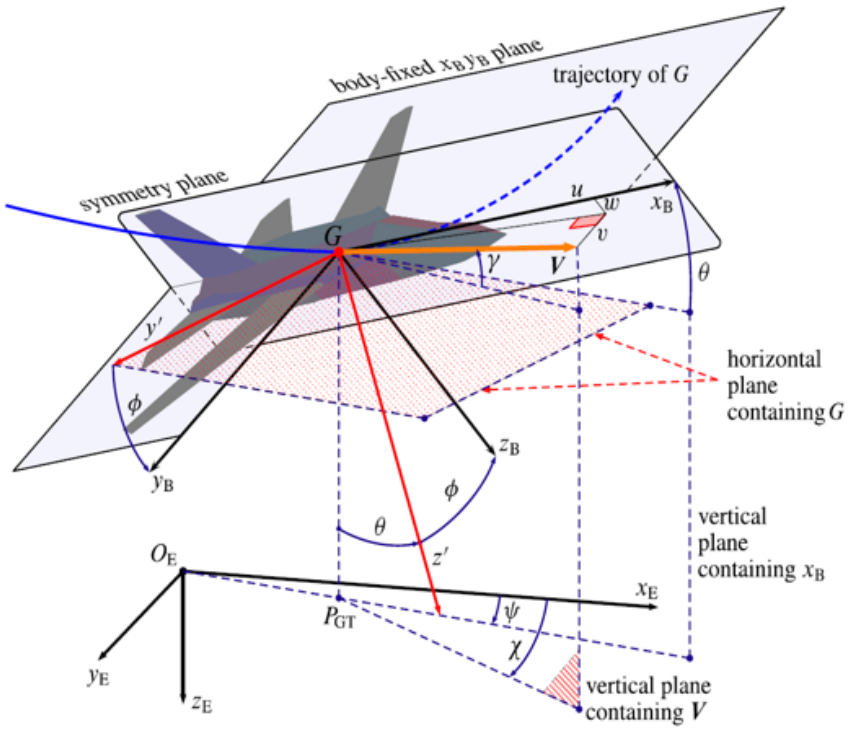


Figure 5.3: Reference frames and attitude angles used to describe a 6-DoF rigid-body aircraft motion. [14]

dynamics over short time scales). The aircraft's equations of motion are often written by tracking how G moves in this Earth frame, while forces and moments are easier to compute in aircraft-related frames.

Body Axes and Velocities

The aircraft-fixed frame is the body frame, shown by axes such as x_B, y_B, z_B in Fig. 5.3. This frame moves and rotates with the aircraft:

- x_B points forward along the fuselage
- y_B points to the right wing
- z_B points downward (typical aerospace convention)

In this body frame, the aircraft's velocity is decomposed into three components:

- u: forward (along x_B)
- v: lateral (along y_B)
- w: vertical (along z_B)

These components are important because aerodynamic forces depend strongly on how the airflow meets the aircraft relative to its body axes. For example, if v is large, the aircraft is "slipping" sideways; if w is large, the aircraft has strong vertical motion relative to its nose direction.

Attitude: Roll, Pitch, Yaw

The aircraft's orientation relative to the Earth frame is described by three angles, which appear in Fig. 5.3:

- **Yaw** ψ: rotation around the Earth vertical axis (turning left/right in the horizontal plane). In the figure, it is associated with the angle measured in the horizontal plane
- **Pitch** θ: nose-up or nose-down rotation. It describes how much the forward direction tilts above or below the horizontal plane
- **Roll** ϕ: rotation around the forward axis x_B. It corresponds to "banking" the wings left or right

The figure highlights these angles using vertical and horizontal reference planes. Conceptually, yaw sets the heading in the horizontal plane, pitch sets the climb or dive angle and roll sets the bank angle that enables turning through lift tilt.

Angular Motion and Moments

While translation describes how G moves, rotation describes how the aircraft's attitude changes. The rotational dynamics are driven by moments (torques) around the body axes. These moments are created by:

- aerodynamic forces, mainly from wings and tail
- thrust effects, from engine alignment (thrust vectoring) and changes in thrust
- gravity acting at the center of mass, indirectly, through how it interacts with attitude and lift

In the rigid-body 6-DoF model, the aircraft's angular rates (often noted as p, q, r for roll, pitch, yaw rates) evolve according to the applied moments and the aircraft's inertia. Even if the aircraft is moving fast the attitude can remain nearly constant.

Fixed-wing aircraft control surfaces

Control surfaces are the primary means by which a pilot or an autopilot generates the moments needed to control the aircraft attitude described in Fig. 5.3 (roll ϕ, pitch θ and yaw ψ). In particular, Fig. 5.4 highlights the main aerodynamic control surfaces:

- **Ailerons**: positioned on the wings, mainly produce roll (ϕ) by creating a lift difference between left and right wing
- **Elevator**: positioned on the horizontal tail, mainly controls pitch (θ) by changing the tail lift and thus the nose-up/down movement
- **Rudder**: positioned on the vertical tail, mainly controls yaw (ψ) by generating a lateral force and a yawing moment

By deflecting these surfaces, the aircraft modifies local airflow, changes the resulting forces and moments, and therefore updates both its orientation and, indirectly, the trajectory of its center of gravity.

Figure 5.4: Main aircraft control surfaces and their typical deflection directions.

In addition to these aerodynamic surfaces, propulsion and drag devices can also contribute to control. Engine thrust directly affects longitudinal acceleration and climb performance, but it can also generate moments when thrust is applied asymmetrically: aircraft with multiple engines can use differential thrust to produce yaw (and, in some cases, roll), especially at low airspeeds. Moreover, some platforms feature thrust vectoring (Fig. 5.6), where the exhaust direction is deflected to create additional pitch and yaw moments, improving agility and control authority even in high Angle-of-Attack (AoA) regimes. Finally, airbrakes (or speed brakes) are used to rapidly increase drag, allowing quick deceleration and energy management; depending on their placement and symmetric or asymmetric deployment, they can also introduce small attitude effects while helping the aircraft adjust spacing, timing and approach geometry.

Figure 5.5: F16's single-engine afterburner, which cannot provide thrust vectoring.

Figure 5.6: F22's thrust vectoring nozzles.

Why Multiple Frames Are Needed

A key message of Fig. 5.3 is that different parts of the problem are simplest in different frames:

- The Earth (inertial) frame is convenient to track the trajectory and global position of the aircraft
- The body frame is convenient to express forces, moments, and control inputs, because control surfaces and engine thrust are fixed to the aircraft
- Frames aligned with the airflow (often called wind and stability frames) are useful to define aerodynamic angles like Angle of Attack (AoA, α) and Sideslip Angle (AoS, β), which strongly affect lift and drag.

In practice, the simulation continuously converts vectors between these frames.

5.2.2. Translational and Rotational Dynamics

The aircrafts translational motion is described by Newtons second law applied to the center of gravity.

$$m\frac{d\mathbf{V}}{dt} + m(\boldsymbol{\omega} \times \mathbf{V}) = \mathbf{F}_{aero} + \mathbf{F}_{prop} + \mathbf{F}_{gravity} \tag{5.1}$$

In JSBSim, this balance is typically written in the body frame, meaning that the velocity vector $\mathbf{V}$ is expressed along the aircraft-fixed axes. Because the body frame rotates with the aircraft, the time derivative of velocity is not the same as in an inertial frame: an additional term appears, $\boldsymbol{\omega} \times \mathbf{V}$, which accounts for the apparent acceleration caused purely by the rotation of the reference frame. Physically, this term captures effects such as the sideways and vertical components that arise when the aircraft is turning or pitching while moving forward. The forces present in (5.1) are grouped into three main compo-

nents: aerodynamic forces (lift, drag, and side force, depending on airspeed and angles), propulsion forces (thrust and any additional contributions from the engine model), and gravity, which must be expressed in body axes to be combined consistently with the other forces. Together, these terms determine how the aircraft speeds up, slows down, climbs, descends, or translates laterally.

Rotational motion is governed by Eulers equations for a rigid body, which relate the aircrafts angular acceleration to the applied moments.

$$\mathbf{I}\frac{d\boldsymbol{\omega}}{dt} + \boldsymbol{\omega} \times (\mathbf{I}\boldsymbol{\omega}) = \mathbf{M}_{aero} + \mathbf{M}_{prop} + \mathbf{M}_{gear} \tag{5.2}$$

Here, the key state is the body angular velocity vector ω, which represents how fast the aircraft is rotating about its roll, pitch, and yaw axes. The inertia tensor $\mathbf{I}$ captures how the aircrafts mass is distributed and therefore how resistant it is to changes in rotation. As in translation, using the body frame introduces coupling terms, and the rotational equation includes the cross product $\boldsymbol{\omega} \times (\mathbf{I}\boldsymbol{\omega})$. This cross-product term represents gyroscopic and coupling effects: even if no external moment is applied, a rotating rigid body can generate internal apparent torques because its angular momentum changes direction as the aircraft rotates. In practice, this is what makes aircraft rotational dynamics strongly coupled. On the right-hand side of (5.2) it is possible to see the sum of the moments generated by different sources: aerodynamic moments (mainly from control surfaces and airflow), propulsive moments (e.g., thrust misalignment, differential thrust, or thrust vectoring if modeled), and landing gear moments (relevant during ground operations, braking, or gear contact). Combined, these moments determine how the aircrafts attitude evolves over time and how quickly it can respond to control inputs.

5.2.3. Coupled Nonlinear Integration

In a full 6-DoF model, translational and rotational dynamics are not independent: they are strongly coupled and nonlinear. The forces that accelerate the aircraft depend on its attitude and air-relative motion, while the moments that rotate the aircraft depend on the same airflow and on how the vehicle is translating. Because of this mutual dependence, the equations form a coupled system of ordinary differential equations (ODEs) that must be integrated together over time.

$$\dot{\mathbf{X}}_{pos} = \mathbf{T}_{body \to ECEF} \cdot \mathbf{V} \tag{5.3}$$

$$\dot{\mathbf{V}} = \frac{1}{m}\left(\mathbf{F}_{aero} + \mathbf{F}_{prop} + \mathbf{F}_{gravity}\right) - \boldsymbol{\omega} \times \mathbf{V} \tag{5.4}$$

$$\dot{\boldsymbol{\omega}} = \mathbf{I}^{-1} \left(\mathbf{M}_{aero} + \mathbf{M}_{prop} + \mathbf{M}_{gear} - \boldsymbol{\omega} \times \mathbf{I}\boldsymbol{\omega} \right) \tag{5.5}$$

At each simulation step of duration Δt, the simulator updates the aircraft state by propagating three main components: position, linear velocity and angular velocity. Position is usually maintained in an Earth-fixed frame, while velocities and forces are more conveniently handled in the body frame. This leads to the first propagation equation (5.3): in simple terms, the aircraft moves in the simulated world according to its current velocity direction, but that velocity must be expressed in the world axes to update the global position consistently. Eq. (5.4) updates the body-frame linear velocity: the total linear acceleration is given by the sum of the aerodynamic, propulsive and gravitational forces, divided by the mass m. However, because the velocity is expressed in the rotating body frame, the corrective term $-\boldsymbol{\omega} \times \mathbf{V}$ must be included. This term represents the apparent acceleration introduced by expressing the dynamics in a frame that is rotating with the aircraft, and it captures the fact that "straight" motion in a rotating frame does not behave like straight motion in an inertial frame. Eq. (5.5) updates the angular velocity ω. The angular acceleration depends on the new applied moments, mapped through the inverse of the inertia tensor $\mathbf{I}^{-1}$. As in the standard rigid-body formulation, the term $-\boldsymbol{\omega} \times (\mathbf{I}\boldsymbol{\omega})$ accounts of gyroscopic and inertial coupling effects, which become especially relevant during aggressive maneuvers. This is one of the reasons aircraft rotational dynamics can show cross-axis interaction, such as roll inducing yaw or pitch responses depending on the inertia properties and aerodynamic configuration.

Since these equations are nonlinear and coupled, they are solved numerically rather than in closed form. In practice, simulation engines like JSBSim typically use either fixed-step integrators or variable-step methods such as RungeKutta schemes [21]. Variable-step integration can improve stability and accuracy when the dynamics become fast, while fixed-step integration is simpler and often preferred for real-time constraints or synchronized simulation loops. In both cases, at every step the simulator recomputes forces and moments from their respective models, then integrates the state forward, producing a consistent evolution of position, velocity, and attitude over time.

5.2.4. Aircraft Configuration

All reinforcement learning tasks simulated on JSBSim in this work use the General Dynamics/Lockheed Martin F-16 Fighting Falcon [37] fighter jet. The F-16 is a highly agile, afterburning fighter that can perform demanding aerobatic maneuvers, operate in the transonic regime, and execute effective energy-management tactics. These characteristics make it a realistic platform for testing control policies under challenging flight conditions.

Figure 5.7: A U.S. Air Force Lockheed Martin F-16C Fighting Falcon in flight [38].

From a modeling standpoint, JSBSim provides a detailed representation of key behaviors that strongly affect learning. The aircraft exhibits a realistic flight envelope, including lift degradation at high angle of attack and stall-related effects, as well as operational constraints such as G-limits. The model also captures axis coupling, where control actions along one axis can generate secondary effects on other axes (for example, roll inputs producing adverse yaw, or elevator deflection inducing additional roll/yaw tendencies). Propulsion is modeled with a nonlinear engine response, including realistic thrustdrag behavior and throttle dynamics such as delays, which are important for energy control. Finally, the simulation includes actuator dynamics and surface deflection limits, meaning that control surfaces do not respond instantaneously but follow rate limits and incremental motion constraints. This combination makes the F-16 an ideal, yet non-trivial, benchmark for reinforcement learning in both flight control and dogfighting scenarios.

Beyond performance and fidelity, the F-16 was chosen for practical reasons related to data availability and credibility. The aircraft is extensively documented: many references and technical materials are publicly accessible, which supports more reliable parameterization and validation of the simulation model. In contrast, modern fighters such as the F-22 or F-35 have limited publicly available details, and their design focus is largely oriented toward beyond-visual-range (BVR) engagements rather than classic within-visual-range dogfighting. As a result, building a trustworthy high-fidelity simulation for these aircraft is significantly more complex and would introduce additional uncertainty. Moreover, the F-16 has been widely adopted both in aerospace research and in many RL researches about Autonomous Air Combat, including within-visual-range dogfighting, which further motivates its use as a standard reference platform.

5.2.5. Agent Interaction Frequency

A key design choice the proposed setup is to decouple the high-frequency physics integration of JSBSim from the lower-frequency decision making of the RL agent. JSBSim advances the aircraft dynamics at 60 Hz ($\Delta t_{\text{sim}} \approx 0.0167s$). If the policy were queried at every simulation step, the agent would be forced to produce a new control command at an extremely high rate, leading to rapid control switching that can add noise, reduce stability and make learning harder. Instead, an action-hold scheme was adopted: the agent selects an action at time t, and that same action is applied for the next N simulation steps:

$$a_t \rightarrow \text{applied for steps } [t, t+N-1] \tag{5.6}$$

This results in an effective decision frequency of $f_{\text{agent}} = \frac{60}{N}$Hz. This decoupling brings two main benefits. First, it often improves control quality and training stability, because the aircraft experiences smoother inputs and the agent observes the consequences of each decision over a meaningful time window. Second, it can improve computational efficiency and decision throughput: the policy does not need to run at every simulator step, but only once every N steps, which can reduce inference overhead and sometimes speed up training and evaluation, especially when policies are large or when many environments are run in parallel.

Practical Choice of Δt and Action-Hold N

To validate the effect of simulator time step and agent interaction frequency, several controlled experiments were run over multiple hours. All runs used the same hyperparameters and training setup, and each configuration was trained for 5 million environment steps (while roughly 20 million steps are typically needed for full convergence). Even at 5 million steps, the learning curves and the final distance-to-target are sufficient to clearly distinguish configurations that are working from those that are not. Table 5.1 summarizes the outcomes.

Simulator step Δt	Agent steps N	Outcome	Notes
$\frac{1}{10}s$	10	OK	Stable learning
$\frac{1}{10}s$	8	OK	Stable learning
$\frac{1}{10}s$	1	Slow Learning	Better fine control
$\frac{1}{60}s$	10	Does not learn	-
$\frac{1}{60}s$	8	Does not learn	-
$\frac{1}{60}s$	1	Does not learn	-

Table 5.1: Summary of Δt and action-hold experiments

Overall, varying the action-hold interval N between 8 and 10 did not produce large qualitative changes when $\Delta t = \frac{1}{10}s$: both settings led to successful learning. Reducing N to 1 (policy queried every step) did not prevent learning at $\Delta t = \frac{1}{10}s$, but it made training noticeably slower, requiring a significantly larger number of steps to obtain comparable performance. At the same time, the higher decision rate improved fine-grained control and mitigated the tendency to operate close to the stall boundary, likely because the agent could react more frequently to fast state changes.

In contrast, changing the simulator integration step from $\Delta t = \frac{1}{10}s$ to $\Delta t = \frac{1}{60}s$ had a much stronger effect: none of the tested configurations learned reliably at $\Delta t = \frac{1}{60}s$, regardless of the chosen N. This suggests that, in the current setup, the simulator time step is the dominant factor for convergence, while the action-hold interval mainly acts as a secondary tuning knob that influences smoothness and sample efficiency.

Based on these results, $\Delta t = \frac{1}{10}s$ was kept as the default integration step, and an action-hold interval in the range $N \in [1, 12]$ was retained as a practical compromise between stability and responsiveness. If future environments or additional tests show reliable learning with $\Delta t = \frac{1}{60}s$, this choice could be revisited and a new experiments could be performed to evaluate whether the higher-frequency physics integration provides consistent benefits.

5.3. Environment Action-Filtering Wrappers

High-fidelity flight simulation with JSBSim exposes reinforcement learning agents to strongly nonlinear dynamics, actuator limits, and delayed aerodynamic responses. In this setting, naïve exploration can easily drive the aircraft into regimes that are physically valid but undesirable for learning (e.g., deep stall, excessive bank, violent oscillations). To make training stable and to ensure that early exploration remains inside a “learnable”

flight envelope, this framework relies on a small set of lightweight wrapper. These components are designed to be task- and algorithm-agnostic, so the same safety logic can be reused across multiple single-agent and multi-agent environments. Finally, these wrappers integrate cleanly with scalable training. Oftentimes in the pipeline, each individual environment instance is wrapped first (including safety and smoothing), then many copies are executed in parallel using subprocess vectorization (up to 512 environments) [59], and only after that the vectorized environment is wrapped by a *VecNormalize* [60] to normalize observations and rewards. This ordering is important: the safety logic operates on raw, physical flight variables cached from the base environment, while normalization is applied externally for learning stability. As a result, the agent receives normalized inputs, but envelope protection decisions remain grounded in real units (degrees of AoA, m/s calibrated speed, radians of roll), which keeps the protective behavior consistent and interpretable across tasks and training configurations.

5.3.1. Observation Caching for Action-Time Safety

A key design constraint is that safety and filtering must be computed at the moment an action is issued, using the latest available flight state. However, Gymnasium's default *ActionWrapper* interface only receives the action as input, not the current observation. To bridge this gap, a wrapper (ObsCacheWrapper) that stores the most recent observation, both after a reset and after each step, was made. This means that a wrapper can read the aircraft state from the latest cached observation without having to call many time for the observation, increasing performance.

5.3.2. Command Smoothing

JSBSim aircraft dynamics are sensitive to abrupt control changes, especially during early training when policies are noisy. An *ActionSmoothingWrapper* was made, which implements an exponential moving average low-pass filter on the agent's command vector. Given the raw action a_t and the internally stored smoothed action $\tilde{a}_{t-1}$, the wrapper applies:

$$\tilde{a}_t = \tilde{a}_{t-1} + \alpha(a_t - \tilde{a}_{t-1}), \quad \alpha \in (0, 1]$$

Larger α reduces smoothing (with $\alpha = 1$ recovering the original command), while smaller α increases damping. Before smoothing, the action is clipped to the action-space bounds. On reset, the smoothed action state is cleared and initialized from the first received command to avoid startup transients. This wrapper is particularly useful when the learned policy exhibits rapid sign changes or when the environment uses action repeat / control

hold, because a smoothed command sequence tends to produce more consistent aircraft responses and reduces the variance of state transitions seen by the learner.

5.3.3. Envelope Protection and Action Rate Limiting

The core safety component is *EnvelopeProtectionWrapper*, which modifies or constrains agent commands when it detects stall risk, excessive bank or excessively fast command changes. It is implemented as an *ActionWrapper*, so it intercepts commands before they reach the JSBSim interface, and it uses the cached observation (5.3.1) to make decisions.

The wrapper defines two AoA thresholds and a minimum calibrated airspeed. A "stall risk" condition is triggered if the AoA exceeds the critical threshold of it the airspeed falls below the minimum, while a "hard stall" is triggered if the AoA exceeds the hard threshold. When stall risk is detected, the wrapper actively biases the command toward recovery: it enforces a minimum throttle, clamps the elevator into a nose-down-friendly range and softens aileron and rudder authority to avoid aggressive maneuvers that could worsen the stall. In the hard stall case, these constraints become stricter. This mechanism does not fly the aircraft on behalf of the agent; instead, it prevents the agent from repeatedly issuing commands that push the system into dynamics that are difficult to learn from, especially during the exploration-heavy phase of training.

The wrapper also enforces a bank limit: when the absolute value of the roll angle exceeds the limit, it performs an active recovery by commanding an aileron input that rolls back toward wings-level, while also constraining elevator to a small band to avoid compounding the upset with large pitch changes. If the aircraft is beyond 90 degrees of roll (fully inverted regime), it additionally neutralizes elevator and rudder to help return to a recoverable attitude. This is especially important in early learning for tasks like level flight or waypoint tracking, where the agent could discover locally "interesting" but unproductive behaviors such as rolling inverted and oscillating in pitch, which can dominate experience without improving task performance.

Even when the aircraft is inside the envelope, large command jumps can destabilize the closed-loop dynamics or create non-smooth transitions that hinder function approximation. The wrapper implements a per-step change limit on each action dimension. It keeps an internal vector of previous actions and clips the difference between current and previous action into a fixed range. The limit can be either a scalar applied to all dimensions or a vector.

5.3.4. Combination of Multiple Wrappers

For multi-agent environments, applying identical safety logic consistently to each agent is essential; otherwise, subtle differences in wrappers or action pipelines can bias training and evaluation. *MultiAgentActionFilter* addresses this by wrapping a multi-agent environment that exposes multiple observation and action spaces and by applying the same filter chain to each agent action in one centralized pass. Implementation-wise, it creates a lightweight per-agent proxy Gym environment containing only the appropriate action space and a dummy observation space, then it stacks the chosen wrappers (it uses 5.3.1, 5.3.3 and 5.3.2) on that proxy. During each step, it feeds the agents last real observation into the proxy, walks through the action-wrapper chain, and outputs a filtered joint action list in the correct agent order. This design eliminates the need for duplicated filtering logic inside per-agent views and guarantees that both agents are constrained under exactly the same rules.

5.4. LevelFlight

5.4.1. Description

This RL environment is used as a first step into the JSBSim workspace, it serves as a benchmark to understand whether the coordination and communication between our environments and the actual simulator works, and how much harder the problem becomes in this setting. The goal of the task is to teach the agent basic autonomous flying skills, specifically steady, level flight. The reward function encourages the agent to keep a target heading and altitude, remain nearly level (small roll/pitch), and avoid rapid oscillatory maneuvers by penalizing large angular rates.

5.4.2. Observation Space

The observation space vector has been defined once and then used across this and subsequent environments. It is a 20-dimensional vector:

$$\texttt{obs} = [lon, lat, alt, \phi, \theta, \psi, v_{north}, v_{east}, v_{down}, v_{body_x}, v_{body_y}, v_{body_z}, v_c, a_{north}, a_{east}, a_{down}, \alpha, \beta, p, q, r] \tag{5.7}$$

where:

- (lon, lat, alt) represent the longitude (°), latitude (°) and altitude (m) of the agent respectively

- (ϕ, θ, ψ) represent roll, pitch and yaw angles (rad) respectively
- $(v_{north}, v_{east}, v_{down})$ represent the North, East and Down velocities (m/s) respectively in the NED frame
- $(v_{body_x}, v_{body_y}, v_{body_z})$ represent the X, Y and Z velocities (m/s) respectively in the Body frame
- v_c represents the calibrated airspeed (m/s)
- $(a_{north}, a_{east}, a_{down})$ represent the accelerations along the X, Y and Z axis (G) respectively in the Body frame
- α represents the Angle of Attack (AoA), which is the angle (rad) measured between the aircrafts longitudinal axis (roughly where the nose points) and the relative wind (incoming airflow). It strongly affects lift and drag, and if it becomes too large the wing can stall (loss of lift, large drag increase).
- β represents the Sideslip Angle (AoS), which is the lateral angle (rad) measured between the aircrafts forward axis and the relative wind, measured in the horizontal plane of the body. A non-zero β means the airflow hits the aircraft from the side (the aircraft is slipping or skidding), which impacts lateral-directional forces, yawing moments, and overall efficiency.
- (p, q, r) represent the roll, pitch and yaw rates (rad/s) respectively, also known as angular velocities.

5.4.3. Action Space

Like the observation space, the action vector has been defined once and shared across environments. It is a four-dimensional vector:

$$a = [aileron, elevator, rudder, throttle] \tag{5.8}$$

with $(aileron, elevator, rudder) \in [-1, 1]$ and $throttle \in [0.4, 0.9]$. By choosing these normalized ranges, we allow the agent to command its control surfaces fully in a normalized way, which is then converted into effective input by JSBSim, and the throttle command in a regulated range, in order to avoid full throttle and low throttle that could lead to stall.

5.4.4. Reward Design

In this environment, the reward shaping adopts a continuous feedback to smoothly guide the agent to fly leveled and keep same heading as well as altitude. There are three main components:

$$r_{heading} = \exp\left(-\frac{1}{2}\left(\frac{e_\psi}{\sigma_\psi}\right)^2\right) \tag{5.9}$$

$$r_{altitude} = \exp\left(-\frac{1}{2}\left(\frac{e_{\text{altitude}}}{\sigma_{\text{altitude}}}\right)^2\right) \tag{5.10}$$

$$r_{level} = \exp\left(-\frac{1}{2}\left[\left(\frac{\phi}{\sigma_{\text{level}}}\right)^2 + \left(\frac{\theta}{\sigma_{\text{level}}}\right)^2\right]\right) \tag{5.11}$$

where:

- $\sigma_\psi = 2^\circ = \frac{2\pi}{180} rad$
- $\sigma_{\text{altitude}} = 100\ m$
- $\sigma_{\text{level}} = 2^\circ = \frac{2\pi}{180} rad$
- ϕ, θ represent the roll and pitch angles respectively

The errors used in (5.9), (5.10) and (5.11) are calculated as follows:

$$e_\psi = \psi - \psi_{ref}, \quad e_\psi \in [-\pi, \pi]$$

$$e_{\text{altitude}} = h - h_{ref}$$

where:

- ψ, h represent the agent's heading and altitude
- ψ_{ref}, h_{ref} represent the set altitude and heading that the agent must keep

We then average these components and add a penalty to get the final reward:

$$r_{base} = \frac{r_{heading} + r_{altitude} + r_{level}}{3} \tag{5.12}$$

$$c_\omega = \lambda(p^2 + q^2) \tag{5.13}$$

$$r = r_{base} - c_\omega \tag{5.14}$$

where:

- λ represents a tunable parameter to weight the penalty
- p, q represent the roll and pitch rate respectively

5.4.5. Discussion

The LevelFlight environment represents a deliberate entry point into the JSBSim-based research pipeline. Its primary purpose is not to maximize tactical performance, but to validate that the full stack works reliably under high-fidelity flight dynamics. Compared to the kinematic benchmarks used earlier, JSBSim introduces nonlinear aerodynamics, inertia, coupling between axes, and delayed responses that make even seemingly simple control objectives nontrivial. In this sense, LevelFlight functions as a system-level sanity check: if an agent cannot learn to stabilize attitude and maintain a prescribed flight condition, then more complex tasks are unlikely to succeed regardless of reward shaping.

From a learning perspective, the task is designed to isolate core competencies required in all subsequent environments. Maintaining heading and altitude while keeping roll and pitch close to zero implicitly requires the policy to discover stable closed-loop behavior, compensate for simulator-induced lag, and avoid oscillatory inputs that can destabilize the aircraft. These skills are not strictly transferable, as in a more dynamic setting like dog-fighting, the agent will almost never fly fully straight and at the same height, nonetheless this gives us an idea of how the learning responds to our reward design. The environment also provides a controlled setting to reason about safety and stability mechanisms that are essential in high-fidelity flight RL. Even though LevelFlight is conceptually simple, unconstrained exploration can still drive the aircraft into unrecoverable conditions (stall, excessive bank, divergent oscillations). Therefore, it is an appropriate context to test wrappers such as envelope protection and action smoothing, and to verify that they improve training stability without trivializing the task.

5.5. FollowPoint & FollowRandomPoint

5.5.1. Description

With FollowPoint we include two closely related single-agent navigation experiments, FollowPoint and FollowRandomPoint, designed to develop and benchmark point-tracking behavior with the same simulator interface. FollowPoint is a deterministic task in which the aircraft must reach a fixed target position (defined once in the environment settings and kept constant across episodes), with termination conditions enforcing altitude safety limits, a maximum step budget, and success/failure based on distance to target. In contrast, FollowRandomPoint keeps the same overall structure and constraints but increases task diversity by sampling a new target at every reset within a bounded spherical shell around the initial position (optionally also randomizing the initial orientation), so the agent must generalize its guidance behavior across many goal locations.

5.5.2. Observation Space

The FollowPoint environment uses the same observation space already defined in (5.7) and adopted across the earlier environment in this work, in order to keep the state representation consistent and directly comparable between tasks. In particular, the agent receives the identical fixed-size observation vector containing the same flight, kinematic, and attitude-related variables (e.g., position, orientation, velocities, accelerations, and angular rates), without introducing additional task-specific features. This design choice ensures that any performance differences are driven by the environment dynamics and reward structure, rather than by changes in the information available to the policy.

5.5.3. Action Space

The FollowPoint environment also uses the same action space already defined in (5.8) and shared across the previous environments, preserving a consistent control interface throughout the project. The agent outputs the identical set of continuous flight-control commands, without introducing any task-specific actuators or extra degrees of freedom. This choice keeps training and evaluation comparable across tasks, ensuring that behavioral differences arise from the objective and dynamics rather than from changes in how the aircraft can be controlled.

