Heterogeneous-Agent Reinforcement Learning for Light Aircraft Game

CHENG Feilin [*1]

Abstract

Traditional Multi-Agent Reinforcement Learning (MARL) relies on parameter sharing among homogeneous agents, limiting applicability to real-world scenarios with heterogeneous agents. Heterogeneous-Agent Reinforcement Learning (HARL) addresses this limitation by coordinating agents with distinct capabilities without parameter sharing. This paper evaluates two prominent HARL algorithms HAPPO and HASAC in the Light Aircraft Game (LAG) environment, providing insights into algorithm performance in complex multi-agent tasks. And we propose and implement HAPPO-Optimized to enhance training stability and convergence speed.

1. Introduction

1.1. Baseline Algorithms

We introduce two baseline HARL algorithms that form the foundation of our comparative study.

1.1.1. HAPPO

Heterogeneous-Agent Proximal Policy Optimization (HAPPO) extends PPO for multi-agent settings with heterogeneous agents.

Core Idea and Evolution from MAPPO HAPPO's key innovation is the sequential update scheme: agents are updated one-by-one rather than simultaneously as in MAPPO. This enables heterogeneous agents without parameter sharing while maintaining coordination through a centralized critic. HAPPO introduces factor-based importance weighting in the policy loss to account for agent heterogeneity.

[1]2300010850, Peking University. Correspondence to: CHENG Feilin <cflyuke@gmail.com>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

1.1.2. HASAC

Heterogeneous-Agent Soft Actor-Critic (HASAC) extends SAC for multi-agent settings with maximum entropy reinforcement learning.

Core Idea and Evolution from SAC HASAC adapts SAC for heterogeneous multi-agent scenarios using centralized training with decentralized execution. Each agent maintains its own actor network while sharing a centralized critic. The algorithm supports both continuous and discrete action spaces and maintains off-policy learning through experience replay.

Role of the Maximum Entropy Principle HASAC maximizes both expected return and policy entropy:

$$J(\pi) = \mathbb{E}[\sum_{t=0}^{T} \gamma^t (r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t)))]$$

The temperature parameter α controls the exploration-exploitation trade-off and can be automatically tuned to maintain target entropy levels.

1.2. LAG Experimental Environment

We use the Light Aircraft Game (LAG) environment, a realistic aerial combat simulation built on JSBSim flight dynamics. Our experiments focus on a **2v2 cooperative scenario without missiles**.

1.3. Main Contributions

The primary contributions of this work are:

- We conduct a comprehensive comparative analysis of HAPPO and HASAC algorithms in the LAG 2v2 cooperative aerial combat environment, identifying their relative strengths and limitations.
- We propose and implement HAPPO-Optimized, an enhanced version of HAPPO with adaptive parameter adjustment, KL penalty regularization, dual clipping mechanisms, and advanced learning rate scheduling. These improvements significantly enhance training stability and convergence speed.

2. Baseline Algorithm Performance Analysis

This chapter presents a comprehensive analysis of the experimental results for HAPPO and HASAC baseline algorithms in the LAG environment, including experimental setup, training curve comparisons, and model evaluation results.

2.1. Experimental Setup

Configuration Parameters The detailed parameter configurations for both algorithms can be found in the result folder.

Environment Configuration The experimental environment is configured as **2v2 noWeapon VsBaseline**, where two intelligent agents cooperate against two baseline AI opponents without weapon systems, focusing on maneuverability and tactical positioning capabilities.

Hardware Configuration All experiments were conducted on a single RTX 4090 GPU with 24GB VRAM. Each experiment ran for approximately 10 hours with a total of 10,000,000 training steps.

