UAV Swarm Tracking With Multi-Agent RL

Jayanth Shreekumar, Rohan Kamatar, Sanidhya Shrivastava

Abstract

Multi-agent reinforcement learning is recently an often-proposed approach to controlling a UAV swarm for surveillance and tracking of targets in many use-cases including security, conservation, and search and rescue efforts. Through the implementation and simulation of various multi-agent algorithms and learning frameworks within a simplified tracking environment in OpenAI's PettingZoo, this work develops insights into what makes RL tracking algorithms successful, and how to tackle the UAV swarm problem in complex environments.

1 Introduction

This project aims to study the problem of surveillance of a dynamic environment using a swarm of unmanned arial vehicles (UAVs), guided by reinforcement learning. Surveillance of multiple dynamic targets using multiple UAVs necessitates a multi-agent reinforcement learning (MARL) problem setup and algorithm.

To simplify this high-level objective due to time and resource constraints, we focused on studying an open-source multi-agent cooperative-competitive environment that mirrors a UAV swarm tracking problem. Through training and analysis of MARL results in the simplified environment, we can extrapolate lessons about the more complex UAV swarm problem. Outcomes from this work are a comparative study of a selection of RL algorithms, a comparison of how the type of intelligence of tracking targets changes the RL problem, and a study of the effects of partial observability.

2 Background and Motivation

The use of UAV swarms for surveillance and tracking can provide a high impact in a variety of scenarios including but not limited to: disaster zone surveillance, search and rescue, wildfire management, endangered species protection, and the tracking of illicit activities. UAV swarms can also be used to localize an unauthorized RF/GPS jamming near civilian airports. Centralized control of UAVs requires high computational load and deep existing knowledge of the environment in order to efficiently design paths for each UAV and tasks to accomplish. Traditional decentralized control also requires complex information sharing and particle filtering/sensor fusion methods that can lead to inefficient behavior in complex environments.

Multiple authors have proposed MARL for the control of UAV swarms, since MARL allows for learning coordination and collaboration between independent agents that cannot be easily managed through traditional deterministic methods (Arranz et al. 2023) (Bhagat and Sujit 2020) (Pan et al. 2022). Deep reinforcement learning (DRL) combined with MARL uses deep-learning to enhance policy optimization and state-action value learning within a multi-agent environment. While MARL and DRL may enhance the capabilities of a UAV swarm, they come with additional challenges compared to traditional reinforcement learning including:

- Non-stationary: The environment is dynamic as other agents' policies change, making learning more difficult (Lowe et al. 2017)
- Decentralization and resource sharing; Agents should have their own policy but still be aware of the actions and states of other agents
- Target invariance: agents must consider the overall goal of tracking all agents and not consider only their own tracking goal

The challenges from multi-agent learning are evidenced and discussed in the results and conclusion sections of this work.

3 Related Works

This section focuses on related works in two primary avenues: the setup and current approaches for generic multi-agent reinforcement learning, and similar applications of reinforcement learning to UAV swarm tracking.

The MARL problem faced in UAV tracking, and in our simplified setting, is characterized as either cooperative (in the case of random behavior of targets) or cooperative-competitive (when targets are capable of learning to avoid surveillance). In the cooperative case, agents have an additive reward and learning is achieved by maximizing the total reward of the agents. In contrast, in the cooperative-competitive use case groups of agents are competing against each other, where the reward function of one group is negative the reward function of an adversary group. The extreme case of this is a zero-sum game. (Gronauer and Diepold

2022) MARL is also characterized by its training and execution framework.

MARL methods are either centralized or decentralized in learning and execution. In centralized learning and execution, a single policy is learned for all agents with shared parameters, and the same policy is used in execution. In decentralized learning and execution, each agent learns its own single-agent policy without knowledge of the other agents' observations and actions; thus policies are individually, rather than jointly, maximized. Decentralized training and centralized execution falls between two extremes; most commonly, learning in this framework is done jointly (often a single critic), while execution is done individually (separate policies). (Yu et al. 2022)

Regarding specific algorithms for our use-case, (Lillicrap et al. 2015) proposes deep deterministic policy gradient (DDPG) while (Lowe et al. 2017) extends DDPG for the multi-agent case, creating MADDPG. Our initial plan was to compare these algorithms; however, based on the initial computation time and failure to learn, our work instead focused on extending proximal-policy opitimization (PPO) to the multi-agent case as studied in (Yu et al. 2022). We also focused on multi-agent soft actor-critic (SAC), another option created by extending (Haarnoja et al. 2019) with either decentralized or centralized training and execution.