5.5.4. Reward Design

FollowPoint

This reward function is mostly negative, based on classic guidance reward designs which make use of errors to shape a dense reward by penalizing errors. The errors adopted are the following:

$$e_\psi = \psi_c - \psi, \quad e_\psi \in [-\pi, \pi]$$

$$e_\phi = \phi, \quad e_\theta = \theta$$

$$e_h = h_t - h$$

$$d = ||pos_t - pos_a||_2$$

where:

- ψ, ϕ, θ are yaw, roll and pitch angles respectively
- h, h_t are agent's and target's height respectively
- pos_a, pos_t are agent's and target's position respectively

Therefore, we can compute a dense reward and add some sparse components for episode termination:

$$r_{dense} = -w_h \frac{|e_h|}{H_s} - w_\psi \frac{|e_\psi|}{\pi} - w_{lvl}(|e_\phi| + |e_\theta|) - k_d \frac{d}{d_0} \tag{5.15}$$

$$r = r_{dense} + \begin{cases} -500 & h \leq 1000 \text{ or } h \geq 10000 \\ +500 & d \leq 200 \\ 0 & \text{otherwise} \end{cases} \tag{5.16}$$

with:

$$w_h = 1.0, \quad H_s = 100, \quad w_\psi = 2.0, \quad w_{lvl} = 0.2, \quad k_d = 0.1$$

As previously said, this reward is based on penalizing errors:

- $\frac{|e_h|}{H_s}$ penalizes the altitude error
- $\frac{|e_\psi|}{\pi}$ penalizes the heading error, normalized in $[0, 1]$
- $|e_\phi| + |e_\theta|$ penalizes non-leveled flight
- $\frac{d}{d_0}$ penalizes relative distance, with d_0 representing the target distance at the beginning of the episode

FollowRandomPoint

Unlike the previous, this reward function is mostly positive, based on a more goal-reaching shaping which uses different positive (bounded) terms to guide the agent towards optimal goals. These terms are:

$$f_{dist} = \exp\left(-\left(\frac{d}{d_0}\right)^2\right), \quad f_{dist} \in (0, 1] \tag{5.17}$$

$$f_{align} = \cos\left(\psi_c - \psi\right), \quad f_{align} \in [-1, 1] \tag{5.18}$$

$$f_{alt} = \exp\left(-\left(\frac{e_h}{1000}\right)^2\right), \quad f_{alt} \in (0, 1] \tag{5.19}$$

where:

- $d = ||pos_t - pos_a||_2$ is the distance between agent and target, with pos_a, pos_t being the agent's and target's position respectively and d_0 being the initial distance
- $\psi_c = \text{atan2}(E_t - E, N_t - N)$ being the commanded heading, with E, E_t and N, N_t being the agent's and target's East and North coordinates respectively
- $e_h = h_t - h$ being the altitude error, with h, h_t being the agent's and target's altitude respectively

Given these terms, we can calculate the dense reward. Again, the dense reward is then summed to some sparse components marking the success or fail:

$$r_{dense} = f_{dist} + f_{align} + f_{alt} \tag{5.20}$$

$$r = r_{dense} + \begin{cases} -1000 & h \leq 1000 \text{ or } h \geq 10000 \\ -500 & d \geq 10000 \\ +1000 & d \leq 200 \\ 0 & \text{otherwise} \end{cases} \tag{5.21}$$

5.5.5. Discussion

Despite being conceived as simple point-tracking benchmarks, FollowPoint and FollowRandomPoint turned out to be significantly more challenging than initially expected. As navigation tasks, they were somewhat underestimated under the assumption that going to a point would be trivial for modern continuous-control algorithms. In practice, however,

the combination of a high-dimensional observation space, a large number of geometric symmetries in the plane, and the many feasible trajectories that can lead to the same goal produced a learning landscape that was far from straightforward. .

A second key aspect of this study is that the two environments intentionally adopt different reward philosophies to examine how distinct shaping principles behave under a similar objective. FollowPoint relies on a more classical, mostly negative, error-penalization design that densely discourages deviations in altitude, heading, and attitude while steadily reducing distance. FollowRandomPoint, on the other hand, uses a mostly positive, bounded shaping based on distance progress, alignment with the commanded bearing, and altitude consistency, complemented by sparse terminal bonuses and penalties. Comparing these two reward lines of thought within closely related tasks helps highlight how different formulations can affect learning stability, exploration, and the emergence of meaningful guidance behaviors, even when the underlying dynamics and interfaces remain unchanged.

It is also worth noting that FollowRandomPoint provides a useful check on generalization and resistance to overfitting. By sampling a new goal at every reset (and optionally varying the initial orientation), the agent cannot simply memorize a single trajectory or exploit task-specific shortcuts; instead, it must develop a reusable guidance policy that transfers across many target configurations. The stronger performance observed with rPPO in this setting therefore suggests not only better optimization properties on these navigation tasks, but also a higher robustness to overfitting compared to the other tested algorithms.

5.6. Dogfight

5.6.1. Description

The Dogfight environment represents the core benchmark of this thesis and the closest setting to the final research objective: training two autonomous agents to fight each other in a realistic 3D within-visual-range (WVR) air combat scenario, using high-fidelity aircraft dynamics simulated in JSBSim. Dogfight is treated as the most important environment in the experimental pipeline, both in terms of complexity and in terms of relevance to real-world aerial combat. Each agent controls a simulated aircraft whose motion is governed by JSBSim's physics-based flight model. The environment takes place in an open 3D airspace, where both agents must simultaneously handle aircraft control (stability, maneuvering, and envelope constraints) and combat objectives (positioning, pursuit, and evasion). This leads to a learning problem where good performance cannot be achieved

by purely kinematic shortcuts: effective strategies must remain consistent with flight dynamics and the geometric constraints of WVR combat.

To avoid overfitting to a single initial condition and to evaluate robustness across different tactical contexts, the Dogfight environment includes four possible combat scenarios, representing distinct initial configurations of relative position, heading, and engagement geometry. These scenarios are designed to span different opening situations (e.g., favorable, unfavorable, neutral, or symmetric setups), forcing agents to learn behaviors that generalize across varying starting conditions rather than exploiting a single scripted trajectory. As a result, the environment encourages the emergence of adaptable tactical patterns such as gaining angles, managing closure, avoiding overshoots, and attempting to reach advantageous behind-the-target configurations.

No Curriculum or Imitation Learning

A key design constraint of this thesis is that agents are trained without curriculum learning and without imitation learning. There is no staged increase in difficulty, no expert demonstrations, and no handcrafted teacher behaviors. Agents are exposed directly to the full Dogfight setting from the beginning, meaning that the final policies are the result of pure reinforcement learning and rely on emergent behavior driven by the reward signal and self-play interaction. This choice emphasizes the central thesis question: whether realistic WVR dogfighting behaviors can arise from reward-driven optimization alone, and what reward formulations and observability assumptions make this emergence more likely.

Observability: Fully Observable vs Partially Observable

A second major experimental axis concerns observability. While fully observable state representations are common in simulation-based RL (and can simplify credit assignment), real aerial combat is inherently limited by sensing constraints, uncertainty, and incomplete information. To study how realism in state information impacts learned behavior and generalization, the thesis evaluates two settings:

- a fully observable configuration, where each agent has access to a richer description of the engagement state
- a partially observable configuration, where information is restricted to a more realistic subset, requiring implicit memory and inference from observations over time

5.6.2. Observation Space

Fully Observable MDP

In the fully observable setting, the Dogfight environment extends the baseline observation space defined in (5.7) to provide each agent with a complete and explicit description of the engagement state. Concretely, the observation vector retains the same fixed-size core features used throughout the earlier environments but it is augmented with:

- the full state of the opposing aircraft expressed in the same feature format
- a set of dogfight-specific geometric variables that directly capture the relative combat configuration

These additional terms include the line-of-sight (LOS) information and key angular metrics commonly used to characterize WVR engagements (ATA, AA, HCA), together with the instantaneous inter-aircraft distance.

This design converts the learning problem into a true MDP approximation by removing the latent opponent-related factors that are otherwise hidden under partial observability. Most importantly, it provides the policy with the exact variables on which many reward components are computed. Instead of having to infer the opponents position and orientation, the agent can directly observe the engagement geometry and learn consistent actionoutcome associations. In practice, the inclusion of explicit relative features reduces apparent reward stochasticity, improves credit assignment, and accelerates the acquisition of tactically meaningful behaviors, since the agent can attribute reward changes to the key dogfight factors that define advantage in WVR combat.

Partially Observable MDP (POMDP)

The Dogfight environment adopts the same observation space defined in (5.7) and consistently used across the earlier environments in this work, in order to preserve a uniform and directly comparable state representation between tasks. While this choice guarantees that any performance gap can be attributed to the interaction dynamics and the reward formulation rather than to engineered dogfight-specific inputs, it also turns Dogfight into a markedly harder learning problem under partial observability. In particular, the agent is trained in a genuine POMDP regime: it must act using a fixed-size observation vector limited to its own flight and attitude quantities, without direct access to critical opponent-dependent variables.

This limitation has immediate consequences for reinforcement learning. Several reward terms typically used to shape WVR behavior depend on quantities that are either not observable at all or only indirectly inferable over time. As a result, the policy is expected to optimize signals that are partially hidden: the environment can assign reward (or penalty) based on engagement geometry that the agent does not explicitly perceive at the current timestep. This weakens the causal link between action and outcome, increases reward noise from the agents perspective, and makes credit assignment substantially more challenging. In practice, the agent must learn to reconstruct latent combat-relevant state, thus inferring the opponent's configuration, from its own motion history and the delayed consequences of its maneuvers, which demands memory and increases the risk of unstable learning or convergence to trivial behaviors.

5.6.3. Action Space

The Dogfight environment also employs the same action space defined in (5.8) and shared across the previous environments, preserving a consistent control interface throughout the project. Each agent outputs the identical set of continuous flight-control commands, without introducing any dogfight-specific actuators or additional degrees of freedom. This choice keeps training and evaluation directly comparable across tasks, ensuring that behavioral differences stem from the objective and interaction dynamics rather than from changes in how the aircraft can be controlled.

5.6.4. Reward Design

The reward structure is one of the main experimental axes. In the literature on competitive RL and air-combat-inspired tasks, both strictly competitive (zero-sum) formulations and non-zero-sum (or mixed-motive) designs are commonly used, often with significantly different learning dynamics and outcomes. To enable a clearer comparison and to better position the results relative to prior work, this thesis implements and evaluates two different reward designs. This comparison is not purely conceptual: the choice affects optimization stability, policy cycling, exploitable strategies, and the type of behaviors that emerge (e.g., aggressive trading of risk versus conservative stabilization). By including both reward families, Dogfight becomes a controlled testbed to observe how competitive pressure and reward coupling influence emergent WVR tactics.

Zero-Sum Reward

Let the two agents be $i \in \{0, 1\}$. At teach step t, we define a per-agent score $s_i(t)$ and then convert the score difference into a zero-sum reward.

We define some bounds to prevent crashing and letting the agents fly too high or too far, by making use of altitude $h_i(t)$:

$$h_{\min} = 1000 \text{ m}, \quad h_{max} = 10000 \text{ m}$$

We define a crash event as:

$$crash_i(t) \equiv (h_i(t) \leq h_{\min}) \vee (h_i(t) \geq h_{\max})$$

If exactly one agent crashes, the reward is terminal and zero-sum:

$$r_i(t) = -R_{\text{crash}}, \quad r_j(t) = +R_{\text{crash}}, \quad j \neq i \tag{5.22}$$

with $R_{\text{crash}} = 5000$. If both agents crash simultaneously, then the agents receive $r_0(t) = r_1(t) = 0$.

Given the definitions of ATA and AA presented in Chapter 4.5.2, we derive two normalized angular utilities:

$$a_{ata,i}(t) = \cos(\text{ATA}_i(t)), \quad a_{tail,i}(t) = -\cos(\text{AA}_i(t))$$

Then the angular advantage is:

$$a_{ang,i}(t) = \frac{1}{2}(a_{ata,i}(t) + a_{tail,i}(t)) \in [-1, 1] \tag{5.23}$$

We define a range gate:

$$g_r(t) = \begin{cases} \exp\left(-\left(\frac{d(t)-d_{\text{gun}}}{\sigma_r}\right)^2\right) & d(t) \leq d_{\max} \\ 0, & d(t) \geq d_{\max} \end{cases} \tag{5.24}$$

where $d_{\text{gun}} = 450$ m, $\sigma_r = 700$ m and $d_{\max} = 4500$ m.

In addition to that, we also employ a closure shaping term. Given the previous distance

for agent i $d_i^{\text{prev}}(t)$, we define:

$$\Delta_i(t) = d_i^{\text{prev}}(t) - d(t), \quad \Delta_i(t) \in [-60, 60]$$

We normalize it:

$$\text{close}_i(t) = \frac{\Delta_i(t)}{60} \in [-1, 1]$$

and then gate it by angular quality:

$$\text{gate}_i(t) = \max(0, a_{ang,i}(t))$$

Therefore, the closure contribution is:

$$k_{close}\text{gate}_i(t)\text{close}_i(t), \quad k_{close} = 0.35 \tag{5.25}$$

We also add an inversion penalty to discourage inverted flight. We define

$$u_i(t) = \max(0, |\phi_i(t)| - \phi_{thr}), \quad \eta_i(t) = \frac{u_i(t)}{\phi_{thr}} \in (0, 1] \tag{5.26}$$

where $\phi_i(t)$ is the roll angle and $\phi_{thr} = \frac{\pi}{2}$ is the threshold. The inversion score, which is always ≤ 0 is:

$$\text{inv}_i(t) = -\eta_i(t)^2 \tag{5.27}$$

which is then weighted by $w_{inv} = 0.25$.

We can now compute the base dense per-agent score s_i:

$$s_i(t) = w_{ang}a_{ang,i}(t) + w_r g_r(t) + k_{close}\text{gate}_i(t)\text{close}_i(t) + w_{inv}\text{inv}_i(t) \tag{5.28}$$

where the weights are:

$$w_{ang} = 0.70, \quad w_r = 0.30, \quad k_{close} = 0.35, \quad w_{inv} = 0.25$$

To add the WEZ shaping, we define its conditions (depth and angle):

$$d_{WEZ} = 900 \text{ m}, \quad \text{ATA}_{\text{max}}^{\text{WEZ}}0 = 15^\circ$$

Let $\mathbf{1}_{WEZ,i}(t) \in \{0,1\}$ indicate that agent i is in WEZ against opponent j:

$$\mathbf{1}_{WEZ,i}(t) = \begin{cases} 1, & d(t) \le d_{WEZ} \wedge \cos\left(\mathrm{ATA}_i(t)\right) \ge \cos\left(15^\circ\right) \wedge \neg \mathrm{destroyed}_j \\ 0, & \text{otherwise} \end{cases} \tag{5.29}$$

Let $W_i(t)$ be a persistence counter (to count how many steps the target has stayed in the agent's WEZ), updated as:

$$W_i(t) = \begin{cases} W_i(t-1) + 1, & \mathbf{1}_{WEZ,i}(t) = 1 \\ \max\left(W_i(t-1) - 1, 0\right), & \mathbf{1}_{WEZ,i}(t) = 0 \end{cases}$$

Dense WEZ shaping adds:

$$s_i(t) = s_i(t) + k_{WEZ}\mathbf{1}_{WEZ,i}(t), \quad k_{WEZ} = 1.20 \tag{5.30}$$

And a one-time bonus at a given persistence threshold:

$$\texttt{if } W_i(t) = N_{\text{bonus}}, \; s_i(t) = s_i(t) + B_{WEZ} \tag{5.31}$$

where $N_{\text{bonus}} = 15$ and $B_{WEZ} = 2.50$

To get the zero-sum reward, we need to define advantage:

$$\mathrm{adv}(t) = \tanh\left(\alpha(s_0(t) - s_1(t))\right), \quad \alpha = 2.0$$

Then the zero-sum step reward is computed as:

$$r_0(t) = \mathrm{adv}(t), \quad r_1(t) = -r_0(t) \tag{5.32}$$

Finally, we define a kill condition for agent i:

$$kill_i(t) \equiv \mathbf{1}_{WEZ,i}(t) = 1 \wedge d(t) \le d_{kill} \wedge \cos\left(\mathrm{ATA}_i(t)\right) \ge \cos\left(12^\circ\right) \wedge W_i(t) \ge N_{kill}$$

If exactly one agent kills another:

$$r_i(t) = r_i(t) + R_{kill}, \quad r_j(t) = -r_i(t) \tag{5.33}$$

If both kill each other simultaneously:

$$r_0(t) = r_1(t) = 0 \tag{5.34}$$

Dynamic Fight Reward

Let the two agents be $i \in \{0, 1\}$ and the opponent of i be $j = \text{other}(i)$. The reward is not zero-sum: each agent receives its own shaped reward.

Again, we define some bounds to prevent crashing and letting the agents fly too high or too far, by making use of altitude $h_i(t)$:

$$h_{\min} = 1000 \text{ m}, \quad h_{\max} = 10000 \text{ m}$$

We define a crash event as:

$$crash_i(t) \equiv (h_i(t) \leq h_{\min}) \vee (h_i(t) \geq h_{\max})$$

If the agent crashes or goes out of bound, it receives a terminal penalty:

$$r_i(t) = -P_{\text{crash}}, \quad P_{\text{crash}} = 5000 \tag{5.35}$$

Also in this reward function we introduce the inversion penalty to avoid roll beyond 90°. Given the roll angle $\phi_i(t)$ and the threshold $\phi_{thr} = \frac{\pi}{2}$:

$$u_i(t) = \max(0, |\phi_i(t)| - \phi_{thr}), \quad \eta_i(t) = \frac{u_i(t)}{\phi_{thr}} \in (0, 1]$$

Then:

$$r_i^{\text{inv}}(t) = -P_{inv}\eta_i(t)^2, \quad P_{inv} = 0.4 \tag{5.36}$$

Another safety term is introduced: the altitude safety shaping term. Given the constants:

$$h_{danger} = 2000, \quad h_{upper-danger} = 8000, \quad P_{danger} = 10, \quad P_{sink} = 2, \quad B_{alive} = 0$$

Let $\Delta h_i(t) = h_i(t) - h_i(t-1)$, we define a low-altitude normalized danger as:

$$t_{low,i}(t) = \frac{h_{danger} - h_i(t)}{h_{danger} - h_{\min}} \in (0, 1]$$

and an high-altitude normalized danger as:

$$t_{high,i}(t) = \frac{h_i(t) - h_{upper-danger}}{h_{\max} - h_{upper-danger}} \in (0, 1]$$

Therefore we can define a low-band penalty:

$$r_i^{\text{low}}(t) = -P_{danger} \cdot t_{low,i}(t)^2 \tag{5.37}$$

and an high-band penalty:

$$r_i^{\text{high}}(t) = -P_{danger} \cdot t_{high,i}(t)^2 \tag{5.38}$$

An additional sink penalty, used when descending while low ($\Delta h_i(t) < 0$), is also included:

$$s_i(t) = \frac{-\Delta h_i(t)}{50} \in (0, 1], \quad r_i^{\text{sink}}(t) = -P_{sink} \cdot s_i(t) \cdot t_{low,i}(t) \tag{5.39}$$

We can sum up the overall altitude shaping as:

$$r_i^{\text{alt}}(t) = \begin{cases} r_i^{\text{low}}(t) + r_i^{\text{sink}}(t), & h_i(t) < h_{danger} \\ B_{alive} + r_i^{\text{high}}(t), & h_i(t) \geq h_{danger} \end{cases} \tag{5.40}$$

Lastly, we add a distance penalty to discourage disengagement once agents are close:

$$t_{far}(t) = \frac{d(t) - d_{start}}{d_{full} - d_{start}} \in [0, 1], \quad r_i^{\text{far}}(t) = -P_{far} \cdot t_{far}(t)^2 \tag{5.41}$$

where $d_{start} = 5000$, $d_{full} = 15000$ and $P_{far} = 0.6$.

To reduce "fight incentives" when the agents are too low, we adopt an engagement scaling term:

$$\lambda_i(t) = 1 - 0.95 t_{low,i}(t), \quad \lambda_i(t) \in [0.05, 1]$$

Similarly to what we have previously seen, the base shaping is composed of three terms. Let distance be $d(t) = ||p_j - p_i||$ we define a Gaussian distance term:

$$f_{dist}(t) = 2 \left(\exp\left(-\frac{1}{2} \left(\frac{d(t) - d_0}{\sigma} \right)^2 \right) - \frac{1}{2} \right), \quad d_0 = 1200, \quad \sigma = 500 \tag{5.42}$$

so that $f_{dist} \approx [-1, 1]$, positive near d_0. We then define the following unit vectors:

- $\hat{\ell}_{ij}(t)$: unit LOS from opponent j to agent i
- $\hat{\ell}_{i\to j}(t) = -\hat{\ell}_{ij}(t)$: unit LOS from agent i to opponent j
- $\hat{n}_i(t)$: agent nose direction
- $\hat{n}_j(t)$: opponent nose direction

Given:

$$c_{pos}(t) = \hat{\ell}_{ij}(t) \cdot \hat{n}_j(t), \quad c_{ata}(t) = \hat{n}_i(t) \cdot \hat{\ell}_{i\to j}(t), \quad c_{pos}(t), c_{ata}(t) \in [-1, 1]$$

we convert to $[0, 1]$ shaped terms:

$$f_{pos}(t) = \frac{1 - c_{pos}(t)}{2}, \quad f_{align} = \frac{1 + c_{ata}(t)}{2} \tag{5.43}$$

Therefore, the base combat reward is:

$$r_i^{\text{base}} = 0.6 f_{dist}(t) + 0.05 f_{pos}(t) + 0.35 f_{align}(t) \tag{5.44}$$

This reward terms alone are not enough in a complex environment like this one, so we add a closure bonus. Let per-agent stored previous distance be $d_i^{\text{prev}}(t)$, we define the closure as:

$$\Delta d_i(t) = d_i^{\text{prev}}(t) - d(t) \in [-50, 50]$$

and update $d_i^{\text{prev}}(t) \leftarrow d(t)$. Then:

$$r_i^{\text{close}}(t) = k_{close} \Delta d_i(t), \quad k_{close} = 0.02 \tag{5.45}$$

For a dogfight task, a WEZ shaping term is needed to better guide the agents towards the real goal. We defined some WEZ parameters:

$$d_{WEZ} = 1500, \quad \theta_{WEZ} = 30^\circ, \quad c_{WEZ} = \cos(\theta_{WEZ}), \quad R_{WEZ-step} = 4$$

Let $\mathbf{1}_{WEZ,i}^{\text{my}} = \begin{cases} 1, & \text{if agent } i \text{ is in WEZ against } j \\ 0, & \text{otherwise} \end{cases}$

Let $\mathbf{1}^{\text{op}}_{WEZ,i} = \begin{cases} 1, & \text{if opponent } j \text{ is in WEZ against } i \\ 0, & \text{otherwise} \end{cases}$ Then:

$$r_i^{WEZ}(t) + R_{WEZ-step}\mathbf{1}^{\text{my}}_{WEZ,i}(t) - R_{WEZ-step}\mathbf{1}^{\text{op}}_{WEZ,i}(t) \tag{5.46}$$

It is also important to award the agent if it keeps the opponents inside its WEZ and, eventually, takes the opponent down. Let $W_i(t)$ the number of steps that the opponent stays inside agent's WEZ, we update it as:

$$W_i(t) = \begin{cases} W_i(t-1) + 1, & \mathbf{1}^{\text{my}}_{WEZ,i}(t) = 1 \wedge \neg\text{destroyed}_j \\ \max(W_i(t-1) - 1, 0), & \text{otherwise} \end{cases}$$

If $W_i(t) \geq 5$ and the opponent is alive, we apply a "hit" to the opponent and reduce its Health Points (HP):

$$HP_j(t) = \max(HP_j(t) - \Delta HP, 0), \quad \Delta HP = 5$$

and grand:

$$r_i^{\text{hit}}(t) = R_{hit} = 30 \tag{5.47}$$

Then reset $W_i(t) = 0$. If this makes $HP_j(t) = 0$, we set $\text{destroyed}_j = \texttt{True}$ and add:

$$r_i^{\text{destroy}}(t) = R_{destroy} = 250 \tag{5.48}$$

So the event reward is:

$$r_i^{\text{kill}}(t) = \begin{cases} R_{hit}, & \text{hit but not destroyed} \\ R_{hit} + R_{destroy}, & \text{hit and destroyed} \\ 0, & \text{otherwise} \end{cases} \tag{5.49}$$

At the same time, we do apply damage and death penalties to the agents. Let previous and current HP of agent i be $HP_i(t-1)$, $HP_i(t)$. If HP_i decreases:

$$r_i^{\text{damage}}(t) = -R_{hit}\mathbf{1}\{HP_i(t-1) - HP_i(t) > 0\} \tag{5.50}$$

If destroyed:

$$r_i^{\text{death}}(t) = -R_{destroy}\mathbf{1}\{\text{destroyed}_i = \texttt{True}\} \tag{5.51}$$

Finally, a small time penalty is introduced to encourage faster resolution:

$$r_i^{\text{time}}(t) = -P_{time}, \quad P_{time} = 0.08 \tag{5.52}$$

Summing up all these terms we define the engagement reward (scaled by safety):

$$r_i^{\text{eng}} = r_i^{\text{base}}(t) + r_i^{\text{close}}(t) + r_i^{\text{WEZ}}(t) + r_i^{\text{kill}}(t) \tag{5.53}$$

Then the final per-step reward is:

$$r_i(t) = \lambda_i(t) \cdot r_i^{\text{eng}}(t) + r_i^{\text{alt}}(t) + r_i^{\text{dmg}}(t) + r_i^{\text{death}}(t) + r_i^{\text{far}}(t) + r_i^{\text{time}}(t) + r_i^{\text{inv}}(t) \tag{5.54}$$

5.6.5. Discussion

Dogfight constitutes the most challenging and informative benchmark in this thesis because it couples high-fidelity 6-DoF aircraft dynamics with an adversarial, non-stationary multi-agent learning setting. Unlike single-agent tracking tasks, progress is not monotonic: each agent continually adapts to the opponent, so improvements often manifest as shifting tactical patterns rather than a simple increase in reward. The comparison between a strict zero-sum reward and a non-zero-sum shaping highlights a key trade-off observed in competitive RL: zero-sum formulations enforce a clean notion of relative advantage and discourage mutual farming, but can be harder to optimize and more prone to cycling equilibria; conversely, the shaped baseline reward can stabilize early learning by providing richer dense feedback, at the cost of potential bias toward proxy objectives. Finally, the fully observable versus partially observable settings isolate the role of information in credit assignment: explicit access to relative geometry (LOS, ATA, AA, HCA, distance) makes rewardstate causality clearer and accelerates the emergence of meaningful WVR behaviors, whereas partial observability forces the policy to infer latent opponent state from motion history, amplifying reward noise and increasing the likelihood of conservative or degenerate strategies.

6 | Results & Discussion

6.1. Toy Model Results

6.1.1. Experimental Protocol & Implementation Details

Training

Training for all Toy Model environments was performed with episodic rollouts and policy optimization using Adam; in multi-agent scenario, separate optimizers for the actor networks and a centralized critic were used. To keep full reproducibility, a single random seed was fixed across all runs, rather than averaging over multiple seeds. This choice ensures that observed performance changes are attributable to the environment dynamics and training configuration, without variability induced by stochastic initialization. It is worth noting that most of the training has been performed only on PPO as it was the algorithms showing better results overall.

Table 6.1 shows the total number of environment steps used to train each environment.

	FollowPoint	**FollowTrajectory**	**Tag Environment**	**Fight Environment**
PPO	$2.5 \cdot 10^5$	$1 \cdot 10^6$	$2.5 \cdot 10^6$	$2.5 \cdot 10^6$
SAC	$2.5 \cdot 10^5$	$1 \cdot 10^6$	-	-
TD3	$2.5 \cdot 10^5$	$1 \cdot 10^6$	-	-

Table 6.1: Total training steps per algorithm and environment (Toy Model).

Evaluation

For the single-agent environments, evaluation was performed by rolling out the trained, frozen PPO policy for a fixed number of test episodes, using deterministic action selection to remove exploration noise and make outcomes directly comparable. On multi-agent environments, evaluation was carried out by running the trained (frozen) policies for a fixed number of test episodes, once again using deterministic action selection. Evaluation

was kept fully reproducible by using a fixed random seed, rather than averaging across multiple seeds; therefore, results are reported from the same deterministic evaluation protocol and the same environment initialization conditions for all models.

Hyperparameters

Hyperparameters for PPO algorithm are shown in Table 6.2; as previously mentioned, PPO was the most used option, therefore its hyperparameters are the only ones reported. Actor and Critic networks are implemented as Fully Connected MultiLayer Perceptrons (FCMLPs), with their activation functions and layer sizes specified in the tables. Exploration schedules, gradient clipping, and learning rate decay follow standard conventions for continuous control.

Parameter	Value
batch_size	2048
actor_lr	0.00005
critic_lr	0.00005
lr_decay	Linear
gamma	0.99
gae_lambda	0.95
ppo_epoch	20
clip_coef	0.2
max_grad_norm	0.5
actor_arch	[128, 128], Tanh
critic_arch	[128, 128], Tanh
optimizer	Adam

Table 6.2: PPO Hyperparameters for Toy Model

Visualization & Rendering

Visualization of learning curves and evaluation metrics was performed using `matplotlib` (Python), directly from the training and evaluation logs. In addition, animated GIFs were generated with `matplotlib` to provide a qualitative inspection of the learned behavior by visualizing the agent trajectory (and, when present in the environment, the target motion) over complete test episodes.

6.1.2. FollowPoint

Fixed Point Training Reward

Fig. 6.1 shows the training curves for PPO, SAC and TD3, where each curve represents the step reward obtained by the agent. Both raw and smoothed curves are displayed, with smoothed ones using a moving average window of 25 logs.

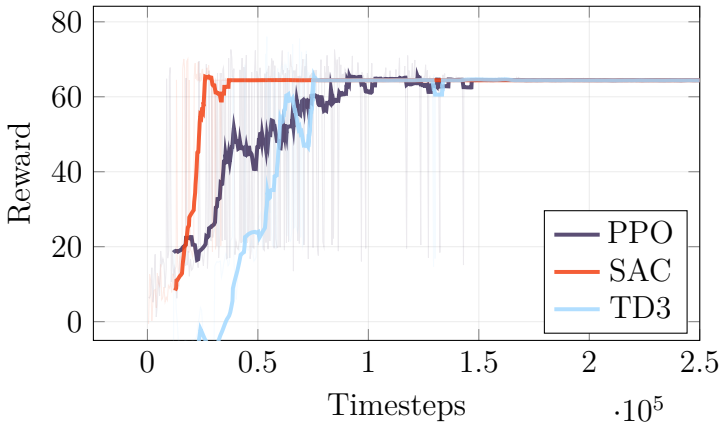
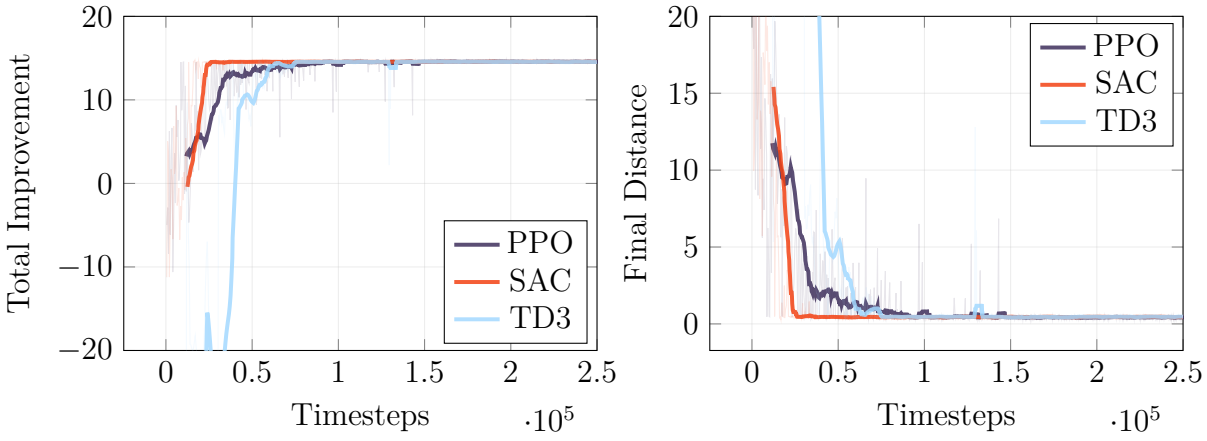


Figure 6.1: Toy Model FixedFollowPoint: training reward for PPO, SAC, and TD3.