2.2. Experimental Results and Analysis

2.2.1. Training Result Analysis

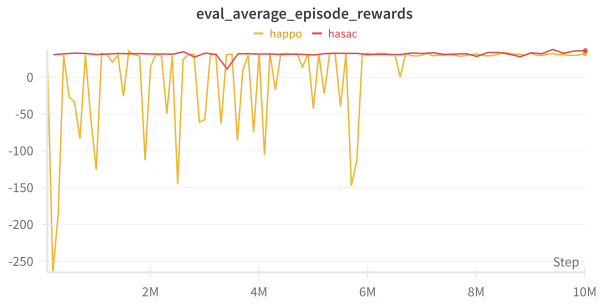

Figure 1. training curve of happo and hasac

- **Training Instability in HAPPO:**

  The significant training instability observed in HAPPO can be attributed to several fundamental algorithmic characteristics evident in its implementation. First, HAPPO employs a **sequential agent update mechanism** where agents are updated one by one, with each subsequent agent's update depending on the policy changes of previously updated agents. This is implemented through the `factor` variable in the `OnPolicyHARunner`, which accumulates importance sampling weights:

$$\text{factor} = \text{factor} \times \frac{\pi_{\text{new}}(a|s)}{\pi_{\text{old}}(a|s)}$$

  This sequential dependency creates a **cascading effect** where early agents' policy changes can dramatically affect the learning landscape for later agents, leading to high variance in gradient estimates. Additionally, HAPPO uses **PPO-style clipped surrogate objectives** with importance sampling ratios that can become unstable when the policy changes significantly between updates. The clipping mechanism, while designed to prevent large policy updates, can create discontinuous gradients that contribute to training oscillations. The on-policy nature of HAPPO also means that each policy update invalidates the collected experience, requiring frequent environment interactions and making the algorithm sensitive to hyperparameter choices such as the clip parameter ϵ and the number of PPO epochs.

- **Slower Convergence of HAPPO:**

  HAPPO's convergence is fundamentally limited by its **on-policy learning paradigm**, requiring fresh experience collection after each policy update. In contrast, HASAC benefits from **off-policy learning** with experience replay, allowing multiple gradient updates per environment step. HASAC employs **soft Q-learning** with twin critics, replacing the standard Bellman equation:

$$Q^*(s,a) = r + \gamma \mathbb{E}_{s' \sim p}[\max_a Q^*(s',a')]$$

  with the entropy-regularized objective:

$$Q^*(s,a) = r + \gamma \mathbb{E}_{s' \sim p, a' \sim \pi}[Q^*(s',a') - \alpha \log \pi(a'|s')]$$

  This formulation provides more stable value estimates and reduces the variance that plagues on-policy methods like HAPPO.

- **Superior Final Performance of HASAC:**

  HASAC achieves better final performance through **maximum entropy reinforcement learning**, explicitly maximizing both expected return and policy entropy:

$$J(\pi) = \mathbb{E}_{\tau \sim \pi}[\sum_{t=0}^{T} r(s_t, a_t) + \alpha H(\pi(\cdot|s_t))]$$

  where $H(\pi(\cdot|s_t)) = -\mathbb{E}_{a \sim \pi}[\log \pi(a|s_t)]$ is the policy entropy. The automatic temperature tuning mechanism dynamically adjusts the exploration-exploitation trade-off, while the twin critic architecture reduces overestimation bias. HASAC's off-policy nature enables continuous learning from diverse replay buffer experiences, whereas HAPPO's on-policy constraint limits its ability to improve on past experiences.

2.2.2. HYPERPARAMETER SENSITIVITY ANALYSIS

Figure 2. hyperparameter fine-tuning training result of happo

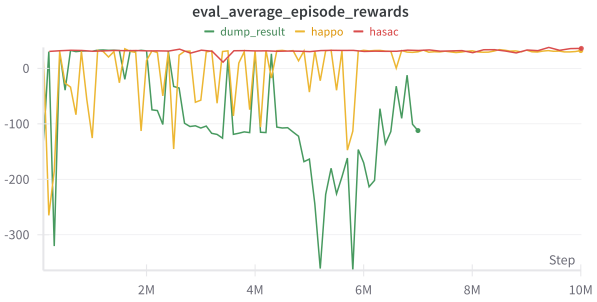

Figure 3. hyperparameter fine-tuning eval result of happo

The extreme volatility observed in the dump_result curve (oscillating between +30 and -360 rewards) demonstrates HAPPO's critical **hyperparameter sensitivity**, which motivated the development of HAPPO_Optimized.