(Arranz et al. 2023) provides a recent review of the UAV swarm tracking problem, including traditional optimization, biological, and reinforcement learning methods. This work also show a taxonomy of the UAV swarm problem that is useful for understanding where an MARL algorithm fits in to the operational picture. While (Arranz et al. 2023) breaks the base layer of UAV control into path-planning and mission planning, other works wrap both layers into a single RL problem. (Bhagat and Sujit 2020) and (Zhou et al. 2021) take distributed architectures for UAV swarm control for surveillance and use deep reinforcement learning to accomplish both mission and path planning as a single step (per UAV). Other works consider partial observability (Pathak et al. 2017) and communication limitations between agents (Yuan et al. 2023), which is a complication not dealt with in this work.

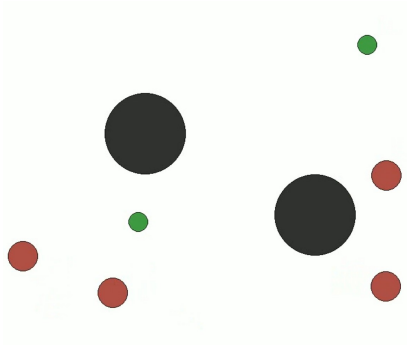

Figure 1: Sample frame from the simple tag environment

4 Methodology

The overall execution of this work followed the plan below:

1. Investigate and select a simplified multi-agent simulation environment
2. Review and select candidate MARL approaches applicable to the UAV swarm problem
3. Implement single-agent RL algorithm in multi-agent case
4. Implement independent multi-agent RL algorithm
5. Implement collaborative (centralized-critic) algorithm
6. Compare relative performance of multiple RL structures in simplified environment
7. Implement additional environment complexities to more accurately model UAV swarm tracking problem

We initially planned to use DDPG and MADDPG as the single-agent and multi-agent comparison algorithm, but were unable to successfully implement either algorithm in the PettingZoo and Rllib libraries. As such, we used both PPO and SAC algorithms for multi-agent investigation. These algorithms are detailed more closely below. Time-permitting we also planned to implement an fully custom UAV tracking environment, but deemed it outside of the time and complexity contraints of this project.

Important details of the methodology are presented in further detail.

Environment Selection

We focused firstly on exploring multi-agent RL environments that may be applicable to testing the UAV swarm tracking problem. In order to build a robust tracking environment in a short timeframe, we utilized existing libraries that implemented related multi-agent RL games.

In testing multi-agent learning environments, we chose to focus on two open-source libraries that we can potentially adapt to train and test a UAV swarm tracking problem: the Multi-Agent Tracking Environment (MATE) (Pan et al. 2022) and Petting Zoo (Terry et al. 2021). Both libraries build multi-agent environments that use OpenAI's gymnasium framework. PettingZoo includes a simple tag environment that comes closest to the UAV swarm tracking problem; however, neither PettingZoo nor MATE include out-of-the-box functionality that matches our exact RL problem.

Simple Tag PettingZoo (Terry et al. 2021) is a free open-source library for testing multi-agent reinforcement learning algorithms. Simple tag is part of the multi-particle environment set that models cooperative-competitive games with simple colored dots.

While RLlib and PettingZoo do have a framework in place for them to work together, quite a bit of it is legacy and so, we worked on putting the two together - leveraging the Simple Tag environment from PettingZoo and the DDPG algorithm from RLLib. The implementation details are as follows:

1. 4 predators (agents) and 2 prey (targets).
2. Pytorch framework was used for the deep learning model. The actor network was an MLP with 2 hidden layers with 64 neurons each and a learning rate of 0.0001.