Fixed Point Additional Training Metrics

For completeness, some metrics were also tracked during training, to better understand agent's performance. Fig. 6.2 shows the two main additional metrics used: the total improvement over the episode (calculated as $d_{\text{initial}} - d_{\text{final}}$) and the final distance reached at the end of the episode.



(a) Total Improvement

(b) Final Distance from Target

Figure 6.2: Toy Model FixedFollowPoint: total improvement and final distance, comparing PPO, SAC, and TD3.

Fixed Point Evaluation Metrics

Evaluation scores, computed over 10 independent runs per algorithm, are summarized in Table 6.3 for FixedFollowPoint. Results are reported as mean ± standard deviation.

Algorithm	Avg Reward	Steps/episode ↓	Avg distance ↓	Success ratio [%] ↑
PPO	64.40 ± 0.00	146.00 ± 0.00	0.457 ± 0.000	100.0 ± 0.0
SAC	64.44 ± 0.03	138.00 ± 0.00	0.425 ± 0.030	100.0 ± 0.0
TD3	64.44 ± 0.01	141.00 ± 0.00	0.455 ± 0.026	100.0 ± 0.0

Values are reported as mean and standard deviation over N=10 runs. Arrows indicate preferred direction of each metric.

Table 6.3: Toy Model FixedFollowPoint: evaluation scores.

Random Point Training Reward

Similarly to Fixed Point, Fig. 6.3 shows the training curves for PPO, SAC and TD3, where each curve represents the step reward obtained by the agent. Both raw and smoothed curves are displayed, with smoothed ones using a moving average window of 25 logs.

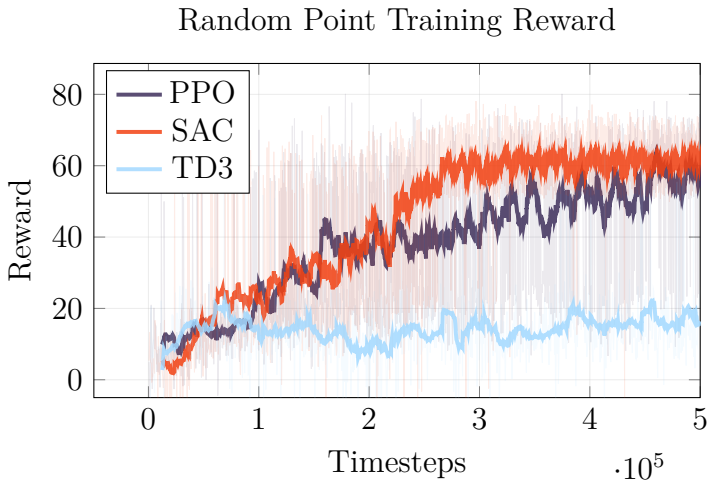
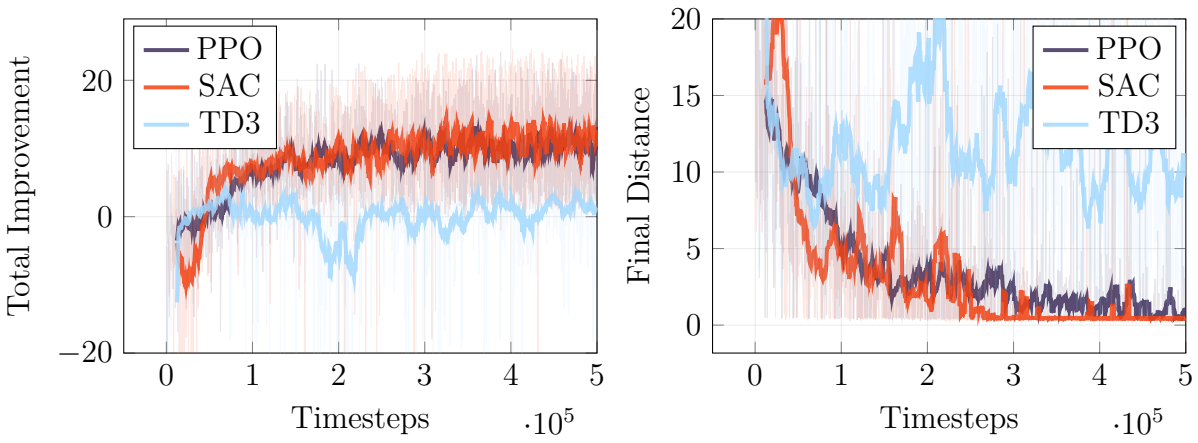


Figure 6.3: Toy Model RandomFollowPoint: training reward for PPO, SAC, and TD3.

Random Point Additional Training Metrics

Again, the same metrics were tracked during training, to better understand agent's performance. Fig. 6.4 shows the two main additional metrics used: the total improvement over the episode (calculated as $d_{\text{initial}} - d_{\text{final}}$) and the final distance reached at the end of the episode.



(a) Total Improvement

(b) Final Distance from Target

Figure 6.4: Toy Model RandomFollowPoint: total improvement and final distance, comparing PPO, SAC, and TD3.

Random Point Evaluation Metrics

Evaluation scores, computed over 10 independent runs per algorithm, are summarized in Table 6.4 for RandomFollowPoint. Results are reported as mean ± standard deviation.

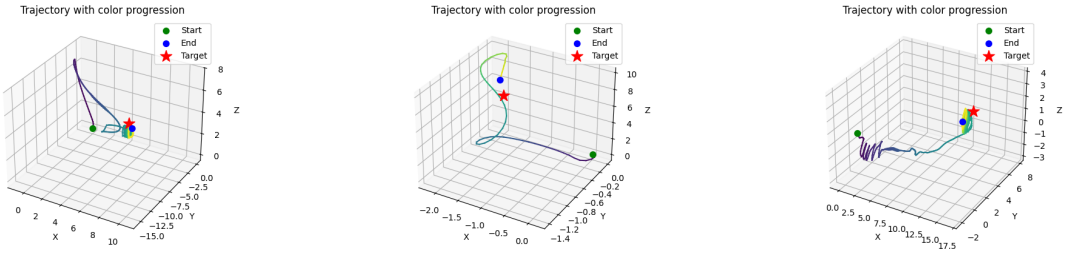
Algorithm	Avg Reward	Steps/episode ↓	Avg distance ↓	Success ratio [%] ↑
PPO	46.57 ± 16.34	139.30 ± 192.59	0.563 ± 0.259	80.0 ± 42.2
SAC	59.87 ± 6.10	96.20 ± 65.52	0.455 ± 0.029	100.0 ± 0.0
TD3	16.89 ± 17.09	479.80 ± 63.88	7.391 ± 8.352	10.0 ± 31.6

Values are reported as mean and standard deviation over N=10 runs. Arrows indicate preferred direction of each metric.

Table 6.4: Toy Model RandomFollowPoint: evaluation scores.

Evaluation Renders

Since FixedFollowPoint is less relevant and too trivial of a task, only the graphical renders of RandomFollowPoint are presented. Agent behavior was represented by generating plots indicating agent's start and end position, target position and the followed trajectory. Fig. 6.5 represents significative episodes for PPO, TD3 and SAC.



(a) PPO (b) SAC (c) TD3

Figure 6.5: Toy Model RandomFollowPoint: evaluation episode comparison across PPO, SAC, and TD3.

Discussion

The FollowPoint task was analyzed under two distinct settings: FixedFollowPoint, where the target remains at a fixed location, and RandomFollowPoint, where the target position changes across episodes, requiring broader generalization. In the FixedFollowPoint variant (Fig. 6.1-6.2), all three algorithms successfully learned stable strategies to reach and track the target. The training reward curves show a rapid increase during early learning, followed by a clear plateau at high reward values, indicating convergence to an effective control policy. Among the three, SAC displays the fastest rise and the most regular stabilization, reaching the plateau earlier and with fewer visible oscillations in the raw reward signal. PPO follows closely, achieving a similarly high asymptotic reward but generally requiring slightly more timesteps to fully stabilize. TD3 also converges to competitive reward levels, yet its raw curve exhibits larger intermittent drops and higher variability during training, suggesting a greater tendency to transiently deviate from the optimal tracking behavior. The additional training metrics reinforce this interpretation. In Fig 6.2a, the total improvement rapidly becomes positive and saturates near a consistent maximum for all methods, confirming that the agent systematically reduces the distance to the target over each episode. In Fig 6.2b, the final distance sharply decreases toward values close to zero, demonstrating reliable target acquisition. Here again, SAC and PPO show the most consistent reduction and the cleanest plateau, whereas TD3 occasionally produces spikes in final distance, consistent with short-lived instabilities or looping behaviors even after apparent convergence. Overall, the FixedFollowPoint results indicate that when the objective is highly structured and stationary, all methods can learn strong tracking policies, with SAC providing the most stable learning dynamics and TD3 being more sensitive to training noise and local instabilities.

The RandomFollowPoint variant (Fig. 6.3-6.5) is noticeably more challenging, as the agent

must cope with changing target locations and cannot rely on a single memorized approach trajectory. This increased complexity is reflected in the training curves: learning is slower, and the separation between algorithms becomes more pronounced. SAC achieves the best overall performance, showing a clear upward reward trend and reaching the highest plateau among the three. PPO improves steadily as well, but stabilizes at a lower reward level compared to SAC, suggesting competent yet less optimal tracking under target randomization. In contrast, TD3 struggles substantially: its reward remains much lower throughout training and improves only marginally, indicating difficulty in consistently solving the randomized tracking objective. The additional training metrics highlight the same pattern. In Fig. 6.4a, total improvement increases for SAC and PPO, confirming that both methods learn to reduce target distance over the episode even under varying initial conditions. TD3, however, stays around low or highly fluctuating improvement values, suggesting frequent failures to translate actions into consistent progress. Fig. 6.4b further supports this: SAC and PPO reduce the final distance substantially compared to early training, while TD3 maintains higher and far more erratic final distances, pointing to unstable tracking and occasional divergence from effective pursuit trajectories.

The three renders shown in Fig. 6.5 provide an intuitive explanation of the quantitative gap observed in both training and evaluation. SAC exhibits a smooth, coherent approach: the agent follows a largely monotonic path toward the target with limited oscillations and only minor corrections close to the goal, resulting in a clean terminal alignment. This behavior is consistent with SAC achieving the lowest residual distance and the highest success rate in RandomFollowPoint evaluation. PPO also reaches the target region reliably, but the trajectory often includes a sharper initial turn and small corrective loops near the target, suggesting a slightly less direct path and more residual control effort in the final phase. This aligns with PPO's good but more variable performance, where most runs succeed yet with a wider dispersion in efficiency and accuracy. In contrast, TD3 frequently displays noticeable oscillations early in the episode and a less coordinated convergence pattern: the path can include pronounced waviness and prolonged adjustments before reaching the vicinity of the target (when it succeeds). Such motion is a qualitative hallmark of inefficient regulation and can easily translate into longer episodes and occasional failure to settle, matching TD3's high steps-per-episode and low success ratio under randomized conditions.

Overall, the FollowPoint results highlight a strong regime-dependent separation. In FixedFollowPoint, all methods converge because the objective is stationary and the policy can exploit a consistent geometry; differences mainly reflect learning stability and transient oscillations. In RandomFollowPoint, however, the task demands robust generalization

across target placements and therefore exposes algorithmic limitations. The evaluation renders reinforce the metric-based conclusion: SAC not only achieves the best aggregate scores, but does so through consistently smooth and goal-directed trajectories, indicating stable closed-loop behavior across diverse initial conditions. PPO remains a solid baseline with frequent success, yet its trajectories suggest slightly less optimal terminal regulation and higher run-to-run variability. TD3 appears most sensitive to the increased stochasticity, with oscillatory and inefficient paths that explain both its degraded accuracy and its significantly reduced reliability.

6.1.3. FollowTrajectory

Training Reward

Fig. 6.6 shows the training curves for PPO, SAC and TD3, where each curve represents the step reward obtained by the agent. Both raw and smoothed curves are displayed, with smoothed ones using a moving average window of 25 logs.

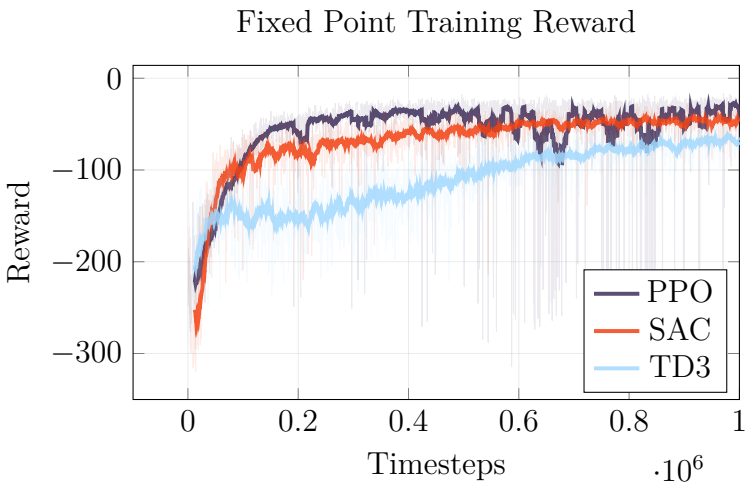
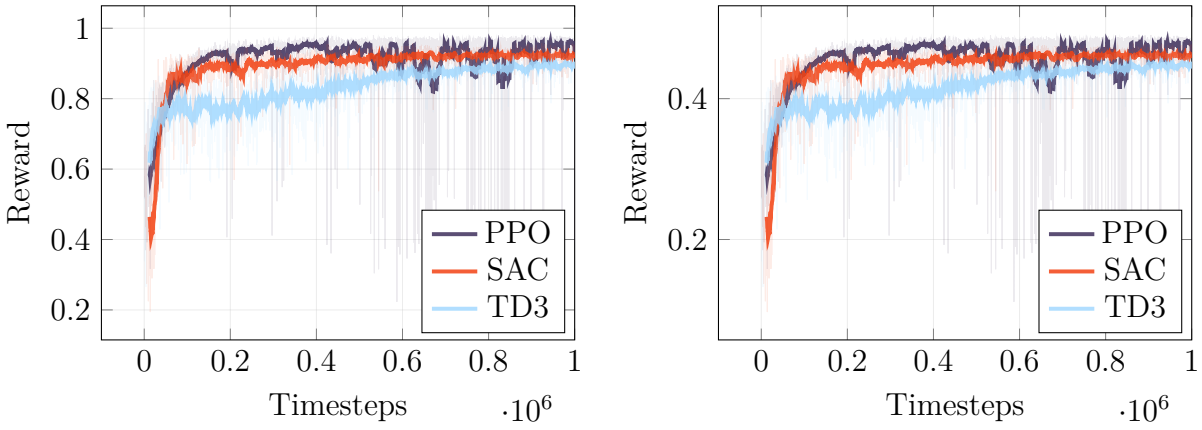


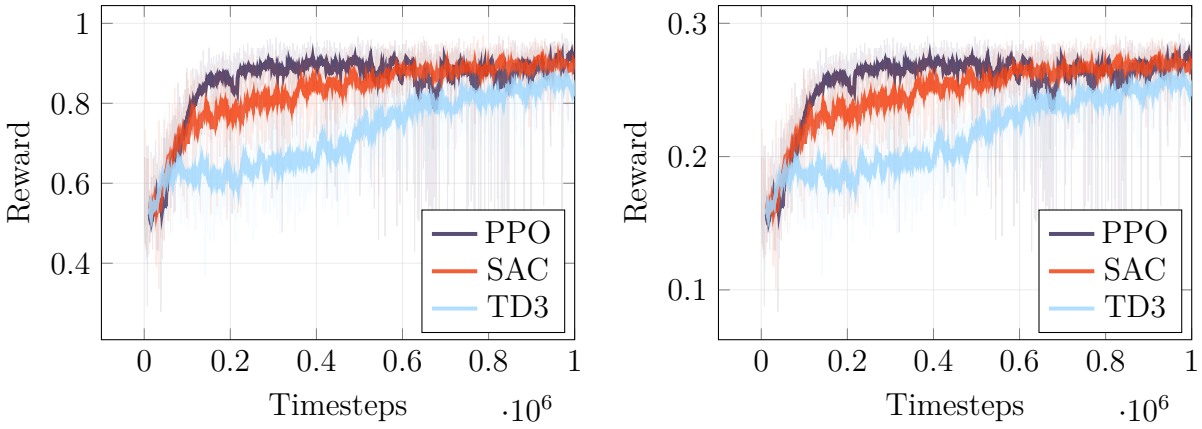
Figure 6.6: Toy Model FollowTrajectory: training reward for PPO, SAC, and TD3.

Furthermore, it is possible to see how the different components of the reward change during training in Fig. 6.7 to Fig. 6.10.



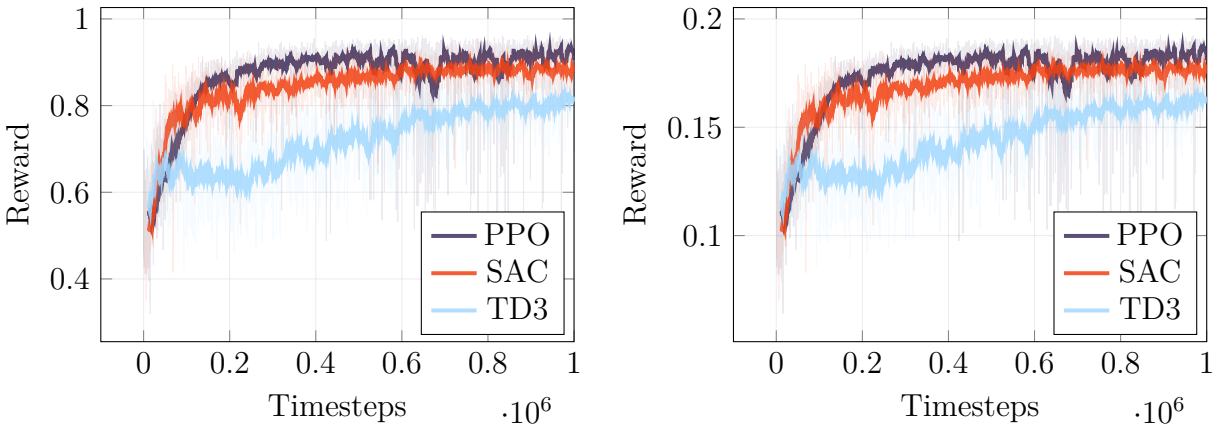
(a) Raw distance reward (b) Weighted distance reward

Figure 6.7: Toy Model FollowTrajectory: distance reward components, raw and weighted.



(a) Raw heading alignment reward (b) Weighted heading alignment reward

Figure 6.8: Toy Model FollowTrajectory: heading alignment reward components, raw and weighted.



(a) Raw spatial alignment reward (b) Weighted spatial alignment reward

Figure 6.9: Toy Model FollowTrajectory: spatial alignment reward components, raw and weighted.

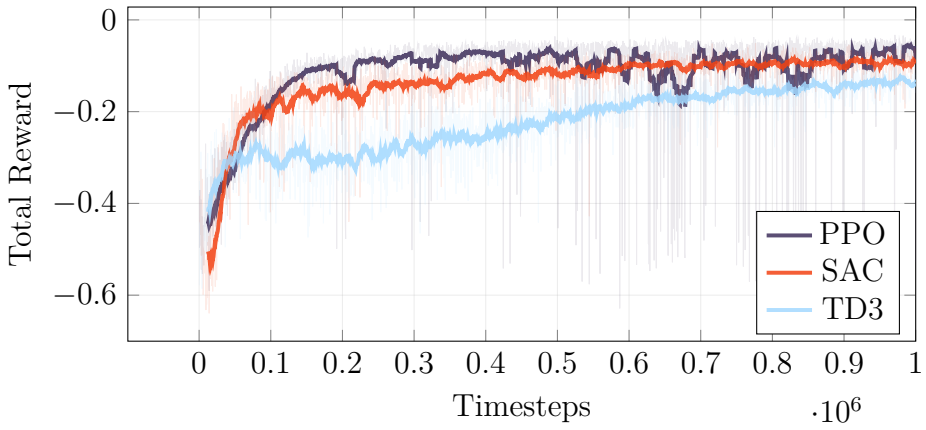
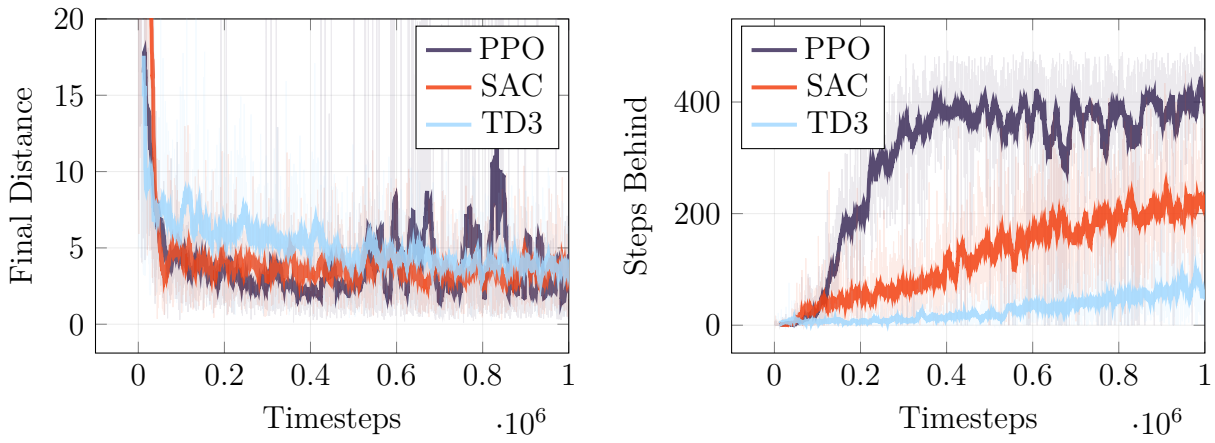


Figure 6.10: Toy Model FollowTrajectory: total reward.

Additional Training Metrics

To better understand training performance, some other metrics were tracked: Fig. 6.11a shows the distance between agent and target at the end of the episode, while Fig. 6.11b shows for how long (how many steps) the agent has been behind the target.



(a) Final distance from target

(b) Steps behing taret

Figure 6.11: Toy Model FollowTrajectory: final distance and totally behind.

Evaluation Metrics

Evaluation scores, computed over 100 independent runs per algorithm, are summarized in Table 6.5. Results are reported as mean ± standard deviation.

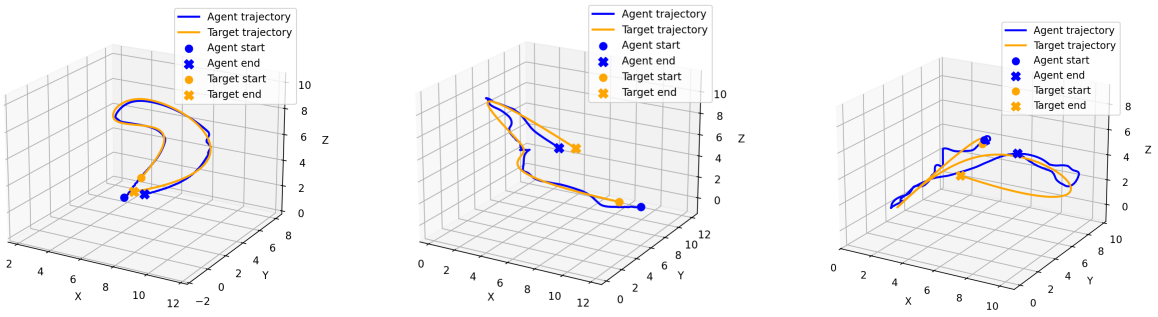
Algorithm	Avg Reward	Final Distance ↓	Totally Behind ↑
PPO	-32.64 ± 28.90	2.378 ± 4.474	419.03 ± 68.78
SAC	-65.36 ± 10.80	3.851 ± 1.859	76.09 ± 41.65
TD3	-66.20 ± 12.32	3.726 ± 1.926	82.23 ± 46.89

Values are reported as mean and standard deviation over N=100 runs. Arrows indicate preferred direction of each metric.

Table 6.5: Toy Model FollowTrajectory: evaluation scores.

Evaluation Renders

Here some plots are presented to better show performance and behavior. Agent behavior was represented by generating plots indicating agent's start and end position, target start and end position and the followed trajectory. Fig. 6.12 represents significative episodes for PPO, TD3 and SAC.



(a) PPO (b) SAC (c) TD3

Figure 6.12: Toy Model FollowTrajectory: evaluation episode comparison across PPO, SAC, and TD3.

Discussion

The FollowTrajectory task is designed to train a continuous-control RL agent to track a moving target whose trajectory is randomized across episodes. Unlike a fixed-point tracking problem, success here requires persistent pursuit under changing geometry: the agent must continuously reduce relative position error while also aligning its motion direction with the instantaneous target path. Across training, all three algorithms show a clear improvement in step reward, with a rapid early phase where performance increases sharply and then a slower saturation phase. The smoothed curves highlight that PPO reaches the highest plateau reward and stabilizes earlier, while SAC converges slightly below PPO, and TD3 improves more slowly and remains consistently behind the other two. The raw curves show substantial variance (especially for PPO), which is consistent with occasional unstable episodes or brief tracking losses even after the policy has largely converged. In this environment, such spikes are expected because the target trajectory changes and a policy can still fail on specific path geometries (sharp turns, high curvature, or unfavorable initial relative states). A key qualitative takeaway is that PPO appears to learn good-enough tracking quickly, while SAC learns more gradually but tends to be smoother, and TD3 struggles the most to reach the same level of consistency. This ordering is reflected not only in the total reward trend, but also in the evolution of each reward component.

The reward decomposition is useful because it shows what the agent learns first and which sub-objectives remain difficult:

- **Distance components**: The raw distance reward rapidly rises toward a high value for all methods, indicating that closing the gap is the easiest and most dominant learning signal early in training. PPO reaches the best distance-related plateau,

SAC follows closely, and TD3 lags, suggesting that TD3 has more difficulty consistently reducing positional error under target motion. The weighted distance term follows the same ordering but with a compressed range, which is desirable: it keeps this component from dominating the total reward scale once the agent becomes close enough, leaving room for orientation/trajectory-alignment terms to shape behavior.

- **Heading alignment component**: The heading alignment curves show a similar hierarchy, but with a more visible gap between TD3 and the other two, especially during the mid-training phase. This typically indicates that TD3 can reduce distance by chasing but fails to orient efficiently along the target direction, which often produces oscillations or lag when the trajectory curvature changes. The weighted version again compresses magnitude and makes this term contribute proportionally without overpowering the objective.
- **Spatial alignment component**: Spatial alignment is effectively a proxy for staying on the correct side of the trajectory rather than simply minimizing Euclidean distance. Here the separation between algorithms becomes meaningful: PPO and SAC achieve high spatial alignment earlier, while TD3 shows a longer transient where it sits at a lower plateau before catching up. This is consistent with a policy that can get near the target but does not generalize as well across different path segments (e.g., overshooting corners, cutting inside turns, or taking inefficient approaches).

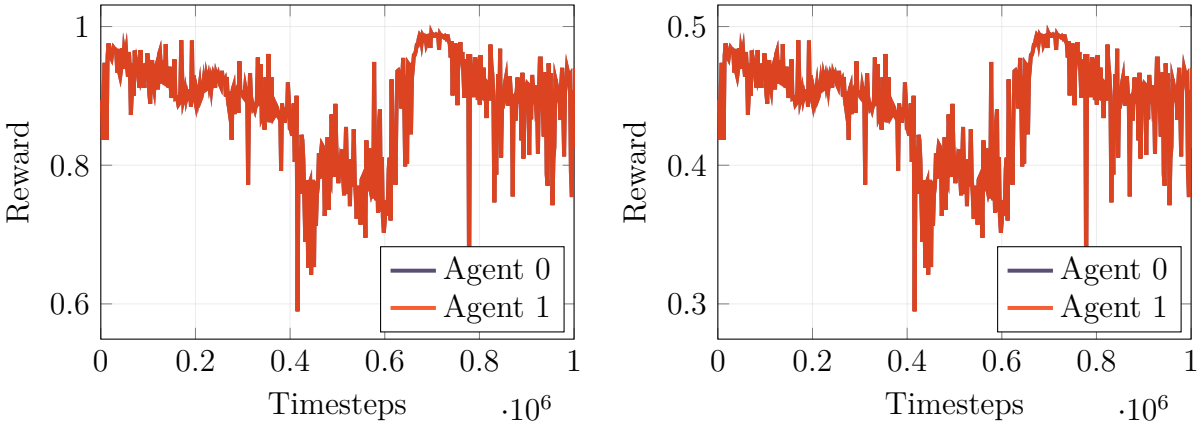
This decomposition indicates that the reward shaping provides dense intermediate signals that rise quickly and guide learning early. The weighted counterparts appear especially important, because they prevent any single component (typically distance) from dominating the scale once basic tracking is learned. This is critical for a random trajectory environment: if distance dominates too strongly, the agent can learn a local chasing behavior that is fragile under curvature changes; if alignment dominates too strongly, the agent may align direction without actually closing the gap efficiently.

Overall, FollowTrajectory demonstrates that the multi-term reward successfully drives learning under trajectory variability, and it highlights how different RL algorithms exploit the shaping differently: PPO excels at stable tracking; SAC and TD3 are less precise in distance minimization.