Key issues include: (1) **Fixed clipping parameter** as policies evolve, cause either insufficient learning or policy collapse; (2) **Static entropy coefficient** that fails to maintain proper exploration-exploitation balance; (3) **Cascading instability** where the sequential update factor:

$$\text{factor} = \prod_{i=1}^{t} \frac{\pi_{\text{new}}^{(i)}(a|s)}{\pi_{\text{old}}^{(i)}(a|s)}$$

amplifies hyperparameter mismatches across agents.

2.3. Trained Model Evaluation

We evaluate the trained models using the `lag_render.py` script, which loads trained models and performs visualization demonstrations in the LAG environment. The evaluation generates two types of outputs:

- **Actual Reward Metrics**: Real performance indicators obtained through environment interaction
- **ACMI Files**: Professional format files for creating flight trajectory animations, available as happo.acmi and hasac.acmi in the render directory

The rendering tests reveal that both algorithms successfully train agents capable of coordinated tactical behaviors and execution of complex aerial maneuvers.

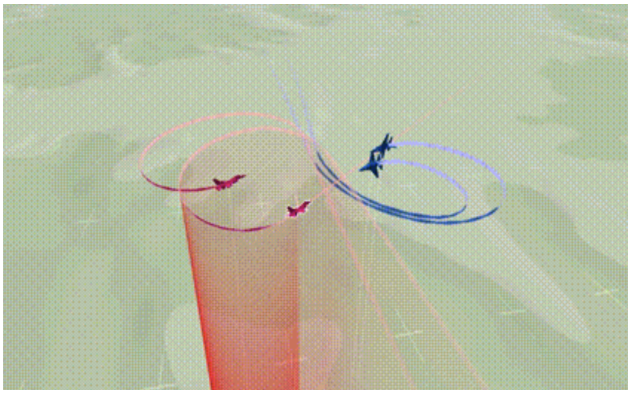

Figure 4. render video of acmi file

3. HAPPO Optimized

3.1. Motivation and Starting Point

As demonstrated in the previous analysis, HAPPO suffers from significant **training instability** and **slow convergence** compared to HASAC. The sequential agent update mechanism creates cascading effects where policy changes propagate through importance sampling factors, leading to high variance in gradient estimates and oscillatory behavior. Additionally, HAPPO's on-policy nature requires fresh experience collection after each update, severely limiting sample efficiency compared to HASAC's off-policy learning with experience replay.

The extreme volatility observed in the dump_result experiments (rewards oscillating between +30 and -360) further demonstrated HAPPO's critical **hyperparameter brittleness**. HAPPO_Optimized addresses these fundamental issues:

- **cascading instability** from sequential updates
- **fixed hyperparameter rigidity** preventing adaptation to evolving dynamics
- **lack of learning progress monitoring** allowing continued updates despite policy divergence

3.2. Key Optimizations and Their Rationale

3.2.1. Adaptive Clipping Mechanism

The most critical optimization addresses HAPPO's fixed clipping parameter limitation through a **KL divergence-based adaptive clipping strategy**. The original HAPPO uses a static clipping parameter $\epsilon = 0.05$, which becomes inappropriate as training progresses. HAPPO_Optimized implements dynamic adjustment:

$$\epsilon_t = \begin{cases} \max(\epsilon_{t-1} \times 0.8, \epsilon_{\min}) & \text{if KL} > 2\text{KL}_{\text{target}} \\ \min(\epsilon_{t-1} \times 1.2, \epsilon_{\max}) & \text{if KL} < 0.5\text{KL}_{\text{target}} \\ \epsilon_{t-1} & \text{otherwise} \end{cases}$$

This mechanism ensures that when policies change too rapidly (high KL divergence), the clipping becomes more conservative to prevent instability. Conversely, when policies change too slowly (low KL divergence), clipping is relaxed to allow faster learning. The rationale is that KL divergence provides a principled measure of policy change magnitude, enabling automatic adaptation to the current learning phase.