Aspect	PettingZoo Simple Tag	MATE
Problem Setup	Small, fast-moving targets attempt to avoid larger, slower agents in an environment with obstacles	Multiple cameras rotate and zoom to attempt to keep mobile targets within their field of view
State Space	Position and velocity of each agent, obstacle and target	Location, size, orientation of each camera, target, obstacle, etc...
Observability	Full observability (can be modified to limit it)	Partial observability suitable for real-world scenarios
Action Space	Continuous actions (x and y velocity)	Camera agents have continuous rotation and zoom actions, targets have continuous velocity
Reward Function	Agents have positive reward for hitting target, target gets negative reward for hitting agent	Rewards for target coverage, penalties for collisions, rewards for unique target tracking
Agents	Customizable amount of agents and targets	Multiple teams of agents, each tracking targets
Training Approach	Centralized training with decentralized execution	Decentralized and team-based training for competitive and cooperative dynamics
Evaluation Criteria	Average rewards of agents and targets	Coverage of targets, collision avoidance, target-switching efficiency

Table 1: Environment Specifications for PettingZoo Tag and MATE

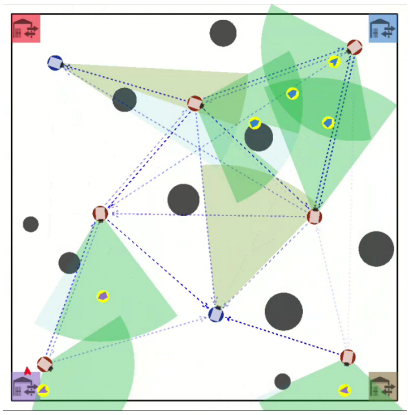

Figure 2: MATE 8v8 config environment

The critic network was another MLP with 2 hidden layers with 64 neurons each and a learning rate of 0.001.

3. The stopping criteria was one of 2500 iterations, giving the agents as well as the adversaries sufficient time to learn the optimal policy.

To visualize the training progress, code was written to save a video of the agents taking part in a simulation sample. An output frame from one of these samples is below:

The red circles are the predator agents, the green circles are the prey agents, and the black circles are obstacles.

MATE Multi-Agent Tracking Environment (MATE) is an environment that simulates the target coverage control problems in the real world. MATE hosts an asymmetric cooperative-competitive game consisting of two groups of learning agents—"cameras" and "targets"—with opposing interests. (Pan et al. 2022) The cameras, representing directional sensors, are tasked with optimizing their perception area to maximize target coverage. The targets, on the other hand, are mobile agents responsible for transporting cargo between randomly assigned warehouses while attempting to minimize their exposure to the camera network. This setup creates a zero-sum game, where the objectives of the two agent groups are in direct opposition.

For the implementation of this asymmetric two team zero sum stochastic game evaluation focuses on

1. the coverage rate of the cameras over targets, with rewards for maximizing target detection time for cameras and penalties for detected targets.
2. Targets are evaluated on their ability to transport cargo successfully between warehouses while minimizing detection.

Training emphasizes competitive strategies where cameras learn to maximize detection efficiency, while targets learn to minimize exposure time during cargo transport, creating an adversarial learning dynamic.

We chose the PettingZoo simple tag environment for continuation of simulation and testing due to a few key advantages: simple tag approximates all actors as circles, which simplifies the action space and hence the learning time required; the tag environment movement and objectives are already similar to those of a UAV tracking task; and finally, simple tag already has significant documentation and integration with rllib algorithms.

Definition of Multi-Agent MDP

While part of our methodology is to change the environment parameters such as the reward function, the observability and the agents, it is helpful to enumerate the elements of the baseline Markov Decision Process (MDP) that is formed by the simple tag environment. We formulated the environment as an episodic MDP for each agent.