6.1.4. Tag Environment

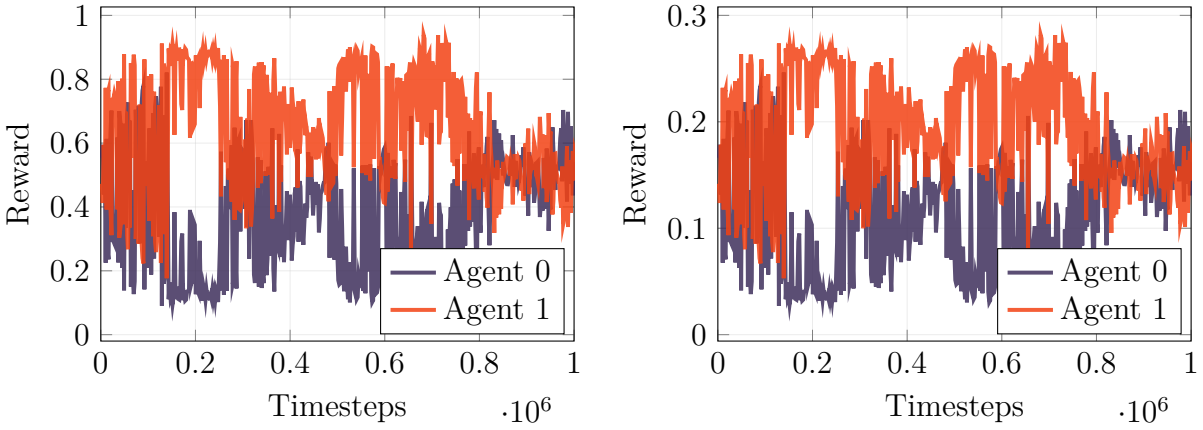
Training Reward

Reward components and total reward for this environment are shown in Fig. 6.13 to 6.16.



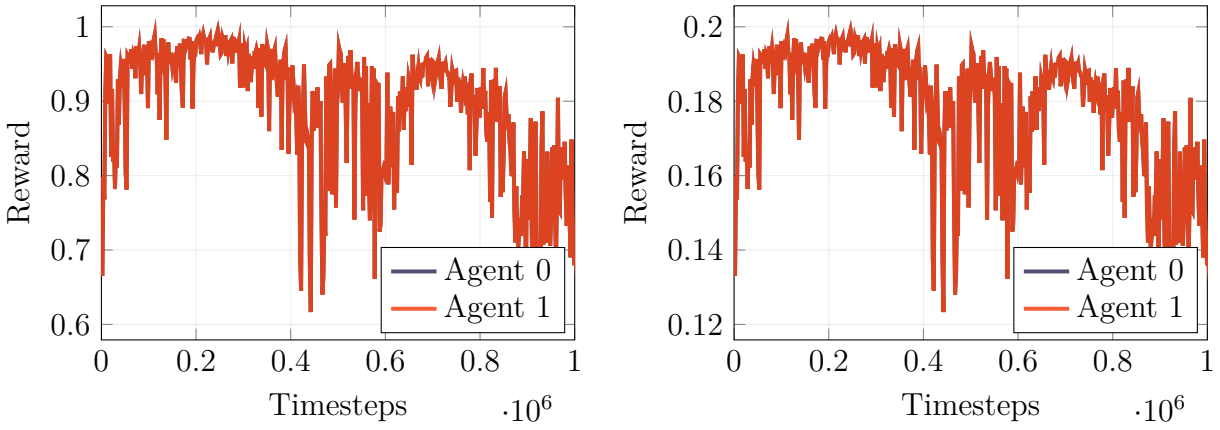
(a) Distance reward (raw) (b) Distance reward (weighted)

Figure 6.13: Toy Model Tag Environment: distance reward, raw vs weighted. Agent 0 and Agent 1 perform exactly the same.



(a) Heading position reward (raw) (b) Heading position reward (weighted)

Figure 6.14: Toy Model Tag Environment: heading position reward, raw vs weighted.



(a) Heading velocity reward (raw) (b) Heading velocity reward (weighted)

Figure 6.15: Toy Model Tag Environment: heading velocity reward, raw vs weighted. Agent 0 and Agent 1 perform exactly the same.

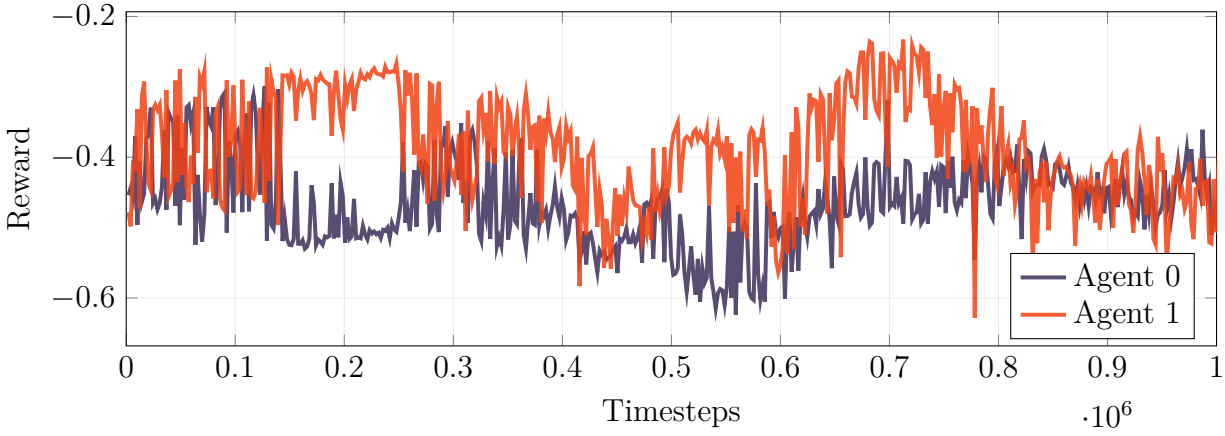
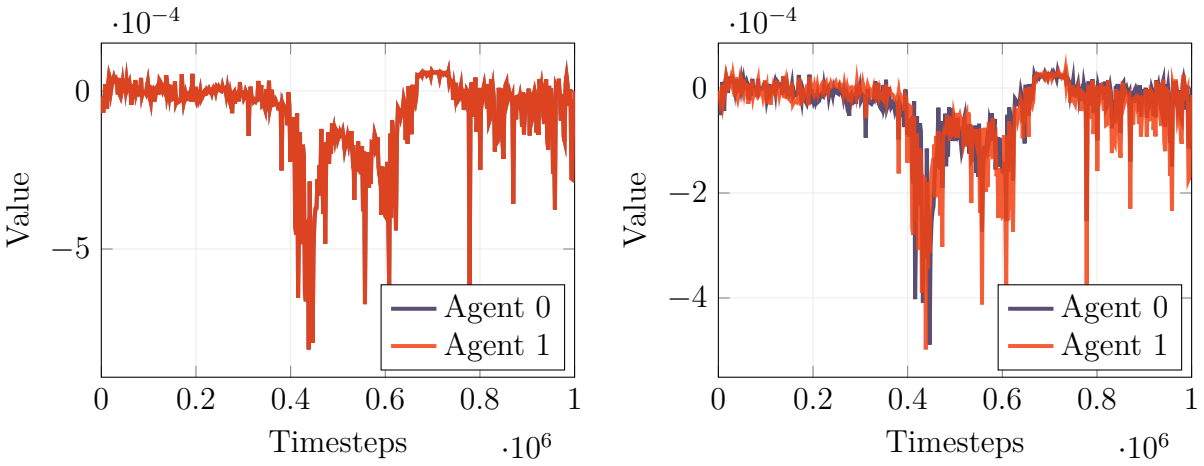


Figure 6.16: Toy Model Tag Environment: total reward.

Additional Training Metrics

The following metrics, shown in Fig. 6.17 to Fig. 6.21, were tracked to better understand and interpret agent performance.



(a) Delta term (raw)

(b) Delta term (weighted)

Figure 6.17: Toy Model Tag Environment: delta term, raw vs weighted.

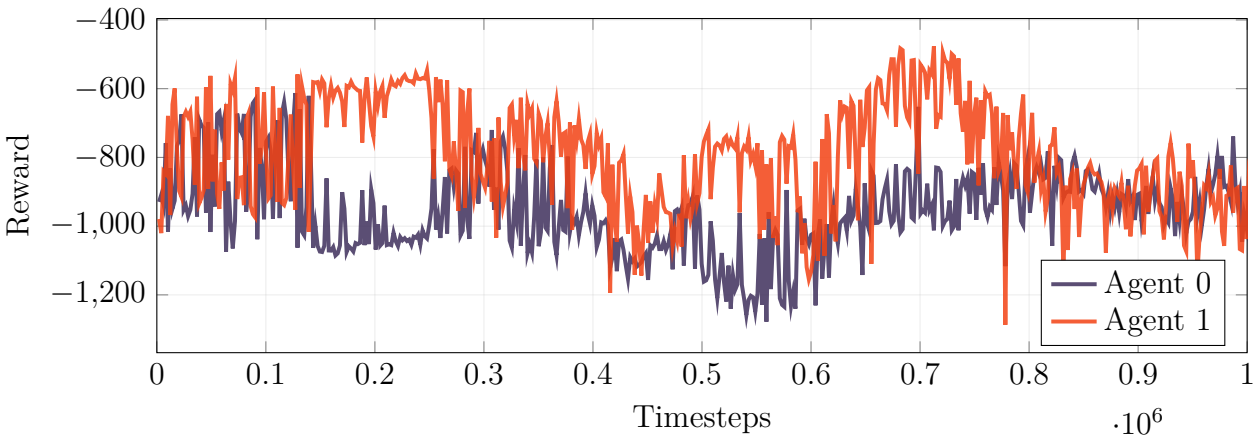
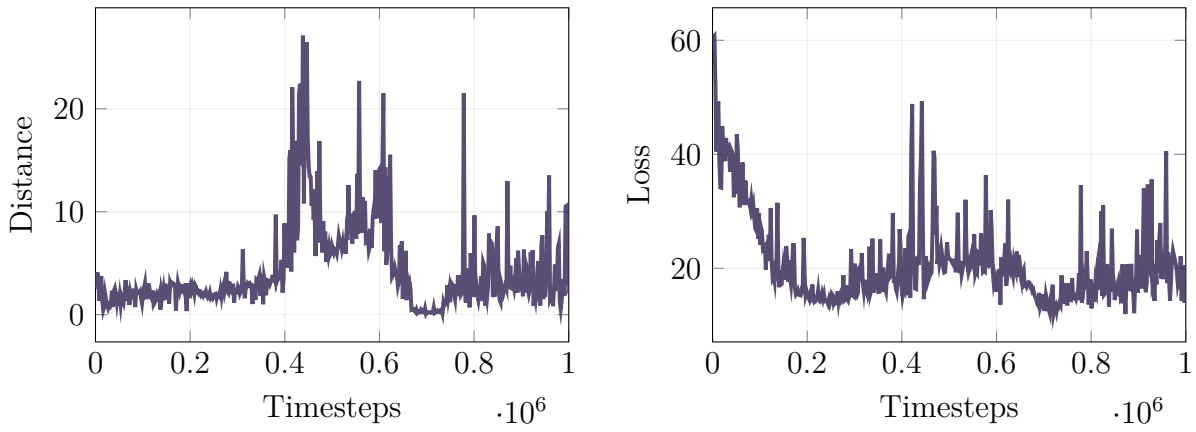
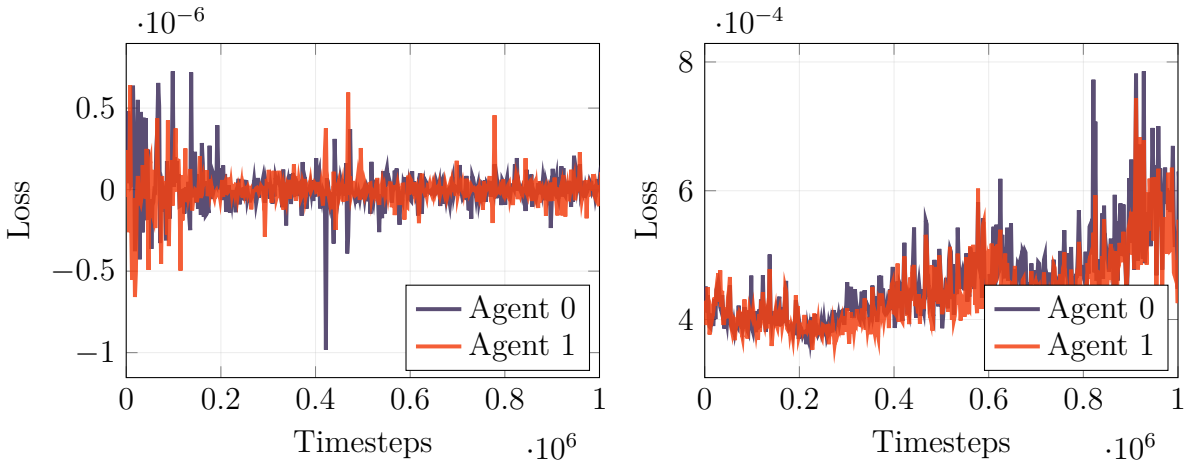


Figure 6.18: Toy Model Tag Environment: episode return.



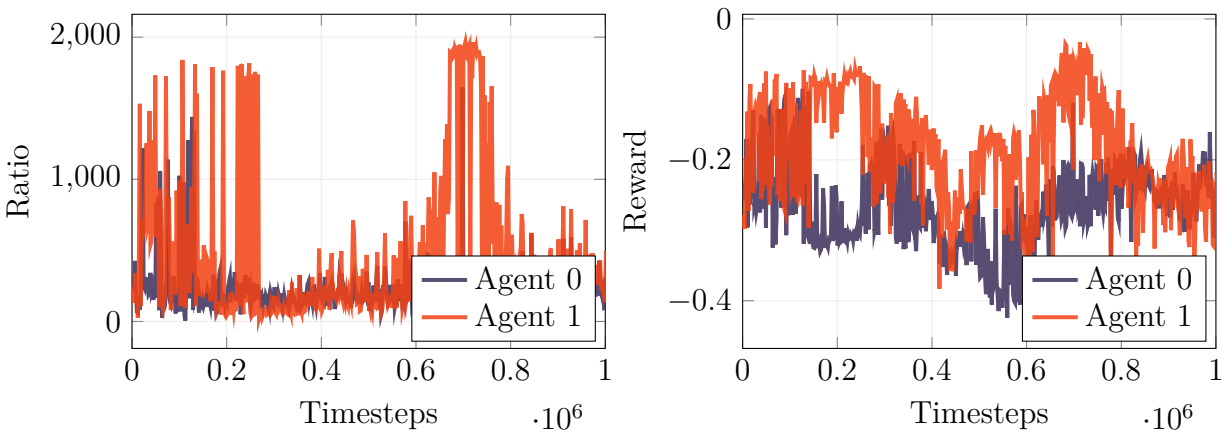
(a) Final distance (b) Critic loss

Figure 6.19: Toy Model Tag Environment: final distance and critic loss.



(a) Actor policy loss (b) Actor smooth loss

Figure 6.20: Toy Model Tag Environment: actor policy loss and actor smooth loss.



(a) Followed ratio

(b) Base reward

Figure 6.21: Toy Model Tag Environment: followed ratio and base reward.

Evaluation Metrics

Evaluation scores, computed over 100 independent runs, are summarized in Table 6.6. Results are reported as mean ± standard deviation. Values for both agents are reported, shown as (0) and (1).

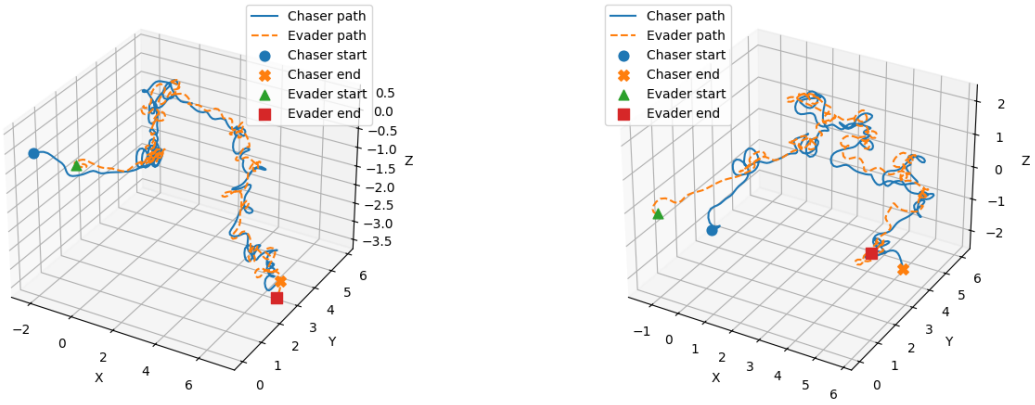
Algorithm	Reward (0) ↑	Reward (1) ↑	Totally Behind (0) ↑	Totally Behind (1) ↑
PPO	-226.353 ± 6.306	-199.600 ± 6.101	175.100 ± 26.657	278.460 ± 33.277

Values are reported as mean and standard deviation over N=100 runs. Arrows indicate preferred direction of each metric.

Table 6.6: Toy Model Tag Environment: evaluation scores.

Evaluation Renders

The 3D plots of a couple of episodes are shown in Fig. 6.22. Agent behavior was represented by generating plots indicating both agents start and end positions.



(a) Evaluation episode 1

(b) Evaluation episode 2

Figure 6.22: Toy Model Tag Environment: evaluation episodes with PPO.

Discussion

The Tag environment results show that learning is largely driven by a consistent shaping structure where distance reduction and heading alignment provide the dominant signal, while secondary terms mainly stabilize the solution rather than changing the policy qualitatively. During training, the distance reward remains high and relatively stable after the initial transient, indicating that both agents quickly acquire a basic close-the-gap behavior and maintain it over most of the horizon. The heading position and heading velocity components then refine this behavior by discouraging lateral drifting and oscillatory corrections: the weighted variants are smoother and compress the dynamic range, which helps prevent single components from dominating the total return and yields a more interpretable contribution profile. The delta term stays close to zero on average with occasional negative spikes, suggesting it acts primarily as a corrective penalty when the agents momentarily violate a constraint or perform abrupt deviations, rather than providing a continuous incentive. This interpretation is coherent with the additional metrics: critic loss remains bounded and trends downward early on, while actor policy loss fluctuates around a small scale, consistent with PPO updates that become more conservative once a viable tracking strategy is found. At the same time, the action smoothing loss increases later in training, indicating that as policies improve, the optimizer still trades off performance against smoothness. The episode return remains negative and exhibits regime changes, which is not contradictory: a large constant penalty or time-based cost can shift the baseline below zero, so improvements are better interpreted through relative trends and complementary metrics such as final distance and followed ratio. Notably, evaluation confirms a systematic asymmetry between the two agents: Agent 1 achieves a less negative average return than Agent 0 and also exhibits a higher Totally Behind score.

The rendered episodes in Fig. 6.22 qualitatively support these findings: trajectories show sustained pursuit with coherent start/end displacement for both agents, but with visible variability across runs.

Overall, the environment produces consistent tracking-oriented behaviors with stable optimization, yet the role imbalance visible in both training curves and evaluation statistics suggests that further reward normalization or symmetric term re-weighting could improve fairness and reduce variance, especially if the goal is to compare agents under equivalent difficulty rather than maximize performance for one role.

6.1.5. Fight Environment

Training Reward

The Fight environment can be regarded as a direct extension of Tag: it preserves the same dense shaping terms defined in (4.12) and augments them with a set of sparse shooting-related bonuses, yielding the final reward in (4.18). Consequently, both the overall training-reward dynamics and the relative contribution of the underlying base components remain directly comparable to the Tag case. The reward-component trends observed in this subsection are therefore consistent with, and effectively equivalent to, those already reported in Figures Fig. 6.13–Fig. 6.21. Any deviations are mainly due to the intermittent activation of the additional sparse terms rather than to changes in the dense shaping structure. For this reason, in the following we focus exclusively on the metrics introduced by the shooting mechanism. For completeness, we report in Fig. 6.23 the total episode return for both agents throughout training.

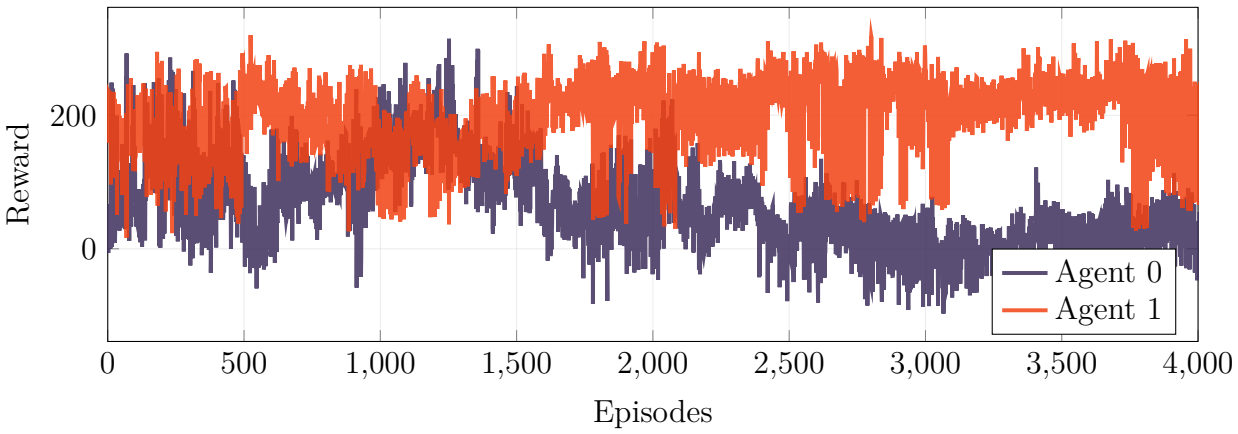


Figure 6.23: Toy Model Fight Environment: total episode return.

Additional Training Metrics

Fig. 6.24 reports a relevant metric in the dogfight context. This shows how many shots are hit and how often from each agent.

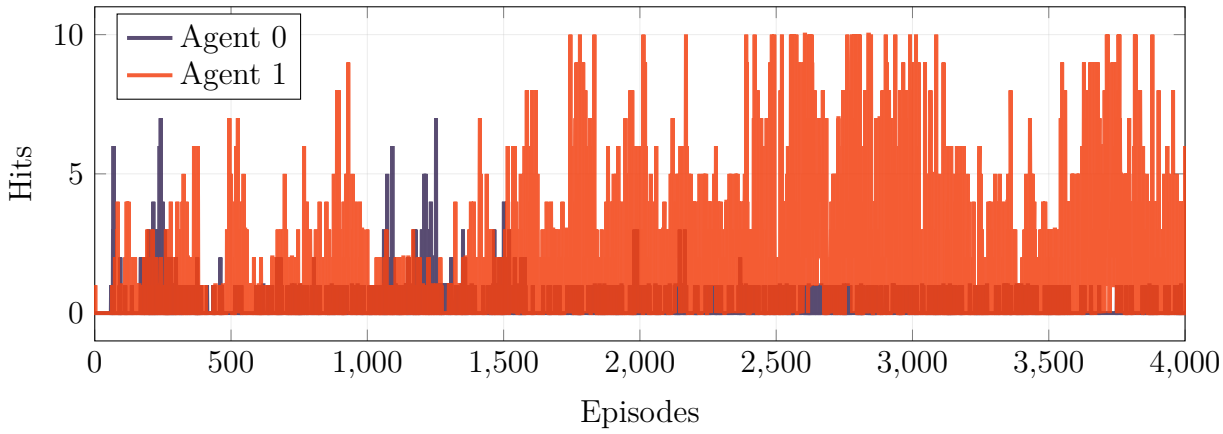


Figure 6.24: Toy Model Fight Environment: total episode hits.

Evaluation Metrics

Evaluation scores, computed over 100 independent runs, are summarized in Table 6.7. Results are reported as mean ± standard deviation. Values for both agents are reported, shown as (0) and (1).

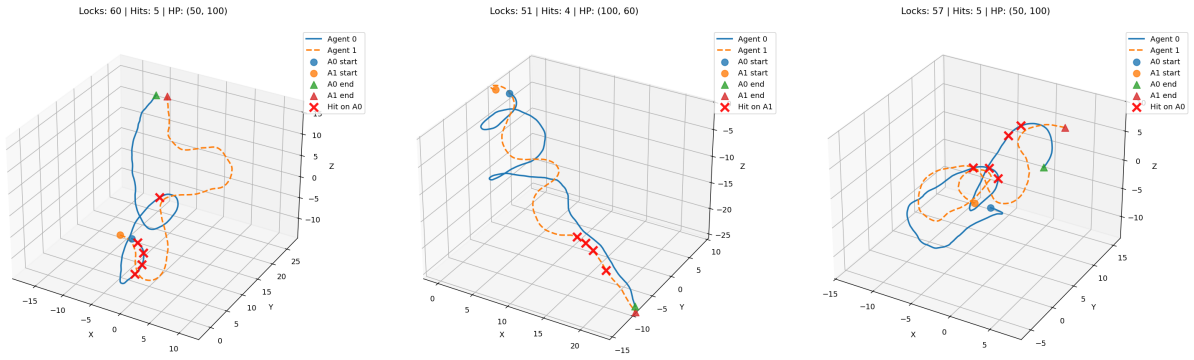
Algorithm	Reward (0) ↑	Reward (1) ↑	Shots Landed (0) ↑	Shots Landed (1) ↑
PPO	66.16 ± 64.49	207.48 ± 51.90	0.18 ± 0.51	4.10 ± 1.85

Values are reported as mean and standard deviation over N=100 runs. Arrows indicate preferred direction of each metric.

Table 6.7: Toy Model Fight Environment: evaluation scores.

Evaluation Renders

In this section some evaluation episodes' plots are shown to have a visual representation of agent performance and behavior. Agent behavior was represented by generating plots indicating both agents start and end positions, their trajectories and any hit occurred during the episode. Fig. 6.25 shows three significative evaluation episodes.



(a) Fight Episode 1 (b) Fight Episode 2 (c) Fight Episode 3

Figure 6.25: Toy Model Fight Environment: evaluation episodes showing two agents involved in WVR dogfight.

Discussion

Overall, the Fight environment behaves as a natural extension of Tag: it keeps the same dense shaping structure and adds sparse shooting bonuses, so the training reward trends remain comparable to the Tag case, with most differences coming from when the shooting terms activate. This is visible in the episode return, where Agent 1 consistently achieves higher returns while Agent 0 gradually drops, suggesting that Agent 1 learns to exploit the shooting mechanism more reliably and turns good positioning into effective engagements. The number of hits confirms this: hits increase mainly for Agent 1, while Agent 0 remains close to zero for most of training, indicating a clear imbalance in combat effectiveness. The evaluation supports the same conclusion: Agent 1 reaches a higher average reward and lands far more shots, whereas Agent 0 shows lower reward and very few successful hits. The relatively large standard deviation, especially for Agent 0, suggests that outcomes can vary a lot depending on initial conditions and early maneuver choices, which is expected in a close-range dogfight where small geometry differences can quickly change who gets the advantage. Finally, the evaluation renders provide an intuitive picture of these results: trajectories show that agents can approach, turn, and separate in realistic patterns, but only one side tends to convert these interactions into consistent shot opportunities. Taken together, these results indicate that the added sparse shooting rewards successfully encourage combat behavior, but they also highlight the need for better balance so that both agents can learn to create and capitalize on firing windows with similar consistency.

6.2. JSBSim Model Results

6.2.1. Experimental Protocol & Implementation

Training

Training for all JSBSim-based environments was carried out using episodic rollouts and policy optimization with the Adam optimizer. Given the nature of the flight-combat tasks and the long-horizon dependencies typical of pursuit and engagement dynamics, the work focused exclusively on PPO and its recurrent variant (rPPO), with the latter serving as the primary training backbone throughout most experiments. The only exception is for SAC, used on the LevelFlight environment as its performance were promising. This choice was motivated by the consistently stronger empirical performance of rPPO in this setting, where memory helps stabilize control and improve tracking and tactical consistency compared to a purely feed-forward policy.

To ensure full reproducibility and isolate the effects of environment dynamics and training configuration, all runs were executed with a single fixed random seed, rather than averaging results across multiple seeds. As a consequence, observed performance differences can be directly attributed to the specific reward structure, scenario setup, and training hyperparameters, minimizing variability due to stochastic initialization or sampling noise.

Table 6.8 shows the total number of environment steps used to train each environment.

	LevelFlight	**FollowPoint**	**Dogfight**
SAC	$5 \cdot 10^6$	-	-
PPO	$5 \cdot 10^6$	$50 \cdot 10^6$	-
rPPO	-	$50 \cdot 10^6$	$50 \cdot 10^6$

Table 6.8: Total training steps per algorithm and environment (JSBSim).

Evaluation

For the single-agent environments, evaluation was performed by rolling out the trained, frozen policy for a fixed number of test episodes, using deterministic action selection to eliminate exploration noise and ensure directly comparable outcomes. For the multi-agent environments, evaluation followed the same protocol: the trained (frozen) policies were executed for a fixed number of test episodes, again with deterministic action selection.

As in the Toy Model setup, evaluation was kept fully reproducible by adopting a single

fixed random seed rather than averaging across multiple seeds. Consequently, all reported results originate from the same deterministic evaluation procedure and identical environment initialization conditions across models, ensuring that performance differences reflect the learned behavior under consistent testing conditions.

Hyperparameters

Parameter	Value
optimizer	Adam
learning_rate	0.0003
n_steps	256
batch_size	4096
n_epochs	20
gamma	0.99
gae_lambda	0.95
clip_range	0.30
clip_range_vf	None
ent_coef	0.02
vf_coef	0.5
max_grad_norm	0.5
target_kl	0.05
use_sde	True
sde_sample_freq	4
actor_arch	[128, 128], Tanh
critic_arch	[128, 128], Tanh

(a) PPO / SAC (JSBSim).

Parameter	Value
optimizer	Adam
learning_rate	0.00005
n_steps	256
batch_size	1024
n_epochs	10
gamma	0.99
gae_lambda	0.95
clip_range	0.20
clip_range_vf	None
ent_coef	0.003
vf_coef	0.5
max_grad_norm	0.5
target_kl	0.01
use_sde	False
sde_sample_freq	4
actor_arch	[128, 128], Tanh
critic_arch	[128, 128], Tanh
lstm_hidden_size	128
n_lstm_layers	1
shared_lstm	False
enable_critic_lstm	True

(b) rPPO (JSBSim).

Table 6.9: Hyperparameters for PPO/SAC and Recurrent PPO (rPPO) in the JSBSim environments.

Hyperparameters for the JSBSim training setup are reported in Table 6.9 for both PPO or SAC (6.9a) and Recurrent PPO (6.9b), which were the only algorithms considered in

this stage. Consistently with the Toy Model experiments, the policy optimization relied on Adam and standard on-policy settings, while the main differences between the two configurations reflect the different role of memory in rPPO. In particular, both methods employ actor and critic FC-MLP backbones with two hidden layers, but rPPO augments them with an LSTM module to cope with partial observability and delayed credit assignment typical of the flight-combat scenarios. The remaining hyperparameters follow common continuous-control conventions.