3.2.2. Dynamic Entropy Scheduling

HAPPO_Optimized introduces **adaptive entropy coefficient adjustment** to maintain optimal exploration-exploitation balance throughout training. The entropy coefficient λ_{ent} is dynamically updated based on current policy entropy relative to a target entropy:

$$\lambda_{\text{ent},t} = \begin{cases} \min(\lambda_{t-1} \times 1.1, \lambda_{\max}) & \text{if } H(\pi_t) < 0.5H_{\text{target}} \\ \max(\lambda_{t-1} \times 0.9, \lambda_{\min}) & \text{if } H(\pi_t) > 2.0H_{\text{target}} \\ \lambda_{t-1} & \text{otherwise} \end{cases}$$

where H_{target} is computed as a heuristic based on action space dimensionality. This addresses the original HAPPO's inability to maintain proper exploration as policies become more deterministic during training, preventing premature convergence to suboptimal policies.

3.2.3. Early Stopping and KL Penalty Regularization

To prevent the cascading instability that plagued the original HAPPO, HAPPO_Optimized implements **early stopping mechanisms** based on KL divergence monitoring:

$$\text{Stop if: } \frac{1}{B}\sum_{b=1}^{B} \text{KL}(\pi_{\text{old}}||\pi_{\text{new}})_b > 1.5\text{KL}_{\text{target}}$$

Additionally, a **KL penalty term** is added to the policy loss:

$$L_{\text{policy}} = L_{\text{PPO}} + \beta_{\text{KL}} \cdot \text{KL}(\pi_{\text{old}}||\pi_{\text{new}})$$

This dual approach ensures that training halts when policies diverge excessively while providing continuous regularization to keep policy updates within reasonable bounds. The rationale is that large policy changes invalidate the importance sampling assumptions underlying PPO, leading to biased gradient estimates.

3.2.4. Dual Clipping for Enhanced Stability

HAPPO_Optimized incorporates **dual clipping** to provide additional stability guarantees:

$$L_{\text{clip}} = \mathbb{E}[\min(\min(r_t A_t, \text{clip}(r_t, 1-\epsilon, 1+\epsilon)A_t), \gamma A_t)]$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ and γ is the dual clip parameter (typically 3.0). This mechanism prevents extremely large policy updates even when the standard clipping fails, providing an additional safety net against the cascading instability observed in the original HAPPO.

3.2.5. Enhanced Learning Rate Scheduling

The optimization includes **sophisticated learning rate scheduling** with multiple options (linear, cosine, exponential) to better adapt to different training phases:

$$\text{lr}_t = \text{lr}_0 \times \begin{cases} 1 - \frac{t}{T} & \text{(linear)} \\ 0.5 \times \left(1 + \cos\left(\pi \frac{t}{T}\right)\right) & \text{(cosine)} \\ \exp\left(-5\frac{t}{T}\right) & \text{(exponential)} \end{cases}$$

This addresses the original HAPPO's static learning rate, which can be too aggressive in later training stages when fine-tuning is required. The cosine schedule, in particular, provides smooth transitions between exploration and exploitation phases.

These optimizations collectively transform HAPPO from a brittle algorithm requiring extensive hyperparameter tuning into a robust method that can automatically adapt to changing learning dynamics, approaching the inherent stability that HASAC achieves through its fundamentally different off-policy soft Q-learning approach.