- **State Space** ($\mathcal{S}$): 2D grid with the positions and velocities 4 agents and 2 targets. Not observed directly
- **Observation Space** ($\mathcal{O}$): $\mathcal{O} \in \Re^{24}$ where each set of 4 elements is the (p_x, p_y, v_x, v_y) of the current agent and all other agents/targets in the environment.
- **Action Space** ($\mathcal{A}$): Each agent has an action space defined by 2 continuous forces $f_x, f_y \in [-1, 1]$ Forces act as acceleration, changing the velocity of each agent over time.

- **Reward function (R)**: Based on intersections with targets, defined explicitly in future sections
- **Transition Function (P)**: Deterministic, agents move in direction of current velocity while velocity is impacted by the current action:

$$x_{t+1} = x_t + v_{x,t} * \Delta t \tag{1}$$

$$v_{t+1} = v_t + f_{x,t} * \Delta t \tag{2}$$

  In this case $\Delta t = 1$ always, and agents are clipped to have maximum speed of 1.0 for agents and 1.3 for targets
- **Time Horizon** T: $T = 100$ cycles per episode for execution. Some experimentation training for shorter and longer episodes, but not enough results to present.
- **Discounting** γ: $\gamma = 0.95$. Discounting required for some algorithms to converge

Built on OpenAI's gym class definitions, the exact implementation of the observation space, action space, and transition may be slightly different than described in order to account for boundaries, edge cases, proper physical movement, etc. but the functionality of the MDP remains the same.

Algorithm Review and Selection

DDPG, PPO, and SAC algorithms were all considered for comparison in the UAV tracking problem. DDPG was the initial candidate for simulation, however, documentation within the rllib implementation and test runs confirmed that DDPG was not converging in the simple tag setting, this may be due to the initial sparse reward design, (Matheron, Perrin, and Sigaud 2020), but the issue was not further investigated. Since soft actor-critic is another off-policy deep learning algorithm that evolved from DDPG, we instead tested SAC in the multi-agent setting.

SAC Soft Actor-Critic is a model-free, off-policy reinforcement learning algorithm intended for continuous action spaces. It incorporates components of both value-based and policy-based approaches. The fundamental innovation of SAC is the inclusion of an entropy term in the aim, which drives exploration by maximizing both reward and policy entropy. SAC uses separate networks for policy (actor), Q-values (critics), and a value function, resulting in highly efficient and reliable learning.

PPO Proximal Policy Optimization is an on-policy policy-gradient reinforcement learning algorithm that approximates a policy function. it takes a small policy update step, so the agent can reliably reach the optimal solution. It improves on classic policy optimization techniques by utilizing a clipped surrogate target, which prevents massive, disruptive policy modifications. PPO alternates between sampling trajectories under the present policy and optimizing the policy based on the gathered data. Even independent PPO (distributed training distributed execution) was shown to have promising performance in a multi-agent setting (Yu et al. 2022).

Algorithm 1: Soft Actor-Critic (Haarnoja et al. 2019)

Input: Initial policy parameters θ , Q-function parameters $\phi 1$,$\phi 2$, V-function parameters, empty replay buffer $\mathcal{D}$
Initialize: Set Target parameters equal to main parameters $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$

repeat
  Observe state s and select action $a \sim \pi_\theta(\cdot|s)$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  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 (number of updates) **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets for Q and V functions
      Compute targets for the Q functions:

$$y(r, s', d) = r + \gamma(1 - d)\left(\min_{i=1,2} Q_{\text{target},i}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right), \quad \tilde{a}' \sim \pi_\theta(\cdot|s')$$

      Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s)|s)\right),$$

      where $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot|s)$ which is differentiable w.r.t θ via the reparameterization trick.
      Update target networks with:

$$\phi_{\text{target},i} \leftarrow \rho \phi_{\text{target},i} + (1 - \rho)\phi_i, \quad \text{for } i = 1, 2$$

    end for
  end if
until convergence

Algorithm 2: Proximal Policy Optimization (Schulman et al. 2017)

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

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \delta$$

$$\delta = \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} \hat{A}^{\pi_k}(s_t, a_t), g(\epsilon, \hat{A}^{\pi_k}(s_t, a_t))\right)$$

  typically via stochastic gradient ascent with Adam.
  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 some gradient descent algorithm.
end for

Implementation and Training

Rllib, the open source deep learning and reinforcement learning library from Ray has existing implementations of PPO and SAC that we utilized for training (Moritz et al. 2018). Taking the base algorithms presented above, we extended them for the multi-use case and integrate them with PettingZoo simple tag.