Visualization & Rendering

Visualization of learning curves and evaluation metrics in the JSBSim-based model was handled primarily through Weights & Biases (WandB). All training and evaluation runs were logged to WandB, including both standard RL diagnostics and a set of custom metrics explicitly defined in our codebase. These metrics were automatically aggregated and rendered as interactive plots during training and evaluation, and can also be exported in multiple formats (e.g., CSV) or explored through alternative WandB visualizations and dashboards.

In addition to quantitative monitoring, qualitative assessment of the learned behaviors was carried out using TacView, which was integrated into the simulation pipeline to enable a clearer inspection of agent motion and interaction dynamics in 3D space over complete episodes. This tooling proved essential to visually validate trajectory realism, engagement geometry, and overall behavioral consistency beyond what can be inferred from scalar metrics alone.

6.2.2. LevelFlight

Training Metrics

This first environment was trained on PPO and SAC, as its task was fairly easy and used as a benchmark. Fig. 6.26 shows the reward for both PPO and SAC.

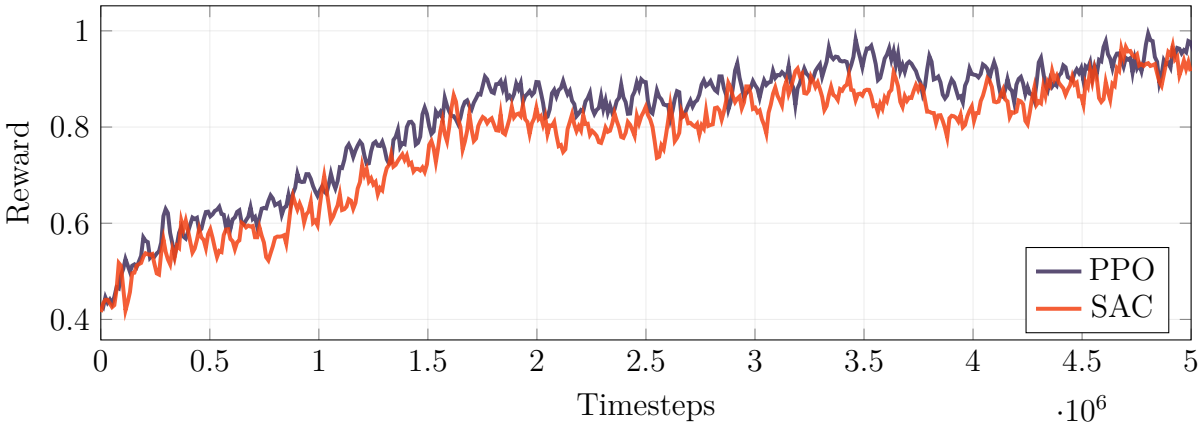
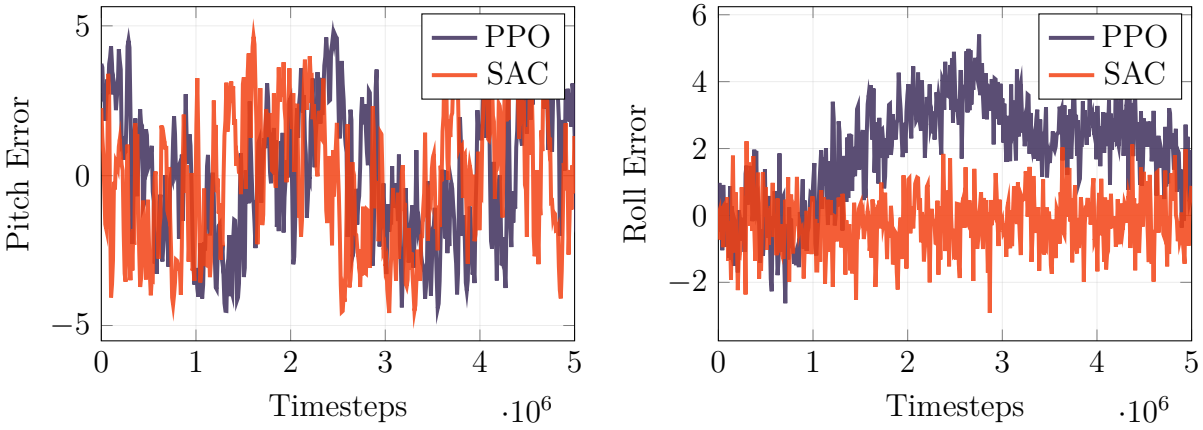


Figure 6.26: JSBSim LevelFlight: training reward for PPO and SAC.

Additional Training Metrics

It's possible to dive deeper into how the agent is trained in Fig. 6.27, which reports the two most important attitude angles to keep track of.



(a) Pitch error

(b) Roll error

Figure 6.27: JSBSim LevelFlight: pitch and roll angles.

Evaluation Metrics

Evaluation scores, computed over 10 independent runs, are summarized in Table 6.10. Results are reported as mean ± standard deviation.

Algorithm	Avg Reward	Roll Error ↓	Pitch Error ↓
PPO	0.905 ± 0.028	1.08 ± 0.22	1.17 ± 0.25
SAC	0.879 ± 0.036	1.21 ± 0.27	1.34 ± 0.31

Values are reported as mean and standard deviation over N=10 runs. Arrows indicate preferred direction of each metric.

Table 6.10: JSBSim LevelFlight: evaluation scores.

Evaluation Renders

To better understand performance and behavior of the agent, the 3D plot of a single episode, shown in Fig. 6.28. is shared. Agent behavior was represented by generating a plot indicating agent's flight path.

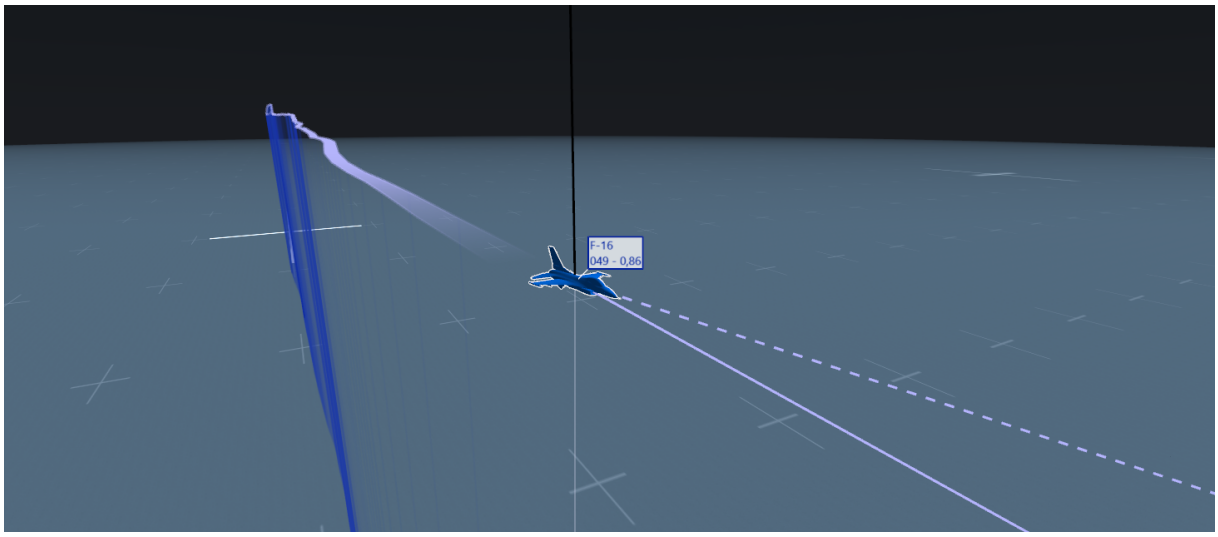


Figure 6.28: JSBSim LevelFlight: evaluation episode.

Discussion

In LevelFlight, the agent is trained to maintain a straight, steady cruise in JSBSim by tracking a desired heading while holding a target altitude, making it a deliberately easy benchmark to verify that the control loop, observations, and reward shaping are well-posed before moving to harder tasks. The training curves show that both PPO and SAC

converge rapidly: after an initial transient where exploration produces large deviations, the total reward increases steadily and reaches a stable high-reward regime. Overall, PPO achieves a slightly higher final performance than SAC (both in growth and in the end plateau), while SAC appears marginally smoother, with fewer oscillations during training. The auxiliary attitude metrics are consistent with this picture. Pitch and roll errors remain bounded within a narrow range and fluctuate around zero, suggesting that both agents learn to keep the aircraft close to a level attitude, but still exhibit noisy corrections due to coupled aircraft dynamics and the fact that the same heading/altitude objective can be achieved through multiple control combinations.

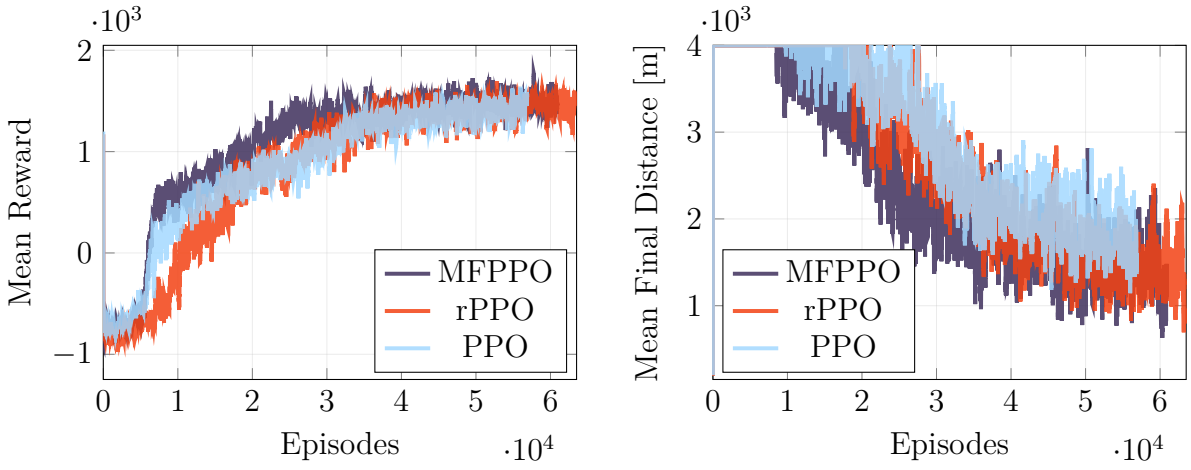
Overall, LevelFlight validates that the reward components properly guide the agent toward the intended equilibrium, and suggests that the main difficulty is achieving precise heading regulation under realistic flight coupling and noisy corrective actions, especially when prioritizing smoothness versus peak performance.

6.2.3. FollowPoint

Since the FollowRandomPoint scenario is more challenging and therefore more informative than the standard FollowPoint setup, the analysis focuses exclusively on the random-target case. Also, as a further benchmark, a MultiFrame [58] version of PPO was tested.

Training Reward

Fig. 6.29 shows the training reward plot together with the final distance to the target during agent training.



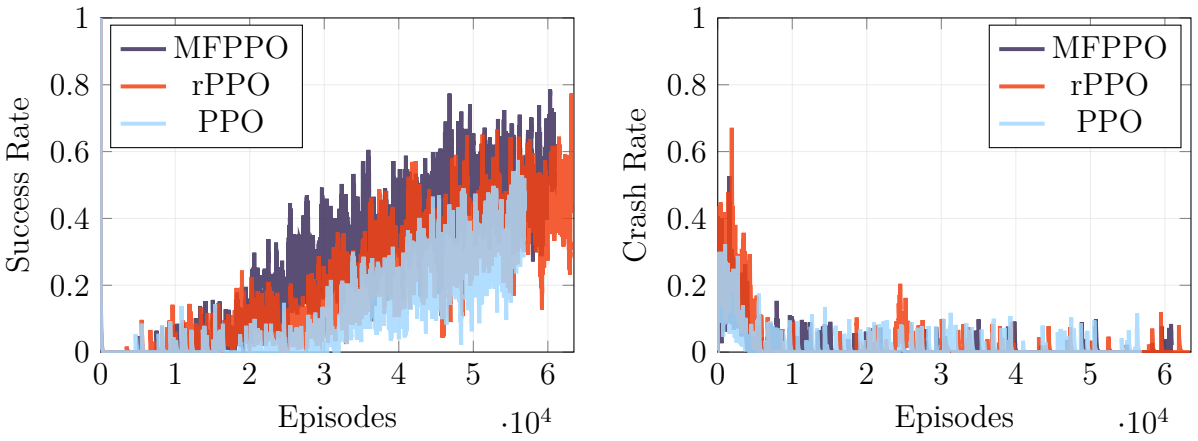
(a) Mean episode reward

(b) Mean final distance

Figure 6.29: JSBSim FollowPoint: reward and mean final distance (MFPPO vs rPPO vs PPO).

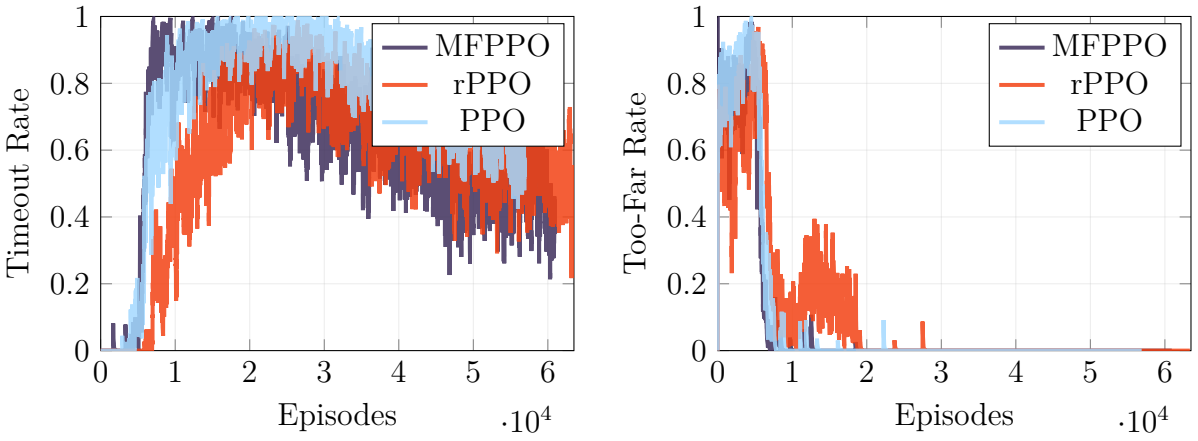
Additional Training Metrics

It is possible to better interpret training performance thanks to the plots shown from Fig. 6.30 to Fig. 6.32.



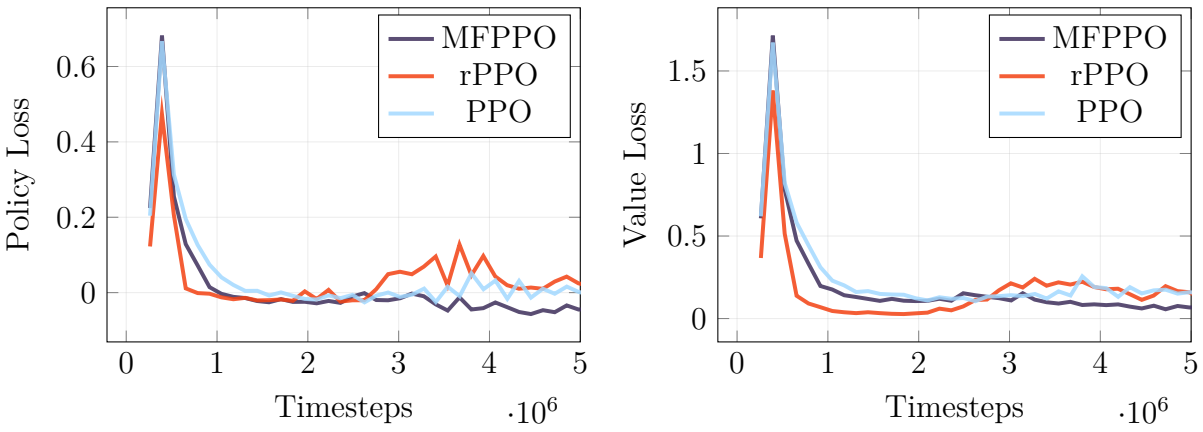
(a) Success rate (b) Crash rate

Figure 6.30: JSBSim FollowPoint: success and crash rates (MFPPO vs rPPO vs PPO).



(a) Timeout rate (b) Too-far rate

Figure 6.31: JSBSim FollowPoint: timeout and too-far rates (MFPPO vs rPPO vs PPO).



(a) Policy loss

(b) Value loss

Figure 6.32: JSBSim FollowPoint: policy and value losses (MFPPO vs rPPO vs PPO).

Evaluation Metrics

Evaluation scores, computed over 100 independent runs, are summarized in 6.11. Results are reported as mean ± standard deviation.

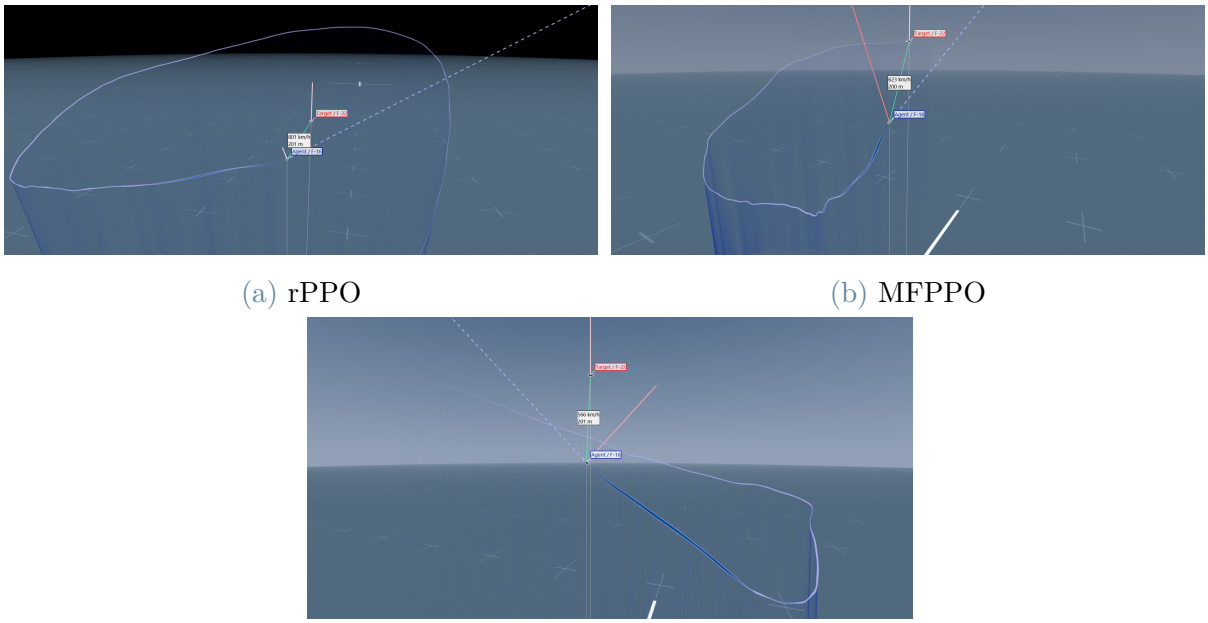
Algorithm	Avg Reward	Steps/episode ↓	Avg distance ↓	Success ratio [%] ↑
PPO	942.334 ± 710.626	791.85 ± 287.49	2457.827 ± 1920.086	28.00 ± 45.13
MFPPO	1377.205 ± 533.568	791.31 ± 275.69	1849.028 ± 2258.573	44.00 ± 49.89
rPPO	1485.767 ± 330.823	791.12 ± 287.67	1435.564 ± 1376.540	44.00 ± 49.89

Values are reported as mean and standard deviation over N=100 runs. Arrows indicate preferred direction of each metric.

Table 6.11: JSBSim FollowPoint: evaluation scores.

Evaluation Renders

To better characterize the agents performance and behavior, 3D plots from two representative episodes are provided in Fig. 6.33. The agents behavior is illustrated through plots showing the flight trajectory alongside the target position.



(a) rPPO (b) MFPPO

(c) PPO

Figure 6.33: JSBSim FollowPoint: evaluation episodes with rPPO, MFPPO and PPO.

Discussion

The FollowPoint task evaluates the capability of a continuous-control RL agent to navigate a high-fidelity JSBSim aircraft toward a randomly sampled target position in 3D space. Compared to the fixed-target variant, the random-target setting is deliberately more informative because it reduces memorization effects and forces the policy to generalize across a broad distribution of relative geometries. In this context, training curves and auxiliary metrics highlight not only how quickly each algorithm improves, but also how it improves.

From the training reward and final distance trends (Fig. 6.29), all methods show a clear learning phase where reward rapidly increases as the agent discovers a viable closed-loop strategy to reduce the target error. MultiFramePPO (MFPPO) consistently reaches higher reward earlier and exhibits a more pronounced reduction in final distance, suggesting that the stacked temporal context helps mitigate partial observability and improves short-horizon prediction of target-relative dynamics. RecurrentPPO (rPPO) follows a similar trajectory and achieves comparable asymptotic performance, with a slightly slower early improvement but a steadier progression once learning stabilizes. Standard PPO, instead, remains systematically behind: its reward growth is slower and noisier, and its final distance curve indicates a weaker ability to consistently converge close to the target under the randomization of the scenario. This gap is consistent with the intuition that single-

step observations can be insufficient to fully infer state trends (e.g., turning dynamics) in a flight-control setting, especially when the target distribution induces diverse approach angles and energy states.

The additional metrics (Fig. 6.30-Fig. 6.32) provide a more diagnostic view of what better means in this environment. The success rate increases for all methods, but it grows earlier and reaches higher values for MFPPO and rPPO, while PPO lags and displays larger variability. In parallel, the crash rate rapidly collapses toward zero after the initial exploration phase for all algorithms, indicating that catastrophic failures are mainly concentrated at the beginning of training and do not dominate long-run performance. This is an important observation: differences between methods are explained by their ability to reliably achieve the termination conditions associated with "reaching the target" within the operational constraints. Conversely, timeout and too-far rates clarify the dominant failure modes after learning begins. The timeout rate increases sharply early on, which is expected when the policy learns to keep the aircraft controlled and avoid crashes but still lacks an efficient guidance strategy to close the distance fast enough. Over training, MFPPO and rPPO progressively reduce this inefficiency, translating into higher success, whereas PPO shows a persistently worse balance between closing behavior and time budget. The too-far rate being prominent at the very start and then dropping toward zero for all approaches suggests that policies quickly learn to remain within the engagement region; once basic stabilizing and directional control is acquired, the key challenge becomes precision and efficiency rather than simply staying near the target. Loss curves are consistent with stable optimization for all algorithms: both policy and value losses decay rapidly from their initial peaks and then settle into a lower-variance regime. Notably, rPPO exhibits the most stable behavior in practice, matching the qualitative observation from rollouts: recurrent memory tends to smooth control by filtering short-term observation noise and implicitly integrating temporal information. MFPPO also benefits from temporal context, but in a more rigid way: frame stacking can improve decision quality, yet it does not provide the same adaptive internal state that a recurrent model can leverage to maintain smooth trajectories across varying approach conditions.

Evaluation results further support the overall picture: MFPPO and rPPO outperform PPO across accuracy-oriented metrics, indicating better tracking quality and more consistent target acquisition. Furthermore, qualitative rollouts and training stability strongly suggest that rPPO matches MFPPO in final performance while delivering smoother, more realistic control. This distinction is relevant for downstream use: in an aircraft setting, the best controller is not only the one that reaches the target, but the one that does so without unnecessary oscillations, aggressive corrections, or control chatter. Importantly,

rPPO achieves these control benefits without an apparent training-time penalty compared to MFPPO, making it a strong candidate when both sample efficiency and control smoothness are priorities.

6.2.4. Fully Observable Dogfight

This chapter presents the training and evaluation results for the environment discussed in Chapter 5.6.2.

Zerosum Training Reward

The plot in Fig. 6.34 shows the total episode return recorded during training when Agent 0 or Agent 1 is being trained, respectively; therefore, each curve effectively reports the return achieved by the agent currently under training.

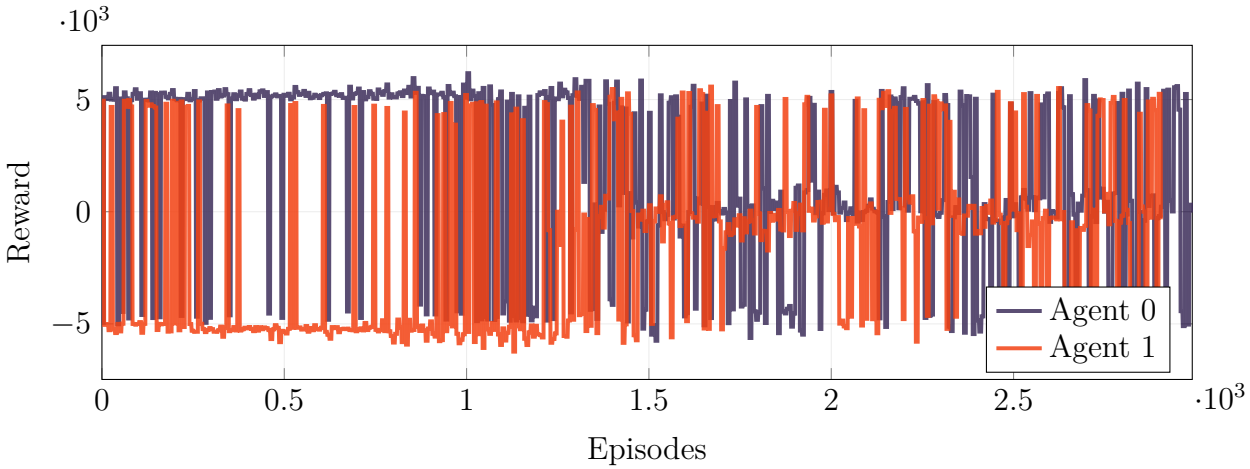
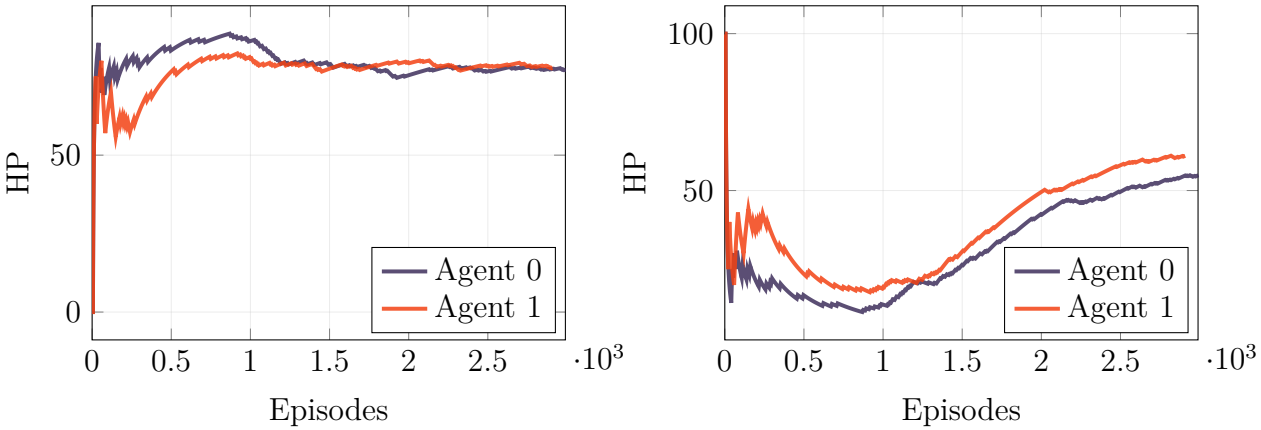


Figure 6.34: JSBSim MDP Zerosum Dogfight: total episode return.

Additional Zerosum Training Metrics

Figure 6.35 reports the evolution of the agents health points (HP) during training, explicitly separating the two training phases (training Agent 0 vs training Agent 1). In particular, Fig. 6.35a shows how Agent 0's HP fluctuate, while Fig. 6.35b reports how Agent 1's HP variate, with each subplot including both phases to highlight how the same health variable behaves when the corresponding agent is the learner versus when it is the opponent.



(a) Agent 0 HP

(b) Agent 1 HP

Figure 6.35: JSBSim MDP Zerosum Dogfight: HP trends for both agents across both training phases.

Zerosum Evaluation Metrics

Evaluation scores, computed over 10 independent runs, are summarized in Table 6.12. Results are reported as mean ± standard deviation. Values for both agents are reported, shown as (0) and (1).

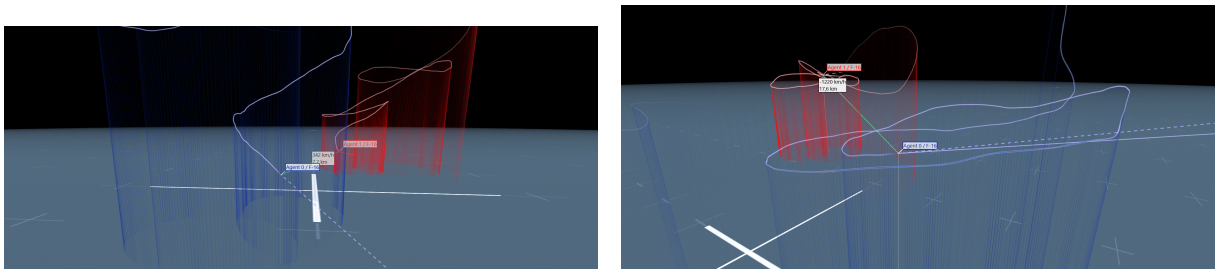
Algorithm	Reward (0) ↑	Reward (1) ↑	Shots Landed (0) ↑	Shots Landed (1) ↑
rPPO	-0.790 ± 3.382	0.790 ± 3.382	0.0 ± 0.0	0.0 ± 0.0

Values are reported as mean and standard deviation over N=10 runs. Arrows indicate preferred direction of each metric.

Table 6.12: JSBSim MDP Zerosum Dogfight: evaluation scores.

Zerosum Evaluation Renders

To better characterize the performance and behavior of both agents, 3D plots from two representative episodes are provided in Fig. 6.36. The agents' behavior is illustrated through plots showing the flight trajectories of both agents.



(a) Evaluation episode 1 (b) Evaluation episode 2

Figure 6.36: JSBSim MDP Zerosum Dogfight: evaluation episodes with rPPO.

Non-Zerosum Training Reward

The plot in Fig. 6.37 shows the total episode return recorded during training when Agent 0 or Agent 1 is being trained, respectively; therefore, each curve effectively reports the return achieved by the agent currently under training.