3.3. Performance Results and Analysis

The experimental results demonstrate that HAPPO_Optimized successfully addresses the core

stability issues of the original HAPPO algorithm. From the training curves and final evaluation metrics, several key improvements are evident:

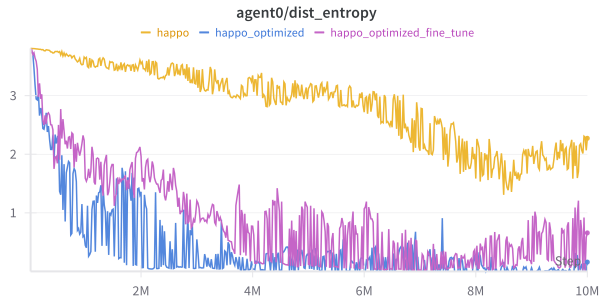

Figure 5. dist entropy

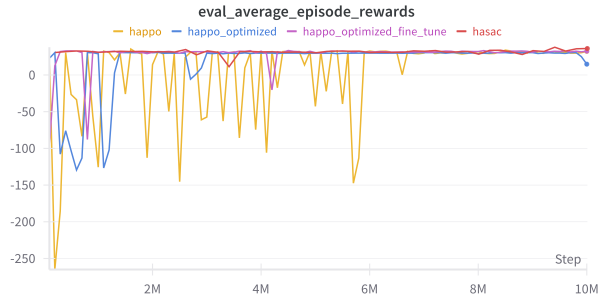

Figure 6. training curve of optimized happo

Training Stability Enhancement: The entropy evolution curves reveal that HAPPO_Optimized achieves significantly more **stable entropy decay** compared to the original HAPPO. While original HAPPO exhibits erratic entropy fluctuations throughout training, HAPPO_Optimized maintains a smooth, controlled entropy reduction from approximately 3.5 to near 0, indicating well-regulated exploration-exploitation transitions. This stability stems from the adaptive entropy scheduling mechanism that automatically adjusts λ_{ent} based on current policy entropy relative to target values.

Faster Convergence Speed: HAPPO_Optimized demonstrates **significantly faster convergence** compared to the original HAPPO. The training curves show that HAPPO_Optimized reaches stable performance levels much earlier in training, avoiding the prolonged oscillatory behavior that characterizes the original algorithm. This acceleration stems from the adaptive mechanisms that prevent policy divergence and maintain consistent learning progress throughout training.

Final Performance Comparison: The final evaluation rewards reveal the performance hierarchy: HASAC (36.06) > HAPPO_Optimized (32.84 and 32.46) > original HAPPO (32.13)[1]. Although HASAC maintains a slight performance advantage, HAPPO_Optimized successfully eliminates the catastrophic failures observed in the original algorithm and achieves competitive performance levels.

Adaptive Mechanism Effectiveness: The smooth training progression of HAPPO_Optimized validates the effectiveness of the adaptive clipping and early stopping mechanisms. The KL divergence monitoring prevents the cascading instability that plagued the original HAPPO, while the dual clipping provides additional safety nets against extreme policy updates.

The results confirm that while HASAC's inherent off-policy advantages provide natural stability, HAPPO_Optimized successfully bridges the gap through carefully designed adaptive mechanisms, transforming an unstable on-policy algorithm into a reliable training method.

References

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S. Soft actor-critic algorithms and applications, 2019. URL https://arxiv.org/abs/1812.05905.

Kuba, J. G., Chen, R., Wen, M., Wen, Y., Sun, F., Wang, J., and Yang, Y. Trust region policy optimisation in multi-agent reinforcement learning, 2022. URL https://arxiv.org/abs/2109.11251.

Liu, J., Zhong, Y., Hu, S., Fu, H., Fu, Q., Chang, X., and Yang, Y. Maximum entropy heterogeneous-agent reinforcement learning, 2025. URL https://arxiv.org/abs/2306.10715.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

[1]The result is logged in `result/render/bash.log`. You can also use the `lag_render.py` to reproduce this result.