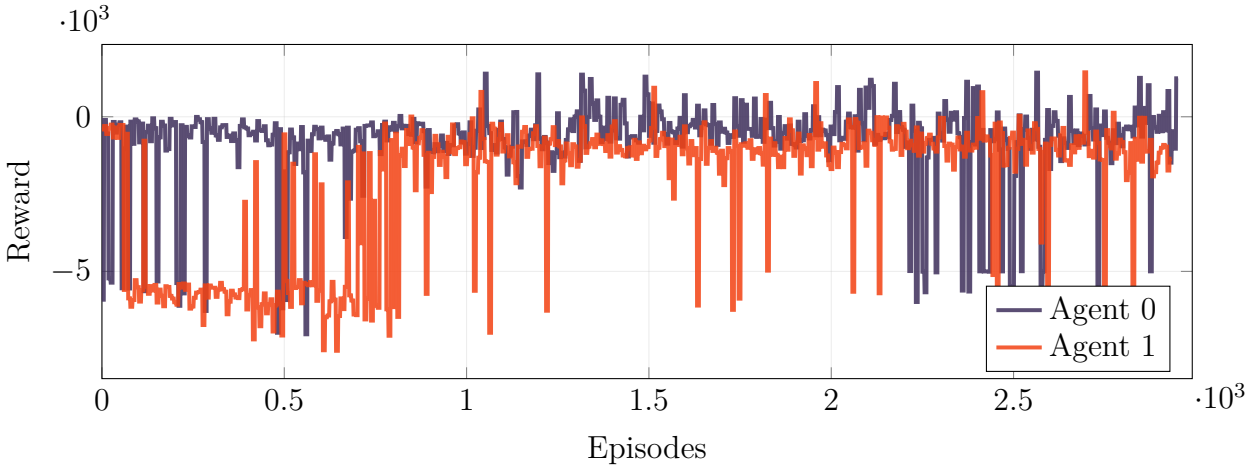
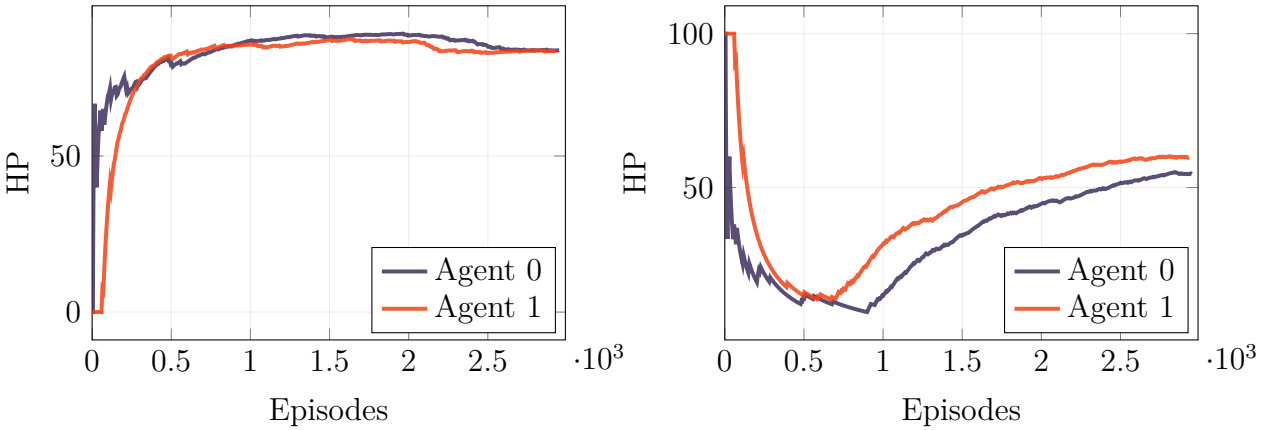


Figure 6.37: JSBSim MDP Non-Zerosum Dogfight: total episode return

Additional Non-Zerosum Training Metrics

Figure 6.38 reports the evolution of the agents health points (HP) during training, explicitly separating the two training phases (training Agent 0 vs training Agent 1). In particular, Fig. 6.38a shows how Agent 0's HP fluctuate, while Fig. 6.38b reports how Agent 1's HP variate, with each subplot including both phases to highlight how the same health variable behaves when the corresponding agent is the learner versus when it is the opponent.



(a) Agent 0 HP

(b) Agent 1 HP

Figure 6.38: JSBSim MDP Non-Zerosum Dogfight: HP trends for both agents across both training phases.

Non-Zerosum Evaluation Metrics

Evaluation scores, computed over 10 independent runs, are summarized in Table 6.13. Results are reported as mean ± standard deviation. Values for both agents are reported, shown as (0) and (1).

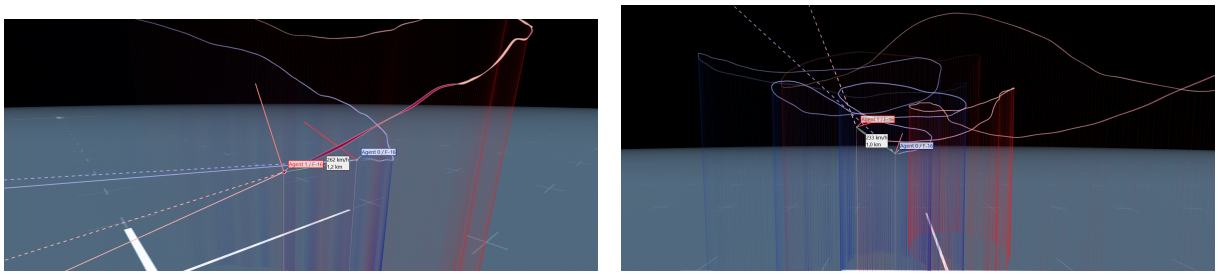
Algorithm	Reward (0) ↑	Reward (1) ↑	Shots Landed (0) ↑	Shots Landed (1) ↑
rPPO	-0.1388 ± 0.3905	-0.4491 ± 0.2096	6.700 ± 7.529	2.500 ± 3.9791

Values are reported as mean and standard deviation over N=10 runs. Arrows indicate preferred direction of each metric.

Table 6.13: JSBSim MDP Non-Zerosum Dogfight: evaluation scores.

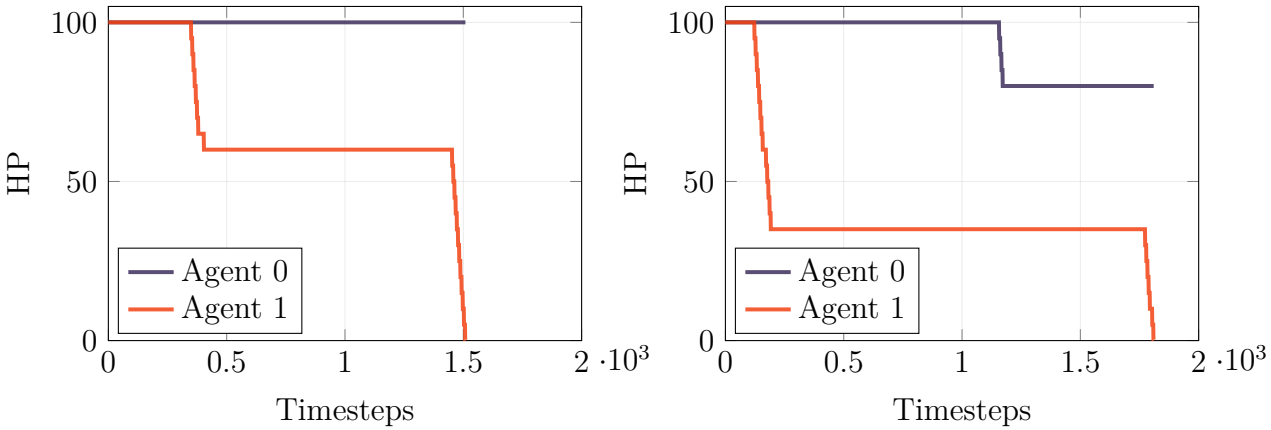
Non-Zerosum Evaluation Renders

To provide an additional view of both agents' performance and behavior, Fig. 6.39 reports 3D visualizations from two sample episodes. The resulting plots depict the flight paths of both agents over time.



(a) Evaluation episode 1 (b) Evaluation episode 2

Figure 6.39: JSBSim MDP Non-Zerosum Dogfight: evaluation episodes with rPPO.



(a) Evaluation episode 1 (b) Evaluation episode 2

Figure 6.40: JSBSim MDP Non-Zerosum Dogfight: HP trajectories over timesteps for the two above evaluation episodes.

6.2.5. Long Fully Observable Dogfight

Given that the short fully observable experiment with the Non-Zerosum reward produced more convincing outcomes than the corresponding Zerosum setting, it was selected as the baseline formulation for a longer training run. This extended run is intended to assess whether the early advantages remain stable and scale with additional experience, and whether prolonged optimization further consolidates the learned behaviors into more consistent engagement and overall performance.

Training Reward

Instead of showing the full return, the base reward is shown in Fig. 6.41 and the dogfighting components in Fig. 6.42.

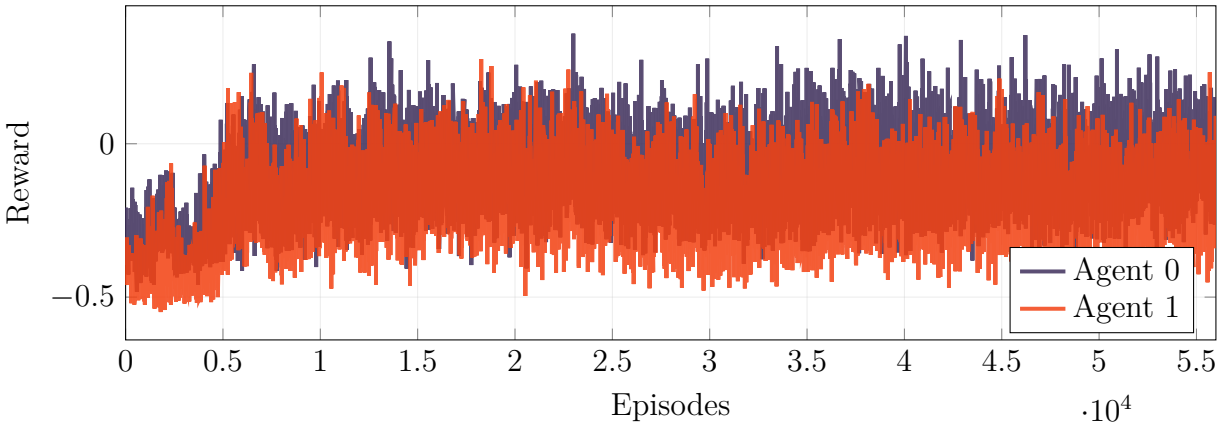
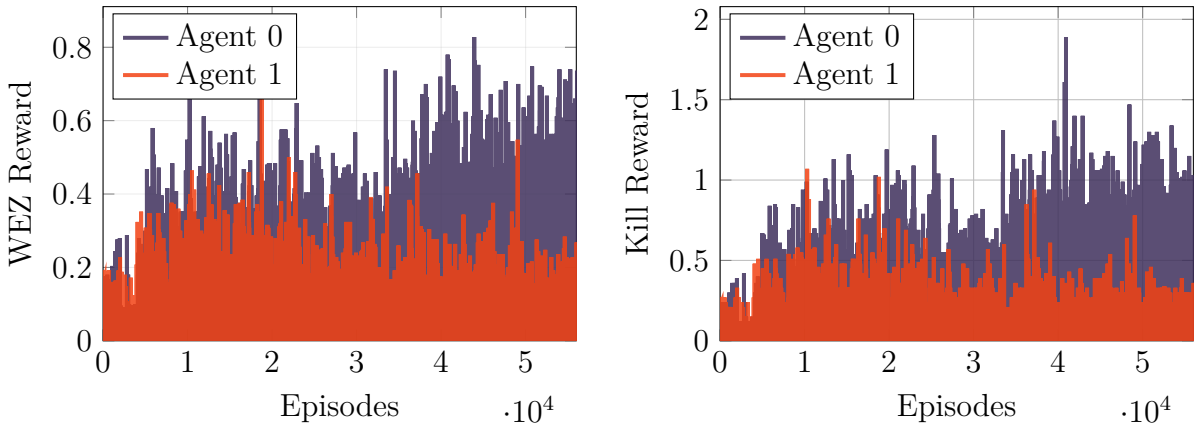


Figure 6.41: JSBSim Long MDP Dogfight: base reward over training (Agent 0 vs Agent 1).



(a) WEZ reward

(b) Kill reward

Figure 6.42: JSBSim Long MDP Dogfight: WEZ and kill rewards (Agent 0 vs Agent 1).

Additional Training Metrics

A more extensive visualization of what happens during training is provided by Fig. 6.43 to Fig. 6.45.

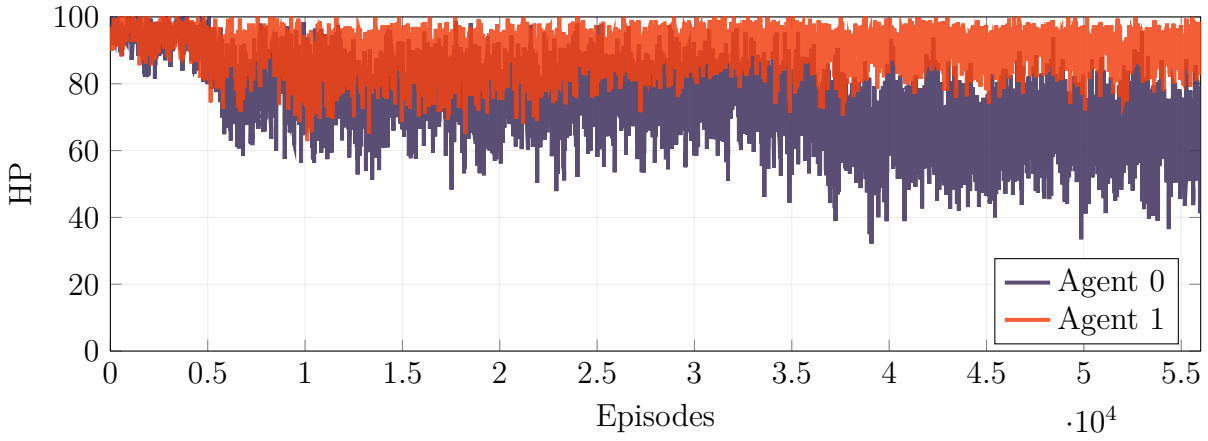
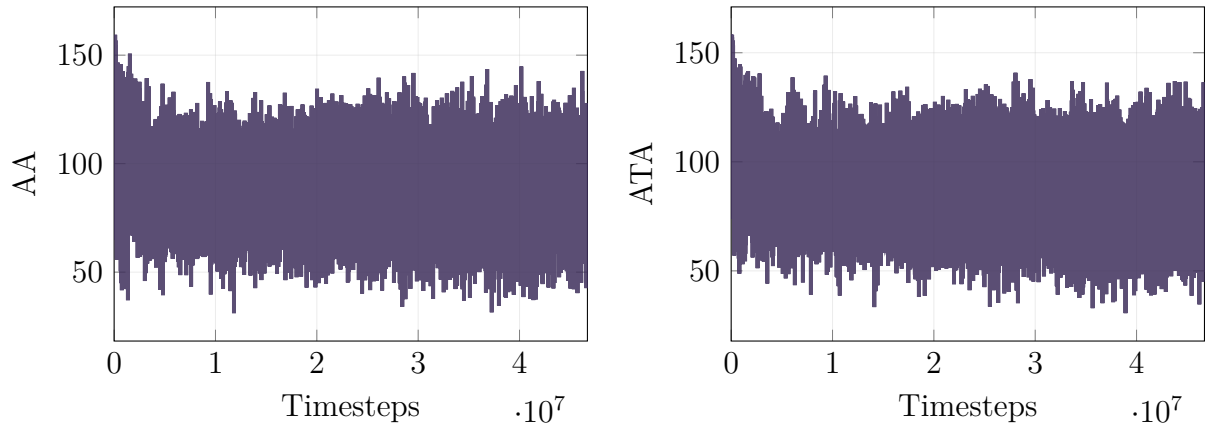
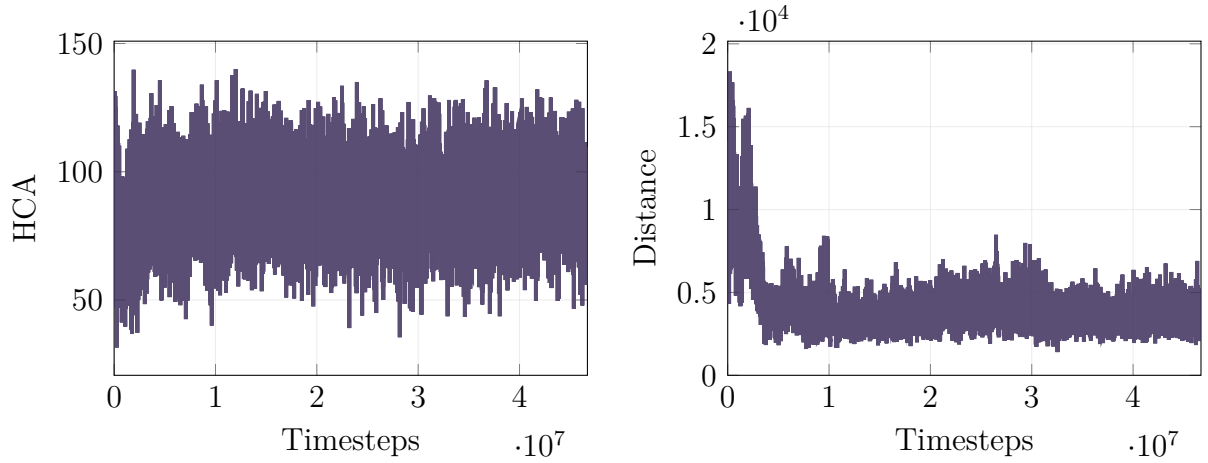


Figure 6.43: JSBSim Long MDP Dogfight: Training HP (Agent 0 vs Agent 1).



(a) AA (b) ATA

Figure 6.44: JSBSim Long MDP Dogfight: AA and ATA.



(a) HCA (b) Distance

Figure 6.45: JSBSim Long MDP Dogfight: HCA and distance.

Evaluation Metrics

Evaluation scores, computed over 10 independent runs, are summarized in Table 6.14. Results are reported as mean ± standard deviation. Values are provided for both agents, denoted as (0) and (1).

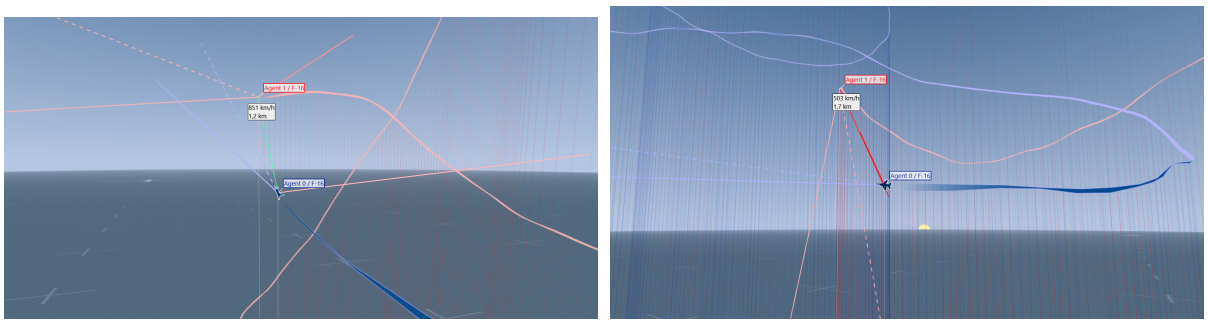
Algorithm	Reward (0) ↑	Reward (1) ↑	Shots Landed (0) ↑	Shots Landed (1) ↑
rPPO	0.0819 ± 0.5485	-0.8498 ± 0.7913	15.000 ± 4.028	1.700 ± 1.567

Values are reported as mean and standard deviation over N=10 runs. Arrows indicate preferred direction of each metric.

Table 6.14: JSBSim Long MDP Dogfight: evaluation scores.

Evaluation Renders

To provide further insight into the performance and behavior of both agents, 3D plots from two representative episodes are reported in Fig. 6.46. The agents' behavior is illustrated through plots showing the flight trajectories of both agents.



(a) Evaluation episode 1 (b) Evaluation episode 2

Figure 6.46: JSBSim Long MDP Dogfight: evaluation episodes with rPPO.

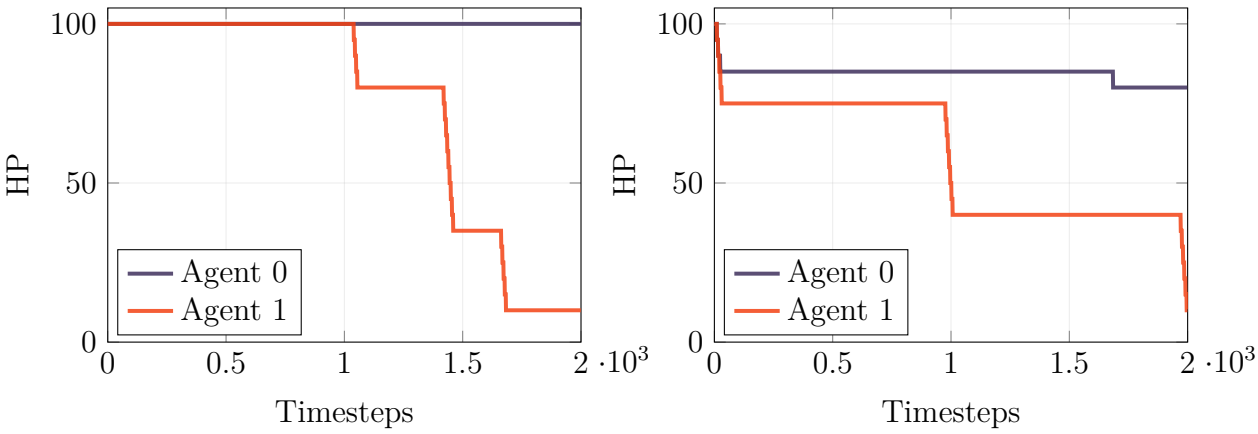


Figure 6.47: JSBSim Long MDP Dogfight: HP trajectories over timesteps for the two above evaluation episodes.

6.2.6. Long Observable Dogfight Variant

This variant retains the fully observable setting used in the previous environment, but modifies the observation design by removing the explicit adversary state from the observation vector while maintaining tactical-geometry angles. This run is used to assess whether comparable performance can be retained when removing part of the fully observable input in exchange for a lighter observation space and faster training.

Training Reward

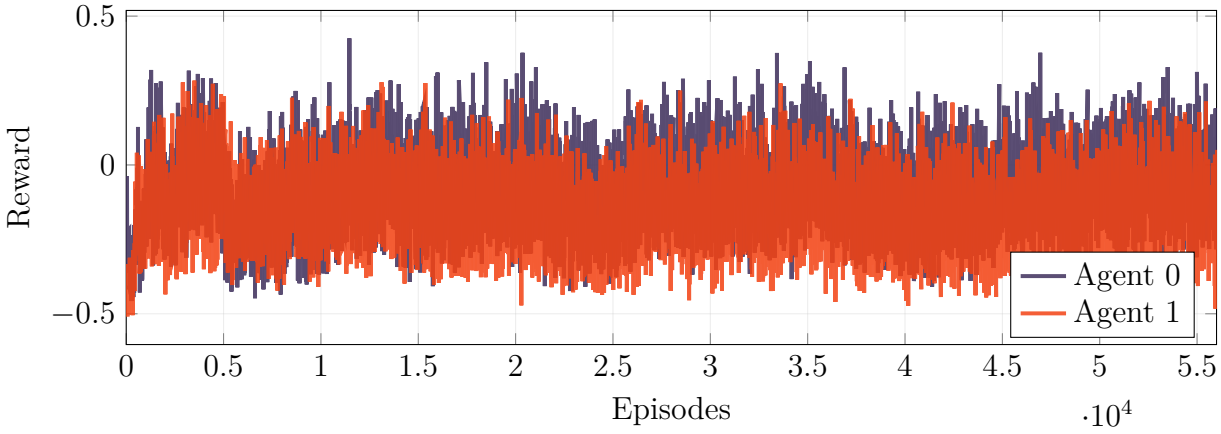
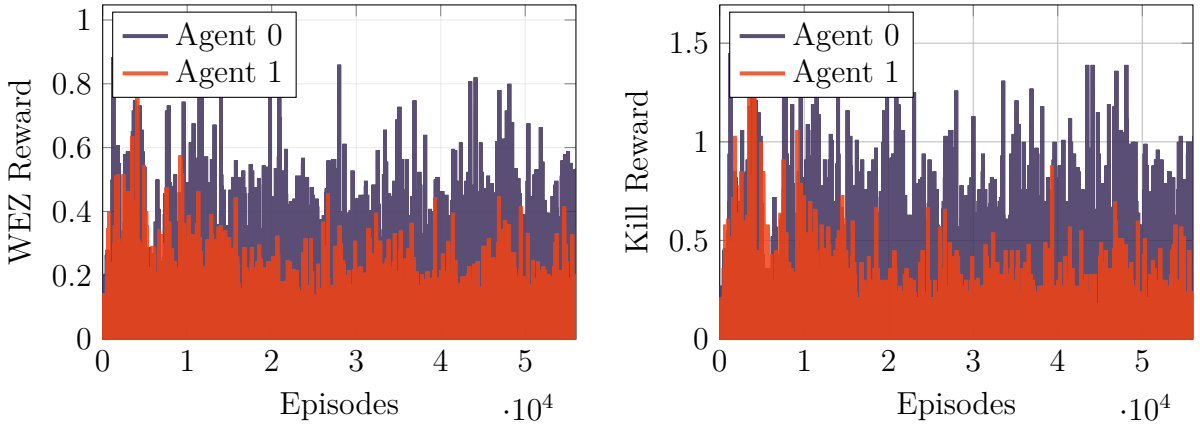


Figure 6.48: JSBSim Long MDP Variant Dogfight: base reward over training (Agent 0 vs Agent 1).



(a) WEZ reward

(b) Kill reward

Figure 6.49: JSBSim Long MDP Variant Dogfight: WEZ and kill rewards (Agent 0 vs Agent 1).

Additional Training Metrics

A more detailed view of the training dynamics is provided in Fig. 6.50 to Fig. 6.52.

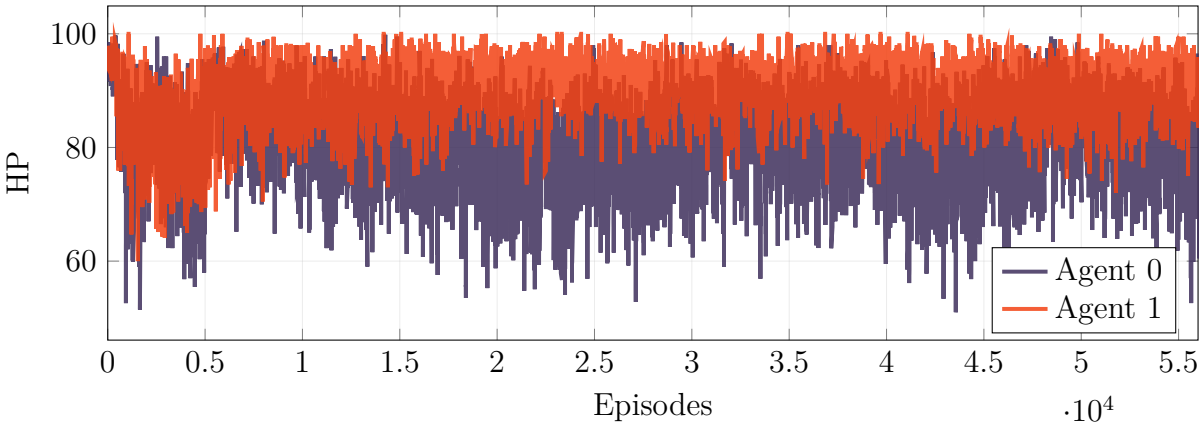
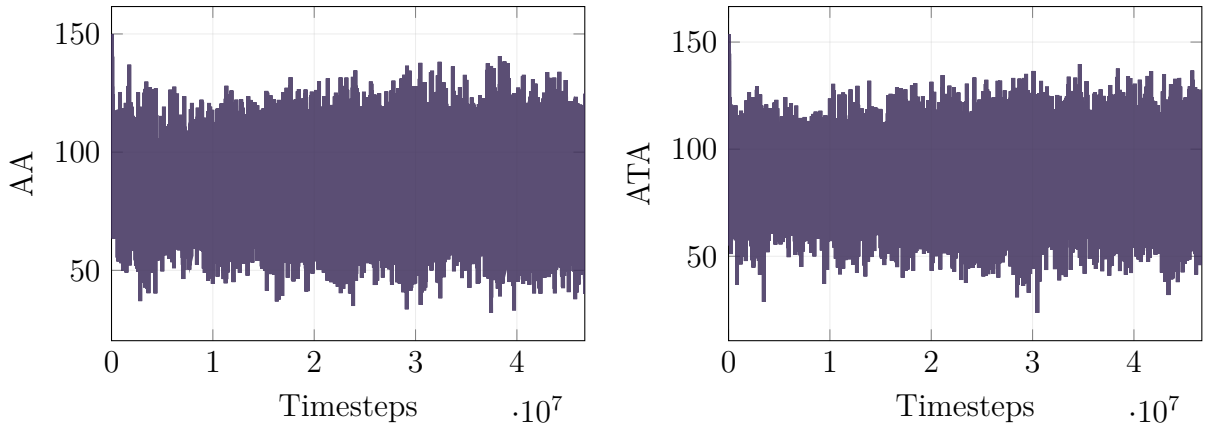
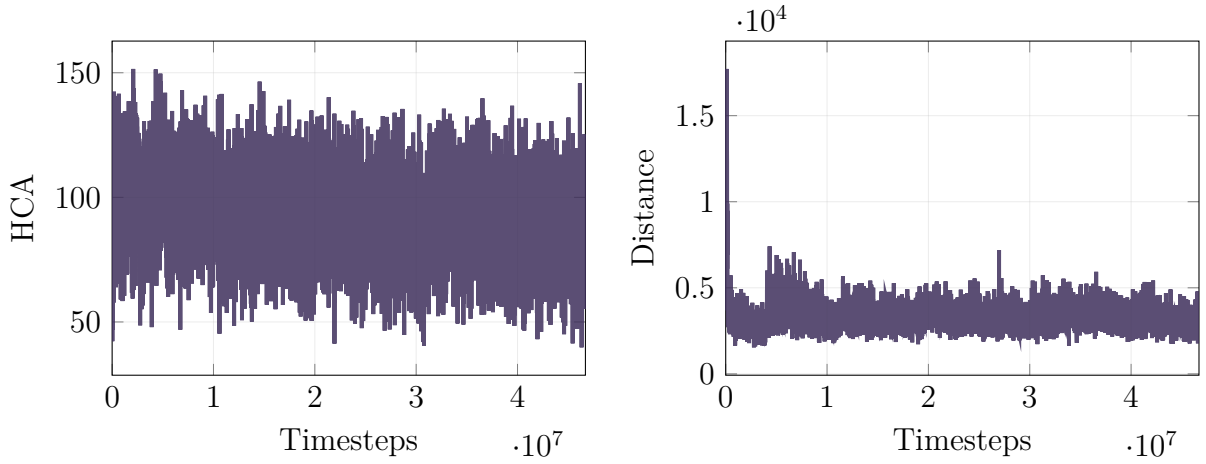


Figure 6.50: JSBSim Long MDP Variant Dogfight: Training HP (Agent 0 vs Agent 1).



(a) AA (b) ATA

Figure 6.51: JSBSim Long MDP Variant Dogfight: AA and ATA.



(a) HCA (b) Distance

Figure 6.52: JSBSim Long MDP Variant Dogfight: HCA and distance.

Evaluation Metrics

Evaluation scores, computed over 10 independent runs, are summarized in Table 6.15. Results are reported as mean ± standard deviation. Both agents' values are reported, shown as (0) and (1).

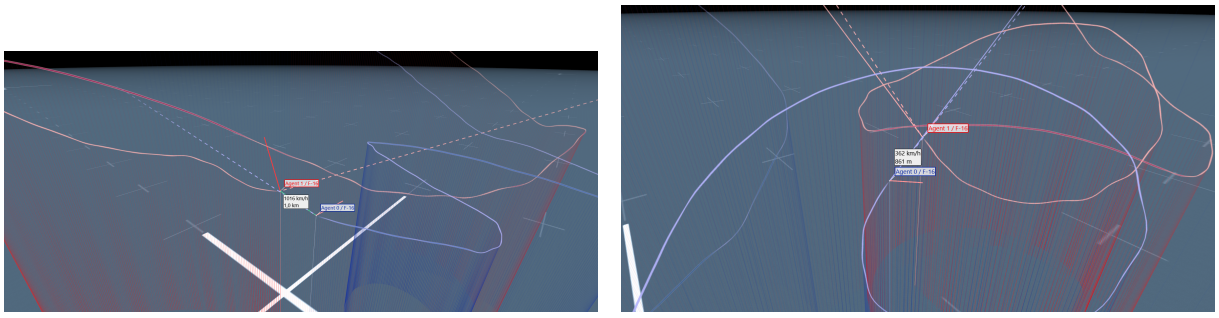
Algorithm	Reward (0) ↑	Reward (1) ↑	Shots Landed (0) ↑	Shots Landed (1) ↑
rPPO	-0.0560 ± 0.2657	-0.4292 ± 0.1579	8.300 ± 6.634	1.300 ± 2.312

Values are reported as mean and standard deviation over N=10 runs. Arrows indicate preferred direction of each metric.

Table 6.15: JSBSim Long MDP Variant Dogfight: evaluation scores.

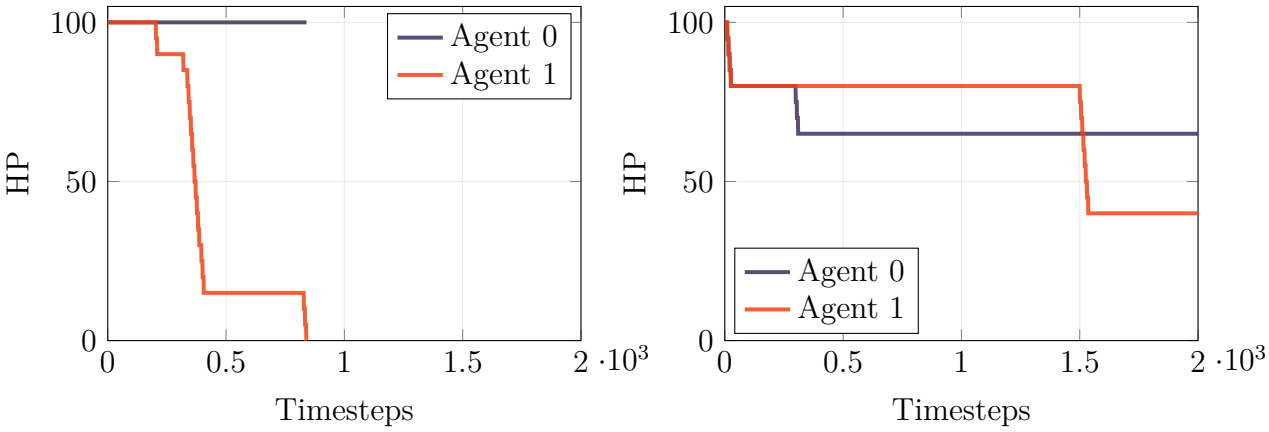
Evaluation Renders

To further characterize the performance and behavior of both agents, 3D visualizations from two representative episodes are reported in Fig. 6.53. The plots depict the flight trajectories of both agents.



(a) Evaluation episode 1 (b) Evaluation episode 2

Figure 6.53: JSBSim Long MDP Variant Dogfight: evaluation episodes with rPPO.



(a) Evaluation episode 1 (b) Evaluation episode 2

Figure 6.54: JSBSim Long MDP Variant Dogfight: HP trajectories over timesteps for the two above evaluation episodes.

6.2.7. Discussion

This section discusses the results obtained across the Dogfight experiments shown in the previous subsections, focusing on how reward structure, training horizon, and observability affect learning stability, emergent engagement behaviors, and evaluation performance. In Chapter 6.2.4, each agent receives a rich observation that includes its own state, the opponents state, and explicit air-combat geometry features. After an initial short-horizon comparison between Zerosum and Non-Zerosum rewards (both in Chapter 6.2.4), the Non-Zerosum formulation was selected for a long training run (reported in Chapter 6.2.5) because it produced more convincing behaviors and measurable combat outcomes. Finally, to isolate the contribution of opponent state information, a long-horizon reduced-observation variant (reported in Chapter 6.2.6) was trained with the same Non-Zerosum reward but with the opponent's state removed from the observation.

Fully Observable Dogfight

The short-horizon Zerosum experiment highlights a typical challenge of competitive self-play: the learning dynamics are highly non-stationary and can lead to unstable training signals. The total episode return (Fig. 6.34) oscillates sharply, with frequent collapses and recoveries, consistent with the learner constantly adapting to an evolving opponent. Importantly, this instability does not translate into effective offensive capability. In evaluation, both agents record zero shots landed (as reported in Table 6.12), suggesting that the learned strategies either fail to consistently reach and maintain favorable firing conditions or converge to behaviors that prioritize avoiding decisive loss over actively closing engagements. The training-time health trends (HP) also do not show a clear and persistent dominance pattern across phases (Fig. 6.35), and the evaluation renders support the impression of less decisive engagement trajectories (Fig. 6.36).

Switching to a Non-Zerosum reward already changes this picture in the short-horizon setting. While the episode return still exhibits variability (as shown in Fig. 6.37), evaluation now reports that the agent begins to discover behaviors that lead to actual damage and engagement closure rather than merely maintaining survivability (Table 6.13). The HP trajectories in evaluation episodes (Fig. 6.38) further corroborate this: the presence of pronounced HP drops in some episodes is consistent with resolved engagements (kills or substantial damage). Practically, these results motivated using the Non-Zerosum reward as the baseline for a longer training run: since the short fully observable Non-Zerosum setup produced better outcomes than the Zerosum one, training was extended to test whether these improvements consolidate over time.

Long Fully Observable Dogfight

Chapter 6.2.5 run probes whether Non-Zerosum shaping and extended optimization yield not only higher returns but also more stable and scalable combat behavior. From the reward perspective, the base reward curve remains noisy, as expected in a dogfight setting, but it settles into a more consistent regime over long training (Fig. 6.41). It is possible to see that WEZ reward and kill reward become more sustained and frequent throughout training (Fig. 6.42). This indicates that, with sufficient training time, the agent increasingly visits states associated with weapon-effective positioning and occasionally converts them into terminal combat outcomes.

The additional metrics reinforce this interpretation. HP trends during training exhibit an increasing asymmetry (Fig. 6.43), suggesting that the self-play process can converge to a regime where one policy gains a systematic advantage that the counterpart does

not fully counterbalance. The geometric combat indicators remain highly variable but remain compatible with more consistent engagement control (Figs. 6.44-6.45): in dogfight dynamics, small differences in timing and maneuvers can quickly amplify, so high variance is expected, but the key observation is the persistent presence of patterns consistent with repeated engagement attempts rather than avoidance.

Evaluation results shown in Table 6.14 confirm a substantial gain in offensive effectiveness for the stronger agent: Agent 0 lands many more shots, implying an emergent dominance under self-play when the observation is fully informative. The qualitative renders and HP evaluation trajectories (Figs. 6.46-6.47) further support the conclusion that long training with full observability yields episodes where engagements are closed more decisively and more repeatably than in the short-horizon setting.

Long Observable Dogfight Variant

To assess whether the long-horizon improvement is primarily due to longer optimization or also depends critically on the opponent-state observability, a variant was trained using the same Non-Zerosum reward but with a reduced observation: the agent retains its own state and explicit geometry features, while the opponent's full state is removed. Training curves show that the agent can still learn engagement-related behaviors, as reflected in the base reward and the WEZ/kill components (Figs. 6.48-6.49), but the overall learning appears less consistent than in the fully observable long run.

This is also reflected in the auxiliary metrics. HP and geometry trends remain noisy (Figs. 6.50-6.52), and the consolidation into a clearly dominant, repeatable engagement strategy is less evident. The evaluation table (Table 6.15) makes the performance gap explicit: Agent 0 shows roughly half of the fully observable long result (Table 6.14), and with substantially higher variance, indicating poorer repeatability across episodes. The qualitative renders of Fig. 6.53 and the HP evaluation trajectories of Fig. 6.54 show that engagements can still be resolved, but successful closures appear more episodic and less reliable. Overall, removing the opponent state does not prevent learning, but it reduces the agents ability to plan and execute consistently informed maneuvers that maximize engagement effectiveness.

Overall Interpretation & Implications

Taken together, the experiments support three main conclusions:

- Non-Zerosum shaping is a key enabler for actionable dogfight behavior. In the short-horizon fully observable setup, it is the difference between a regime with no

measurable offensive success (Table 6.12) and one where shots landed emerge already (Table 6.13).

- Extended training consolidates engagement competence and can produce dominant strategies under self-play, particularly when observations are fully informative. This is supported by the sustained WEZ/kill components and by the marked increase in shots landed (Table 6.14).
- Opponent-state information improves both effectiveness and consistency. When the opponents state is removed, the agent retains some capability through geometry features, but evaluation shows lower mean performance and much higher variability (Table 6.15 vs Table 6.14), implying less robust engagement closure.

In summary, for a self-play dogfight task in JSBSim, a purely antagonistic Zerosum formulation can lead to unstable learning signals and weak engagement outcomes, whereas a Non-Zerosum reward provides a denser, more behaviorally aligned signal that promotes offensive competence. With sufficient training time, fully informative observations support more stable and scalable learning, yielding higher and more repeatable combat effectiveness compared to reduced-observation variants.

6.2.8. Partially Observable Dogfight

As a complement to the fully observable setting, a partially observable formulation of the dogfight task was also considered, treating it as a POMDP where each agent has access only to a restricted subset of information, as reported in Chapter 5.6.2. The goal of this additional experiment is not to redesign the environment, but to evaluate how performance and stability change under limited observability, and to enable a direct comparison with the fully observable results reported in the previous section.

Zerosum Training Reward

The plot in Fig. 6.55 shows the total episode return recorded during training when Agent 0 or Agent 1 is being trained, respectively; therefore, each curve effectively reports the return achieved by the agent currently under training.

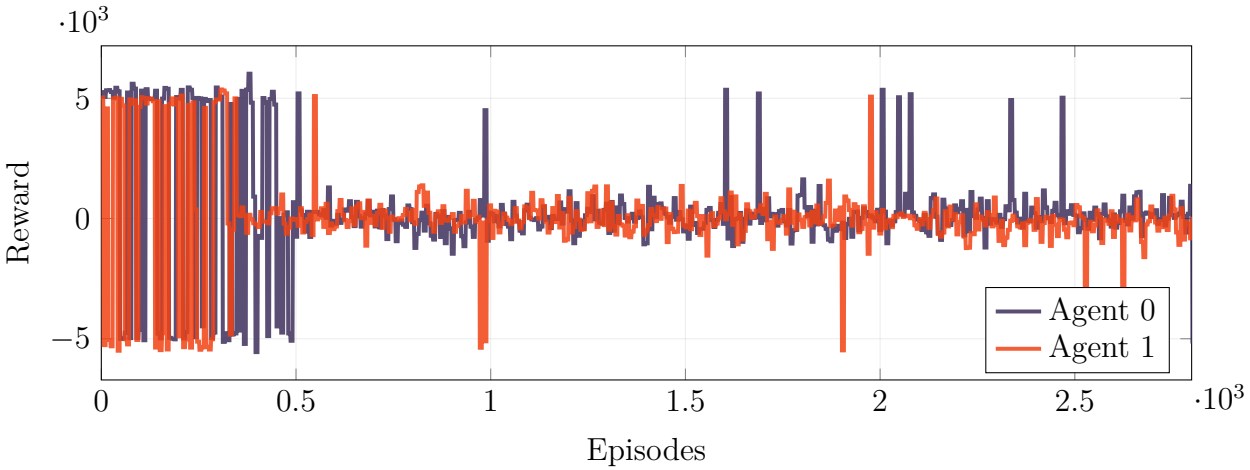
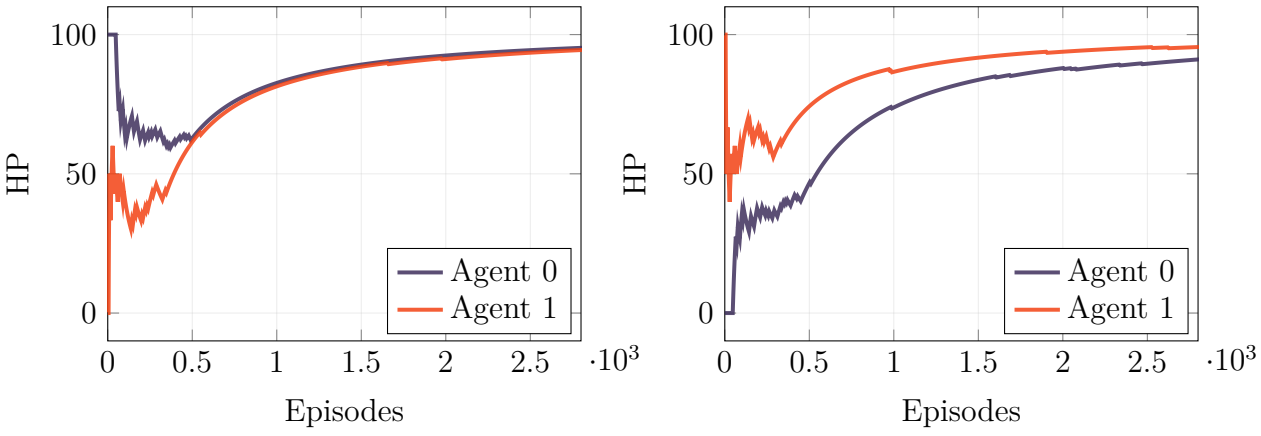


Figure 6.55: JSBSim POMDP Zerosum Dogfight: total episode return.

Additional Zerosum Training Metrics

Figure 6.56 reports the evolution of the agents health points (HP) during training, explicitly separating the two training phases (training Agent 0 vs training Agent 1). In particular, Fig. 6.56a shows how Agent 0's HP fluctuate, while Fig. 6.56b reports how Agent 1's HP variate, with each subplot including both phases to highlight how the same health variable behaves when the corresponding agent is the learner versus when it is the opponent.



(a) Agent 0 HP

(b) Agent 1 HP

Figure 6.56: JSBSim POMDP Zerosum Dogfight: HP trends for both agents across both training phases.

Zerosum Evaluation Metrics

Evaluation scores, computed over 10 independent runs, are summarized in Table 6.16. Results are reported as mean ± standard deviation. Once again, values for both agents are reported, shown as (0) and (1).

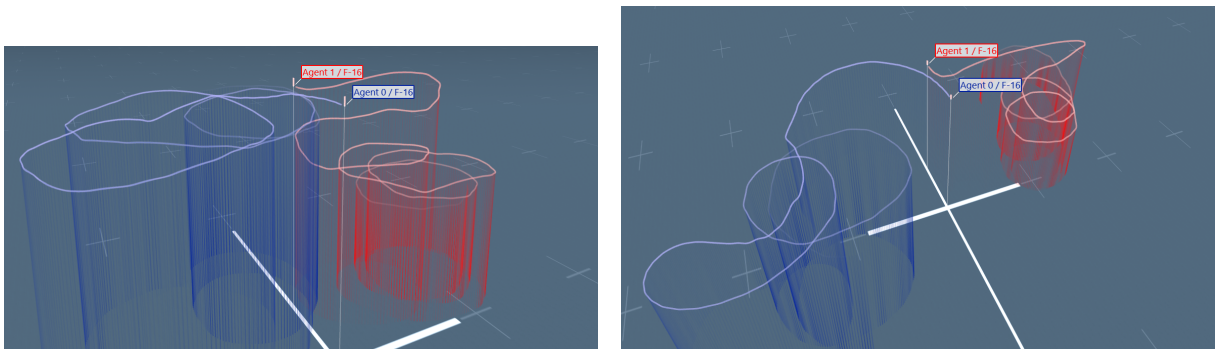
Algorithm	Reward (0) ↑	Reward (1) ↑	Shots Landed (0) ↑	Shots Landed (1) ↑
rPPO	0.2126 ± 0.6318	-0.2126 ± 0.6318	0.0 ± 0.0	0.0 ± 0.0

Values are reported as mean and standard deviation over N=10 runs. Arrows indicate preferred direction of each metric.

Table 6.16: JSBSim POMDP Zerosum Dogfight: evaluation scores.

Zerosum Evaluation Renders

Additional insight into how the two agents behave is provided in Fig. 6.57, which presents 3D renderings from two sample episodes. These visualizations trace the trajectories of both aircraft, offering a qualitative view of their interaction over the engagement.



(a) Evaluation episode 1 (b) Evaluation episode 2

Figure 6.57: JSBSim POMDP Zerosum Dogfight: evaluation episodes with rPPO.

Non-Zerosum Training Reward

The plot in Fig. 6.58 shows the total episode return recorded during training when Agent 0 or Agent 1 is being trained, respectively; therefore, each curve effectively reports the return achieved by the agent currently under training.

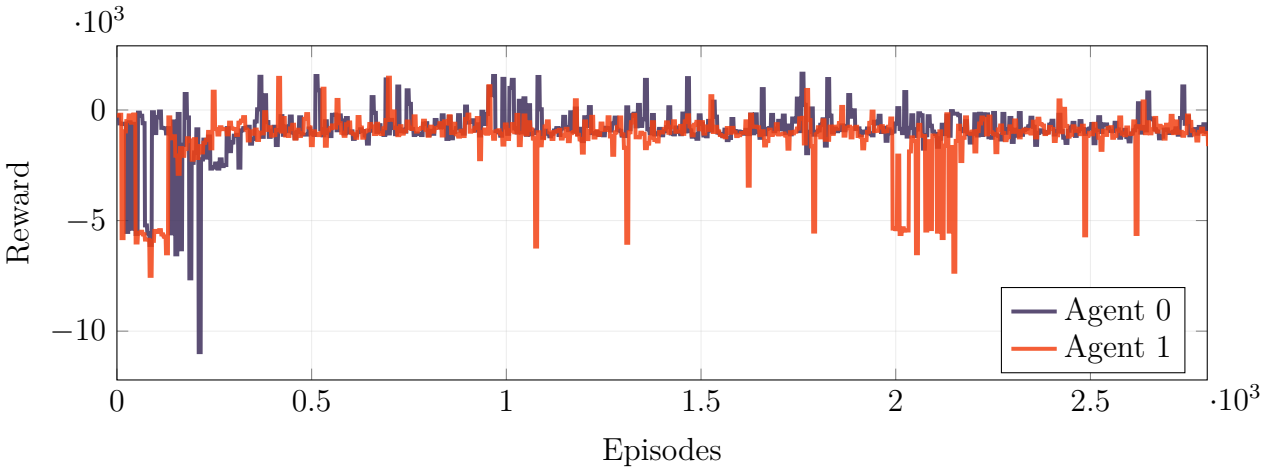
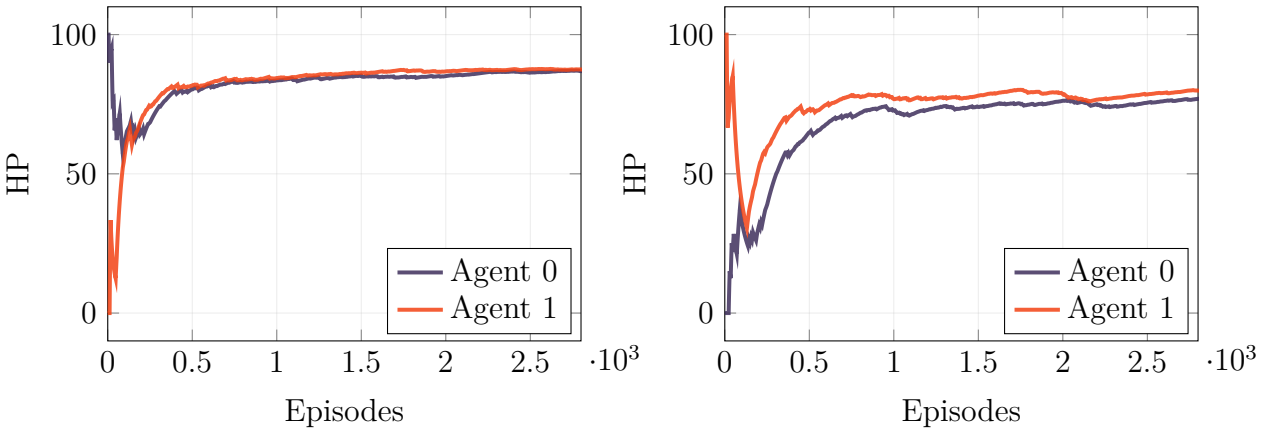


Figure 6.58: JSBSim POMDP Non-Zerosum Dogfight: total episode return

Additional Non-Zerosum Training Metrics

Figure 6.59 reports the evolution of the agents health points (HP) during training, explicitly separating the two training phases (training Agent 0 vs training Agent 1). In particular, Fig. 6.59a shows how Agent 0's HP fluctuate, while Fig. 6.59b reports how Agent 1's HP variate, with each subplot including both phases to highlight how the same health variable behaves when the corresponding agent is the learner versus when it is the opponent.



(a) Agent 0 HP

(b) Agent 1 HP

Figure 6.59: JSBSim POMDP Non-Zerosum Dogfight: HP trends for both agents across both training phases.

Non-Zerosum Evaluation Metrics

Evaluation scores over 10 independent runs are summarized in Table 6.17. Values are reported as mean ± standard deviation, and are provided separately for the two agents, denoted as (0) and (1).

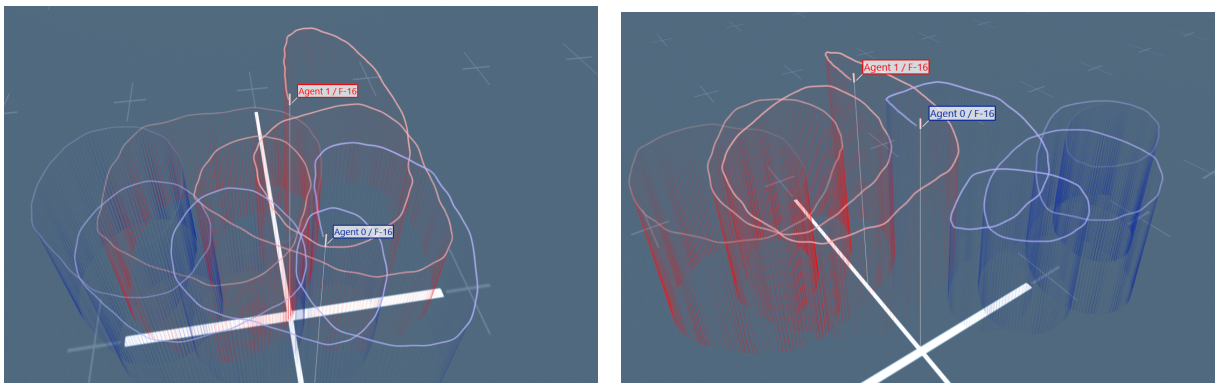
Algorithm	Reward (0) ↑	Reward (1) ↑	Shots Landed (0) ↑	Shots Landed (1) ↑
rPPO	-920.90 ± 329.20	-548.08 ± 598.07	2.50 ± 3.24	4.00 ± 3.74

Values are reported as mean and standard deviation over N=10 runs. Arrows indicate preferred direction of each metric.

Table 6.17: JSBSim POMDP Non-Zerosum Dogfight: evaluation scores.

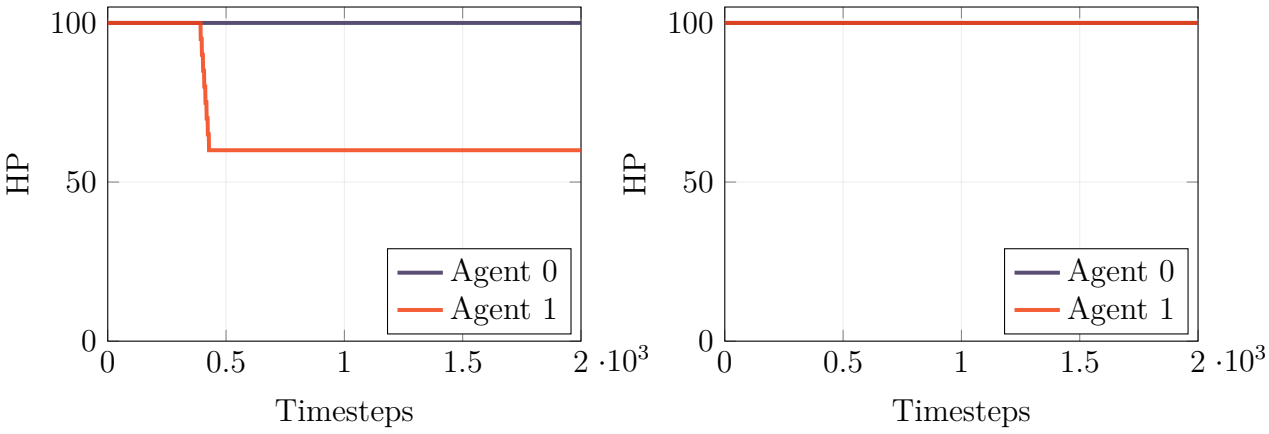
Non-Zerosum Evaluation Renders

A qualitative view of the agents' performance is given in Fig. 6.60, where 3D reconstructions from two representative episodes are shown. The plots track both agents' flight paths throughout the engagement, highlighting their interaction dynamics.



(a) Evaluation episode 1 (b) Evaluation episode 2

Figure 6.60: JSBSim POMDP Non-Zerosum Dogfight: evaluation episodes with rPPO.



(a) Evaluation episode 1 (b) Evaluation episode 2

Figure 6.61: JSBSim POMDP Non-Zerosum Dogfight: HP trajectories over timesteps for the two above evaluation episodes.

Discussion

These experiments extend the fully observable dogfight by introducing a partially observable formulation, where each agent receives only a restricted subset of information. The objective was not to redesign the environment, but to assess how limited observability affects learning dynamics, stability, and final performance, and to compare the results with the previous MDP setting.

Overall, the training curves suggest that the POMDP setting is harder to solve and less stable than the fully observable case. In the Zerosum configuration, the total episode return shows large oscillations early on and then quickly collapses around values close to zero, with only sporadic spikes. This pattern is consistent with agents failing to discover a reliable fighting strategy and converging toward behaviors where meaningful engagement is rare. The HP traces reinforce this interpretation: for most of training, both agents remain close to full health, with only occasional drops. This indicates that contacts and damage events are not happening consistently during training. In contrast, the Non-Zerosum configuration shows more interaction signals. The training return remains largely negative and noisy, suggesting that the agents experience frequent penalties or fail to control the fight efficiently. However, the HP trends fluctuate much more across episodes, meaning that damage is actually being exchanged more often. In other words, learning is still difficult and not clean, but the environment is at least producing repeated combat events, which makes policy improvement more plausible.

The evaluation metrics confirm a clear gap between the two reward settings under partial

observability. In the Zerosum case, the evaluation reports zero shots landed for both agents, together with returns close to zero. This strongly suggests that the learned policies do not reach the conditions needed to engage and shoot, or do not manage to do so reliably within an episode. In the Non-Zerosum case, evaluation results show non-zero shots landed for both agents, with high variability across runs. This matches what can be inferred from the evaluation renders: even if the plots do not explicitly show hits, the trajectories suggest that the agents sometimes manage to approach and interact, while in other episodes they fail to create sustained engagement. The high standard deviation is important here: performance is not consistent, and the behavior depends strongly on the specific episode conditions and initial geometry.

A direct explanation of why the POMDP is less effective than the MDP is that partial observability introduces state aliasing: different true combat situations may look similar from the limited observation vector. This makes control and decision-making harder, especially in a fast dynamic task like a dogfight where small geometric differences matter. Under a POMDP, an agent often needs memory to infer the opponents intent and relative motion trends, but the available signal may still be insufficient to reconstruct a reliable belief over the hidden state. A second factor is credit assignment. Shooting rewards are sparse, and under limited observability it becomes even harder to connect successful hits to the correct sequence of maneuvers. This can lead to policies that avoid risky engagement and instead converge toward safer flight regimes with minimal interaction, which is exactly what the Non-Zerosum setting appears to produce. Finally, the Non-Zerosum objective may amplify the difficulty. In a strict competitive formulation, any mistake is immediately punished relative to the opponent, and training alternation creates a non-stationary learning problem. If the observation is incomplete, this non-stationarity becomes more damaging, because each agent has weaker information to adapt to the opponents policy changes. The outcome can be a degenerate equilibrium where neither agent reliably engages, which is consistent with the absence of shots landed and the near-constant HP during training and evaluation.

The comparison between Non-Zerosum and Non-Zerosum in the POMDP setting suggests two different failure modes:

- **Zerosum POMDP**: the task may be too hard under the current reward and observation design, leading to policies with almost no interaction. This could mean the reward is not informative enough under partial observability, or that exploration is insufficient to reach meaningful combat states
- **Non-Zerosum POMDP**: interaction happens more often, but the behavior is

unstable and inconsistent. The agents sometimes manage to engage and score hits, but they do not converge to a robust fighting policy

In practice, the Non-Zerosum reward appears to be more learnable under partial observability, because it still produces episodes with contact and damage. However, it does not fully solve the stability and consistency issues introduced by the POMDP formulation. These results indicate that, for this environment, partial observability substantially increases difficulty and can prevent the emergence of reliable dogfighting behavior, especially in the Non-Zerosum case. If the goal is to make the POMDP setting competitive with the MDP baseline, several changes would likely be required. Examples include: richer but still realistic combat features, stronger curriculum learning toward engagement, longer training, or training schemes that reduce non-stationarity (for instance, more stable opponent sampling). Another direction is to better exploit memory and belief estimation, since a POMDP policy must compensate for missing information through temporal integration.

7 | Conclusions & Future Research

This chapter provides a comprehensive synthesis of the research contributions, experimental results, and remaining challenges presented throughout this thesis, which explores the application of Deep Reinforcement Learning (DRL) to autonomous aerial combat. Using a two-stage experimental pipeline, progressing from a simplified Toy Model to a high-fidelity JSBSim-based simulation, the study moves from controlled settings to realistic flight dynamics, systematically evaluating both algorithmic behavior and practical constraints.

By benchmarking modern RL agents across a spectrum of introductory and advanced tasks, the thesis highlights the key capabilities achieved so far as well as the main bottlenecks that still limit scalability and robustness, thereby motivating the next steps toward reliable and realistic autonomous dogfighting systems. The following sections retrace this progression, consolidate the main findings, and delineate promising directions for future research.

7.1. Critical Assessment of Results

This thesis shows that a major part of the ambiguity found in prior work can be reduced through a controlled, modular, and highly reproducible experimental setup. A two-stage methodology is adopted. First, simplified Gymnasium toy-model environments are used as controlled sandboxes to prototype observation spaces, action parameterizations, reward shaping, and episode design. This stage directly addresses common gaps in evaluation methodology and reproducibility: comparisons are performed under standardized metrics and consistent training protocols, reducing confounding factors that often blur algorithmic conclusions. Second, the study transitions to a higher-fidelity setting based on JSBSim flight dynamics, exposing the same learning pipeline to realistic 6-DOF dynamics, actuator limits, and safety-relevant constraints. This progression explicitly targets the "simulator-to-operational" gap by testing whether behaviors learned under simplified assumptions remain stable when scenario complexity and physical realism increase.

A systematic validation pipeline is demonstrated across four widely adopted Deep RL algorithms, with an emphasis on fair and repeatable comparisons. Training is performed with a fixed (static) seed whenever possible, to maximize reproducibility and to isolate algorithmic differences from stochastic variance. Under these controlled conditions, robust and consistent learning is obtained in toy-model environments when rewards and episode structure are carefully designed, confirming that reliable pursuit and tracking behaviors can be learned and benchmarked in a transparent way. At the same time, high-fidelity experiments highlight that generalization and stability remain open challenges in randomized and adversarial settings, where performance becomes more sensitive to initialization, scenario composition, and reward details. These findings align with the gaps identified in the literature on reward brittleness, opponent modeling non-stationarity, partial observability effects, and the limited transferability of results across simulators and scenario definitions.

The core contribution is a solid modular benchmarking framework that makes these conclusions testable, inspectable, and repeatable. The infrastructure is designed to keep concerns cleanly separated: environments, observation variants, reward functions, training configurations, evaluation protocols, and logging are implemented as interchangeable modules. This structure enables fast prototyping of new tasks and reward designs while preserving consistent interfaces and comparable metrics. Reproducibility is treated as a primary requirement rather than an afterthought: containerized execution, clean code baselines, structured experiment tracking, and systematic checkpointing support exact re-runs and transparent ablations. Advanced logging across training and evaluation further supports rigorous analysis, enabling detailed inspection of trajectories and emergent behaviors instead of relying only on aggregate win-rate style metrics.

The presented experiments provide clear reference points for method evaluation in autonomous aerial combat, while also making explicit where improvements are still required. The next section therefore focuses on margins for improvement, including more robust evaluation across multiple seeds and opponent sets, stronger constraint integration beyond external wrappers, and broader generalization testing across scenario families and sensing assumptions.

7.2. Next Steps & Future Research

This section outlines the most promising directions to extend and consolidate the work presented in this thesis. In particular, it discusses methodological improvements to training stability and sample efficiency, as well as architectural and environment-level changes

aimed at increasing robustness and scalability in realistic JSBSim-based air-combat tasks.

7.2.1. (Prioritized) Fictitious Self Play

A natural next step is to replace Frozen Opponent Self-Play (where the agent mainly plays against its latest snapshot) with a league-based training scheme inspired by Fictitious Self-Play (FSP). In this setting, the learner is trained against a mixture of opponents sampled from a pool of historical policies, rather than a single moving target, which helps reduce non-stationarity and mitigates collapse into narrow behaviors. In practice, a particularly promising variant is Prioritized Fictitious Self-Play (PFSP), where opponent sampling is biased toward useful opponents (typically those that are neither too weak nor too strong), improving learning signal and training stability. This mechanism has been successfully adopted in large-scale league training systems (e.g., AlphaStar-style leagues), where prioritization focuses training on the most informative matchups and reduces wasted experience on trivial games.

7.2.2. Warm-Starting

A recurring inefficiency in high-fidelity flight simulation is that a large fraction of training is spent on basic survival: trimming, stabilizing attitude, managing energy, and simply not crashing before any meaningful tactical or goal-directed behavior emerges. A straightforward improvement is to pretrain a flight-safety / flight-stability policy (e.g., for level flight, envelope protection, and recovery) and then use it as initialization for downstream tasks (follow-point, tag, dogfight). Warm-starting can reduce time-to-competence and lower variance across seeds because early training becomes less dominated by catastrophic exploration. Concretely, this can be implemented as:

- training a controller-focused policy (or set of skills) with dense shaping and strict termination penalties
- transferring weights to the task policy
- optionally freezing a subset of low-level layers during early fine-tuning to preserve stable flight primitives.

7.2.3. Expanding the Opponent Set

The current results already suggest that learned behaviors can become strongly coupled to the specific distribution of opponents encountered during training (including self-play snapshots). A clear direction is to introduce heterogeneous opponents with different poli-

cies and styles: scripted baselines (pure pursuit, energy fighter, defensive spiral), older checkpoints, adversaries trained with different reward shaping, and exploiters designed to probe weaknesses. League training literature emphasizes that maintaining a diverse population helps prevent brittle strategies and limits cyclical forgetting, because the learner must remain effective against a broader slice of the strategy space rather than a single co-evolving opponent.

7.2.4. Reduce Training Time

Training in JSBSim-based environments is computationally heavy: dynamics integration, reward/termination checks, and logging overhead all accumulate, especially when multi-agent rollouts are required. Future work should therefore include a systematic effort to cut the wall-clock cost through:

- profiling and optimizing the environment step (physics frequency, I/O, feature computation)
- selecting hyperparameters that improve sample efficiency (batch sizes, rollout length, learning rate schedules, entropy settings)
- revisiting network capacity (smaller feature extractors, fewer recurrent units, reduced history windows) to keep the policy expressive enough without unnecessary compute

This is particularly important when scaling to league-based training, where the number of matchups grows and evaluation must be more frequent to manage opponent sampling.

7.2.5. Addressing Symmetries

Another promising but non-trivial direction is to reduce redundant symmetries in the observation and action spaces. In 3D flight, many state configurations are symmetric under reflections/rotations, but choosing the right canonicalization is difficult: an overly aggressive symmetry reduction can destroy information needed for tactics (e.g., left/right ambiguity, sign conventions for yaw/roll coupling, or geometry-dependent engagement constraints). Still, well-designed reductions can shrink the effective state space, accelerate learning, and reduce overfitting to incidental coordinate conventions. Potential approaches include: canonical frames aligned with velocity/LOS, separating invariant scalars (ranges, closure, angles) from signed components, or using equivariant representations, while validating that the transformation preserves control-relevant distinctions.

7.2.6. Hierarchical Control

A final and high-impact research direction is to restructure the agent as a hierarchical policy. The high-level module would decide intent (attack / defend / disengage / reposition / intercept) at a slower timescale, while the low-level controller translates that intent into continuous control commands consistent with the aircraft dynamics and constraints. This decomposition matches the problem structure: aerial combat requires both long-horizon tactical reasoning and short-horizon stabilization/trajectory tracking. It also aligns naturally with warm-starting: the low-level layer can be pretrained as a robust flight controller (or a library of maneuver skills), while the high-level layer learns strategy via self-play/league training. The main challenges here are interface design (what exactly the high-level outputs), temporal abstraction (how often to switch modes), and credit assignment across levelsbut the payoff is potentially large in terms of stability, interpretability, and scalability to more realistic combat scenarios.

Bibliography

[1] Asobo Studio and Xbox Game Studios. Microsoft flight simulator. Computer software, 2020. URL https://www.flightsimulator.com/. Released for Windows on August 18, 2020. Accessed: 2026-01-23.

[2] K. J. Åström. Optimal control of markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, 10:174–205, 1965. URL https://api.semanticscholar.org/CorpusID:121222106.

[3] J. H. Bae, H. Jung, S. Kim, S. Kim, and Y.-D. Kim. Deep Reinforcement Learning-Based Air-to-Air Combat Maneuver Generation in a Realistic Environment. *IEEE Access*, 11:26427–26440, 2023. ISSN 2169-3536. doi: 10.1109/ACCESS.2023.3257849. URL https://ieeexplore.ieee.org/abstract/document/10072404. Conference Name: IEEE Access.

[4] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate, 2016. URL https://arxiv.org/abs/1409.0473.

[5] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(5):834–846, Sept. 1983. doi: 10.1109/TSMC.1983.6313077.

[6] R. Bellman. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences*, 38(8):716–719, 1952. doi: 10.1073/pnas.38.8.716. URL https://www.pnas.org/doi/abs/10.1073/pnas.38.8.716.

[7] R. Bellman. A markovian decision process. *Indiana University Mathematics Journal*, 6:679–684, 1957. URL https://api.semanticscholar.org/CorpusID:123329493.

[8] J. Berndt. Jsbsim: An open source flight dynamics model. *JSBSim Journal*, 08 2004. doi: 10.2514/6.2004-4923.

[9] L. Biewald. Experiment tracking with weights and biases, 2020. URL https://www.wandb.com/. Software available from wandb.com.

[10] J. Chai, W. Chen, Y. Zhu, Z.-X. Yao, and D. Zhao. A Hierarchical Deep Rein-

forcement Learning Framework for 6-DOF UCAV Air-to-Air Combat. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 53(9):5417–5429, Sept. 2023. ISSN 2168-2232. doi: 10.1109/TSMC.2023.3270444. URL https://ieeexplore.ieee.org/abstract/document/10120732. Conference Name: IEEE Transactions on Systems, Man, and Cybernetics: Systems.

[11] C. Chen, L. Mo, M. Lv, D. Lin, T. Song, and J. Cao. Enhanced missile hit probability actor-critic algorithm for autonomous decision-making in air-to-air confrontation. *Aerospace Science and Technology*, 151:109285, Aug. 2024. ISSN 1270-9638. doi: 10.1016/j.ast.2024.109285. URL https://www.sciencedirect.com/science/article/pii/S1270963824004164.

[12] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014. URL https://arxiv.org/abs/1406.1078.

[13] S. G. Clarke and I. Hwang. Deep Reinforcement Learning Control for Aerobatic Maneuvering of Agile Fixed-Wing Aircraft. In *AIAA Scitech 2020 Forum*, Orlando, FL, Jan. 2020. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-595-1. doi: 10.2514/6.2020-0136. URL https://arc.aiaa.org/doi/10.2514/6.2020-0136.

[14] A. De Marco, P. M. DOnza, and S. Manfredi. A deep reinforcement learning control approach for high-performance aircraft. *Nonlinear Dynamics*, 111(18):17037–17077, Sept. 2023. ISSN 1573-269X. doi: 10.1007/s11071-023-08725-y. URL https://doi.org/10.1007/s11071-023-08725-y.

[15] Y. Dong, S. He, Y. Zhao, J. Ai, and C. Wang. Development and Evaluation of Transformer-Based Basic Fighter Maneuver Decision-Support Scheme for Piloting During Within-Visual-Range Air Combat. *Aerospace*, 12(2):73, Feb. 2025. ISSN 2226-4310. doi: 10.3390/aerospace12020073. URL https://www.mdpi.com/2226-4310/12/2/73. Number: 2 Publisher: Multidisciplinary Digital Publishing Institute.

[16] Eagle Dynamics. Digital combat simulator world (dcs world). Computer software, 2012. URL https://www.digitalcombatsimulator.com/en/products/world/. Open beta released May 2012. Accessed: 2026-01-23.

[17] J. L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. doi: https://doi.org/10.1207/s15516709cog1402_1. URL https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog1402_1.

[18] Z. Fan, Y. Xu, Y. Kang, and D. Luo. Air Combat Maneuver Decision Method Based

on A3C Deep Reinforcement Learning. *Machines*, 10(11):1033, Nov. 2022. ISSN 2075-1702. doi: 10.3390/machines10111033. URL https://www.mdpi.com/2075-1702/10/11/1033. Number: 11 Publisher: Multidisciplinary Digital Publishing Institute.

[19] F. Foundation. Gymnasium: A standard api for reinforcement learning and robotics research, 2023. URL https://gymnasium.farama.org/.

[20] S. Fujimoto, H. v. Hoof, and D. Meger. Addressing Function Approximation Error in Actor-Critic Methods, Oct. 2018. URL http://arxiv.org/abs/1802.09477. arXiv:1802.09477 [cs].

[21] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer Series in Computational Mathematics. Springer, 2 edition, 1993. doi: 10.1007/978-3-540-78862-1.

[22] H. Han, J. Cheng, and M. Lv. Interpretable DRL-Based Maneuver Decision of UCAV Dogfight. In *2024 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 109–114, Oct. 2024. doi: 10.1109/SMC54092.2024.10831270. URL https://ieeexplore.ieee.org/abstract/document/10831270.

[23] J. Hansen, A. Louette, P. Leroy, and D. Ernst. Autonomous drone combat: A multi-agent reinforcement learning approach. ORBi (University of Liège Institutional Repository), Sept. 2025. URL https://orbi.uliege.be/handle/2268/335358. Eprint first made available on ORBi (University of Liège). Publication date: 10 September 2025.

[24] J. Heinrich and D. Silver. Deep Reinforcement Learning from Self-Play in Imperfect-Information Games, June 2016. URL http://arxiv.org/abs/1603.01121. arXiv:1603.01121 [cs].

[25] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9 (8):17351780, Nov. 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

[26] R. S. Inc. Tacview: Flight simulation and data analysis software, 2006. URL https://www.tacview.net.

[27] S. Jiang, Y. Ge, X. Yang, W. Yang, and H. Cui. UAV Control Method Combining Reptile Meta-Reinforcement Learning and Generative Adversarial Imitation Learning. *Future Internet*, 16(3):105, Mar. 2024. ISSN 1999-5903. doi: 10.3390/fi16030105. URL https://www.mdpi.com/1999-5903/16/3/105. Number: 3 Publisher: Multidisciplinary Digital Publishing Institute.

[28] X. Jing, F. Cong, J. Huang, C. Tian, and Z. Su. Autonomous Maneuvering Decision-Making Algorithm for Unmanned Aerial Vehicles Based on Node Clustering and Deep Deterministic Policy Gradient. | EBSCOhost, Dec. 2024. URL https://openurl.ebsco.com/contentitem/doi:10.3390%2Faerospace11121055?sid=ebsco:plink:crawler&id=ebsco:doi:10.3390%2Faerospace11121055. ISSN: 2226-4310 Issue: 12 Pages: 1055 Volume: 11.

[29] W.-r. Kong, D.-y. Zhou, Y.-j. Du, Y. Zhou, and Y.-y. Zhao. Hierarchical multi-agent reinforcement learning for multi-aircraft close-range air combat. *IET Control Theory & Applications*, 17(13):1840–1862, 2023. ISSN 1751-8652. doi: 10.1049/cth2.12413. URL https://onlinelibrary.wiley.com/doi/abs/10.1049/cth2.12413. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1049/cth2.12413.

[30] J. Källström and F. Heintz. Agent Coordination in Air Combat Simulation using Multi-Agent Deep Reinforcement Learning. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2157–2164, Oct. 2020. doi: 10.1109/SMC42975.2020.9283492. URL https://ieeexplore.ieee.org/document/9283492. ISSN: 2577-1655.

[31] Laminar Research. X-plane 12. Computer software, 2022. URL https://www.x-plane.com/. Early Access began September 5, 2022. Accessed: 2026-01-23.

[32] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989. URL https://api.semanticscholar.org/CorpusID:41312633.

[33] L. Li, X. Zhang, C. Qian, M. Zhao, and R. Wang. Cross coordination of behavior clone and reinforcement learning for autonomous within-visual-range air combat. *Neurocomputing*, 584:127591, June 2024. ISSN 0925-2312. doi: 10.1016/j.neucom.2024.127591. URL https://www.sciencedirect.com/science/article/pii/S092523122400362X.

[34] S. Li, R. Zuo, P. Liu, and Y. Zhao. An Imitative Reinforcement Learning Framework for Autonomous Dogfight, June 2024. URL http://arxiv.org/abs/2406.11562. arXiv:2406.11562 [cs].

[35] Y.-f. Li, J.-p. Shi, W. Jiang, W.-g. Zhang, and Y.-x. Lyu. Autonomous maneuver decision-making for a UCAV in short-range aerial combat based on an MS-DDQN algorithm. *Defence Technology*, 18(9):1697–1714, Sept. 2022. ISSN 2214-9147.

doi: 10.1016/j.dt.2021.09.014. URL https://www.sciencedirect.com/science/article/pii/S2214914721001781.

[36] L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3):293–321, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992699. URL https://doi.org/10.1007/BF00992699.

[37] Lockheed Martin Corporation. *F-16C/D Flight Manual: Blocks 50 and 52+*. Hellenic Air Force, 2003. URL https://info.publicintelligence.net/HAF-F16.pdf.

[38] S. Marulli. U.s. air force f-16 fighting falcon in flight (tail code av, serial 88-0532). Photograph, MarulliPhoto, 2026. URL https://www.marulliphoto.com/wp-content/uploads/SM302094-DxO_DeepPRIME-XD2s-1-copy.jpg. Online image.

[39] J. S. McGrew, J. P. How, B. Williams, and N. Roy. Air-Combat Strategy Using Approximate Dynamic Programming. *Journal of Guidance, Control, and Dynamics*, 33(5):1641–1654, Sept. 2010. ISSN 0731-5090. doi: 10.2514/1.46815. URL https://arc.aiaa.org/doi/10.2514/1.46815. Publisher: American Institute of Aeronautics and Astronautics.

[40] J. Mei, G. Li, and H. Huang. Deep Reinforcement-Learning-Based Air-Combat-Maneuver Generation Framework. *Mathematics*, 12(19):3020, Jan. 2024. ISSN 2227-7390. doi: 10.3390/math12193020. URL https://www.mdpi.com/2227-7390/12/19/3020. Number: 19 Publisher: Multidisciplinary Digital Publishing Institute.

[41] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning, 2013. URL https://arxiv.org/abs/1312.5602.

[42] A. Müller. A tour d'horizon of de casteljau's work. *Computer Aided Geometric Design*, 108:102251, 2024. doi: 10.1016/j.cagd.2023.102251.

[43] M. Pleines, M. Pallasch, F. Zimmer, and M. Preuss. Generalization, Mayhems and Limits in Recurrent Proximal Policy Optimization, May 2022. URL http://arxiv.org/abs/2205.11104. arXiv:2205.11104 [cs].

[44] A. P. Pope, J. S. Ide, D. Miovi, H. Diaz, D. Rosenbluth, L. Ritholtz, J. C. Twedt, T. T. Walker, K. Alcedo, and D. Javorsek. Hierarchical Reinforcement Learning for Air-to-Air Combat. In *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 275–284, June 2021. doi: 10.1109/ICUAS51884.2021.9476700. URL https://ieeexplore.ieee.org/abstract/document/9476700. ISSN: 2575-7296.

[45] D. Precup, R. S. Sutton, S. Singh, T. Labs, P. Avenue, and F. Park. Eligibility Traces for Off-Policy Policy Evaluation. *University of Massachussets*, 2000.

[46] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann. Stable baselines3 contrib. `https://github.com/DLR-RM/rl-baselines3-zoo`, 2019.

[47] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL `http://jmlr.org/papers/v22/20-1364.html`.

[48] D. J. Richter, R. A. Calix, and K. Kim. A Review of Reinforcement Learning for Fixed-Wing Aircraft Control Tasks. *IEEE Access*, 12:103026–103048, 2024. ISSN 2169-3536. doi: 10.1109/ACCESS.2024.3433540. URL `https://ieeexplore.ieee.org/abstract/document/10609369`. Conference Name: IEEE Access.

[49] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958. URL `https://api.semanticscholar.org/CorpusID:12781225`.

[50] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. doi: 10.1038/323533a0.

[51] A. L. Samuel. Some Studies in Machine Learning Using the Gameof Checkers. II-Recent Progress. *MACHINE LEARNING*, 1956.

[52] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2015. doi: 10.48550. URL `https://arxiv.org/abs/1502.05477`.

[53] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms, Aug. 2017. URL `http://arxiv.org/abs/1707.06347`. arXiv:1707.06347 [cs].

[54] A. Selmonaj, O. Szehr, G. Del Rio, A. Antonucci, A. Schneider, and M. Rüegsegger. Hierarchical Multi-Agent Reinforcement Learning for Air Combat Maneuvering. In *2023 International Conference on Machine Learning and Applications (ICMLA)*, pages 1031–1038, Dec. 2023. doi: 10.1109/ICMLA58977.2023.00153. URL `https://ieeexplore.ieee.org/document/10459738`. ISSN: 1946-0759.

[55] H. Seong and D. H. Shim. TempFuser: Learning Agile, Tactical, and Acrobatic Flight Maneuvers Using a Long Short-Term Temporal Fusion Transformer. *IEEE Robotics and Automation Letters*, 9(12):10803–10810, Dec. 2024. ISSN 2377-3766.

doi: 10.1109/LRA.2024.3477092. URL https://ieeexplore.ieee.org/abstract/document/10711233. Conference Name: IEEE Robotics and Automation Letters.

[56] R. L. Shaw. *Fighter Combat: tactics and maneuvering*. Naval Institute Press, 1985.

[57] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, Oct. 2017. ISSN 1476-4687. doi: 10.1038/nature24270. URL https://doi.org/10.1038/nature24270.

[58] Stable Baselines3. Vectorized environments — stable baselines3 documentation (vecframestack). https://stable-baselines3.readthedocs.io/en/master/guide/vec_envs.html#vecframestack, 2026.

[59] Stable-Baselines3 Contributors. Vectorized environments — stable baselines3 documentation, 2026. URL https://stable-baselines3.readthedocs.io/en/master/guide/vec_envs.html#subprocvecenv. Section: SubprocVecEnv.

[60] Stable-Baselines3 Contributors. Vectorized environments — stable baselines3 2.8.0a3 documentation, 2026. URL https://stable-baselines3.readthedocs.io/en/master/guide/vec_envs.html#vecnormalize. Section: VecNormalize.

[61] L. Sun, H. Qiu, Y. Wang, and C. Yan. Autonomous UAV Maneuvering Decisions by Refining Opponent Strategies. *IEEE Transactions on Aerospace and Electronic Systems*, 60(3):3454–3467, June 2024. ISSN 1557-9603. doi: 10.1109/TAES.2024.3362765. URL https://ieeexplore.ieee.org/abstract/document/10423189. Conference Name: IEEE Transactions on Aerospace and Electronic Systems.

[62] Z. Sun, H. Piao, Z. Yang, Y. Zhao, G. Zhan, D. Zhou, G. Meng, H. Chen, X. Chen, B. Qu, and Y. Lu. Multi-agent hierarchical policy gradient for Air Combat Tactics emergence via self-play. *Engineering Applications of Artificial Intelligence*, 98:104112, Feb. 2021. ISSN 0952-1976. doi: 10.1016/j.engappai.2020.104112. URL https://www.sciencedirect.com/science/article/pii/S0952197620303547.

[63] R. S. Sutton and A. G. Barto. Reinforcement Learning: An Introduction. *The MIT Press*, 1996.

[64] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018. ISBN 9780262039246.

[65] J. D. Team. Jsbsim - open source flight dynamics model, 2025. URL https://github.com/JSBSim-Team/jsbsim.

[66] J. Terry, B. Black, N. Grammel, M. Jayakumar, A. Hari, R. Sullivan, L. S. Santos, C. Dieffendahl, C. Horsch, R. Perez-Vicente, et al. Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 34: 15032–15043, 2021.

[67] P. A. Tuomas Haarnoja, Aurick Zhou and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv:1801.01290*, 2018. doi: 10.48550/arXiv.1801.01290. URL https://doi.org/10.48550/arXiv.1801.01290.

[68] N. K. Ure and G. Inalhan. Autonomous Control of Unmanned Combat Air Vehicles: Design of a Multimodal Control and Flight Planning Framework for Agile Maneuvering. *IEEE Control Systems Magazine*, 32(5):74–95, Oct. 2012. ISSN 1941-000X. doi: 10.1109/MCS.2012.2205532. URL https://ieeexplore.ieee.org/abstract/document/6302315. Conference Name: IEEE Control Systems Magazine.

[69] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

[70] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, Nov. 2019. ISSN 1476-4687. doi: 10.1038/s41586-019-1724-z. URL https://doi.org/10.1038/s41586-019-1724-z.

[71] VNFAWING. Introduction to the geometry of air combat. https://vnfawing.com, 2023. URL https://vnfawing.com.

[72] B. Wang, X. Gao, and T. Xie. An evolutionary multi-agent reinforcement learning algorithm for multi-UAV air combat. *Knowledge-Based Systems*, 299:112000, Sept. 2024. ISSN 0950-7051. doi: 10.1016/j.knosys.2024.112000. URL https://www.sciencedirect.com/science/article/pii/S0950705124006348.

[73] L. Wang, S. Zheng, H. Piao, C. Lu, T. Yue, and H. Liu. Tube-based robust rein-

forcement learning for autonomous maneuver decision for UCAVs. *Chinese Journal of Aeronautics*, 37(7):391–405, July 2024. ISSN 1000-9361. doi: 10.1016/j.cja.2024.03.025. URL https://www.sciencedirect.com/science/article/pii/S1000936124000992.

[74] X. Wang, Z. Yang, S. Chai, J. Huang, Y. He, and D. Zhou. Tactical intent-driven autonomous air combat behavior generation method. *Complex & Intelligent Systems*, 11(1):65, Dec. 2024. ISSN 2198-6053. doi: 10.1007/s40747-024-01685-9. URL https://doi.org/10.1007/s40747-024-01685-9.

[75] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992698. URL https://doi.org/10.1007/BF00992698.

[76] Wikipedia contributors. Tag (game) — Wikipedia, the free encyclopedia, 2025. URL https://en.wikipedia.org/w/index.php?title=Tag_(game)&oldid=1328692735.

[77] R. Williams and J. Peng. Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3:241–, 09 1991. doi: 10.1080/09540099108946587.

[78] R. J. Williams and D. Zipser. A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Computation*, 1(2):270–280, June 1989. ISSN 0899-7667, 1530-888X. doi: 10.1162/neco.1989.1.2.270. URL https://direct.mit.edu/neco/article/1/2/270-280/5490.

[79] J. Yang, X. Yang, and T. Yu. Multi-Unmanned Aerial Vehicle Confrontation in Intelligent Air Combat: A Multi-Agent Deep Reinforcement Learning Approach. *Drones*, 8(8):382, Aug. 2024. ISSN 2504-446X. doi: 10.3390/drones8080382. URL https://www.mdpi.com/2504-446X/8/8/382. Number: 8 Publisher: Multidisciplinary Digital Publishing Institute.

[80] Z. Yang, L. Li, S. Chai, J. Huang, H. Piao, and D. Zhou. Autonomous evasive maneuver method for unmanned combat aerial vehicle in air combat with multiple tactical requirements. *Acta Aeronautica et Astronautica Sinica*, 45(20), Oct. 2024. ISSN 1000-6893. doi: 10.7527/S1000-6893.2024.30629. URL https://www.sciopen.com/article/10.7527/S1000-6893.2024.30629. Publisher: Editorial Office of Acta Aeronautica et Astronautica Sinica.

[81] J. Yoo, H. Seong, D. H. Shim, J. H. Bae, and Y.-D. Kim. Deep Reinforcement Learning-based Intelligent Agent for Autonomous Air Combat. In *2022 IEEE/AIAA*

41st Digital Avionics Systems Conference (DASC), pages 1–9, Sept. 2022. doi: 10.1109/DASC55683.2022.9925811. URL https://ieeexplore.ieee.org/abstract/document/9925811. ISSN: 2155-7209.

[82] H. Zhang, Y. Wei, H. Zhou, and C. Huang. Maneuver Decision-Making for Autonomous Air Combat Based on FRE-PPO. *Applied Sciences*, 12(20):10230, Jan. 2022. ISSN 2076-3417. doi: 10.3390/app122010230. URL https://www.mdpi.com/2076-3417/12/20/10230. Number: 20 Publisher: Multidisciplinary Digital Publishing Institute.

[83] P. Zhang, W. Dong, M. Cai, D. Li, and X. Zhang. Learning and Fast Adaptation for Air Combat Decision with Improved Deep Meta-reinforcement Learning. *International Journal of Aeronautical and Space Sciences*, Sept. 2024. ISSN 2093-2480. doi: 10.1007/s42405-024-00803-8. URL https://doi.org/10.1007/s42405-024-00803-8.

[84] S. Zhang, X. Du, J. Xiao, J. Huang, and K. He. Reinforcement Learning Control for 6 DOF Flight of Fixed-Wing Aircraft. In *2021 33rd Chinese Control and Decision Conference (CCDC)*, pages 5454–5460, May 2021. doi: 10.1109/CCDC52312.2021.9602605. URL https://ieeexplore.ieee.org/abstract/document/9602605. ISSN: 1948-9447.

[85] X. Zhang, W. Dong, P. Zhang, and D. Li. Research on the Reward Design Method for Deep Reinforcement Learning in WVR Air Combat. *IEEE Access*, 12:182693–182707, 2024. ISSN 2169-3536. doi: 10.1109/ACCESS.2024.3510723. URL https://ieeexplore.ieee.org/abstract/document/10777032. Conference Name: IEEE Access.

[86] P. Zhao and Y. Liu. Physics Informed Deep Reinforcement Learning for Aircraft Conflict Resolution. *IEEE Transactions on Intelligent Transportation Systems*, 23 (7):8288–8301, July 2022. ISSN 1558-0016. doi: 10.1109/TITS.2021.3077572. URL https://ieeexplore.ieee.org/abstract/document/9430767. Conference Name: IEEE Transactions on Intelligent Transportation Systems.

List of Figures

List of Tables