We initially planned to include two obstacles to make the environment more complex and force the agents to learn obstacle avoidance; however, the simple tag obstacles were not static or solid as we expected, which complicated training and visualization. A decision was made to exclude obstacles to speed up training and improve visualization.

The first multi-agent framework tested was decentralized training and decentralized execution for all agents (red dots/UAVS) and targets (green dots). In this case, each agent learns its own Q-function and follows its own policy. Next we investigated a simple case of centralized learning and decentralized execution, where a single policy is trained using samples from all agents, and that single policy is repeated for all agents.

The next training configuration was a collaborative only setting. In this case, only the agents were trained, while targets behaved randomly. Since, there were no groups competing and learning against each other, we'd expect this setup to have a faster and more monotonic convergence compared to cooperative-competitive settings. A collaborative only setup, is also relevant to the UAV swarm case because in cases such as wildfire management or endangered species monitoring, we would expect the targets to behave randomly rather than attempting to avoid detection.

The last training structure we aimed to try was a shared-critic framework, where each agent develops its own policy based on a single Q-function updated based on samples from all agents. We have been unable to properly implement this structure at the time of writing.

Reward Function Modification In both independent and parameter sharing algorithms, we found that training was taking an unrealistic amount of time to stabilize. This may be because the reward function in the initial environment is too sparse to promote quick learning. To improve the learning process, we added a negative reward based on distance to the nearest target for each agent and an positive reward to each target based on distance to the nearest agent.

$$R_{agent,old} = 10 * \mathbf{1}[pos_{agent} = pos_{target}] \tag{3}$$

$$R_{agent,new} = R_{agent,old} - 0.1 \min_{tgt' \in targets} \sqrt{(x_{agent} - x_{tgt'})^2 + (y_{agent} - y_{tgt'})^2} \tag{4}$$

where tgt refers to each possible target.

Heuristically, the distance based reward provides more immediate information to each agent about it's policy gradient, promoting faster learning. Additionally, the distance based reward bears resemblance to a UAV tracking problem as it could be derived from pinging an RF tag, LIDAR, or an RF power monitor in the case of tracking emissions.

Partial Observability The next setup tested was including partial observability in the environment setup for the reinforcement learning problem. In the base environment, each agent holds the position of all other agents and targets in its observation space. The partial observability wrapper that was implemented limits the viewable radius of each agent to a configurable distance relative to the size of the environment. If the distance from the current agent to the observation is greater than the observability radius, that observation is nulled for the agent.

The partial observability is meant to approximate a real UAV use-case in that real-world sensors/communication devices will have a limited sensitivity and thus operational region. Beyond this region, the UAVs might not know their relative locations or the locations of the targets - this simple partial observability wrapper allows us to study how partial observability will change the training and algorithm performance. We'd expect a partially observable environment to be significantly harder than the original fully observable environment.

5 Results

Comparison of Independent and Single-Shared Policy

Each multi-agent framework was trained for at least 200,000 steps in the modified tag environment. To evaluate the performance of the policies, we compare the mean reward for

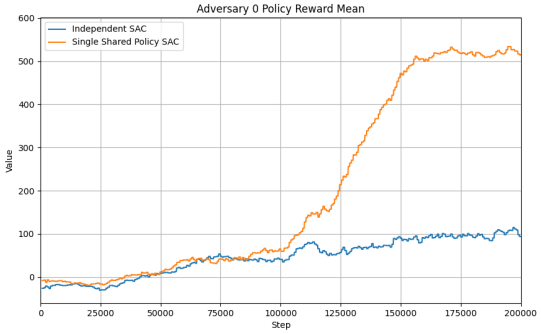

Figure 3: Agent Reward for each SAC training configuration, cooperative setting

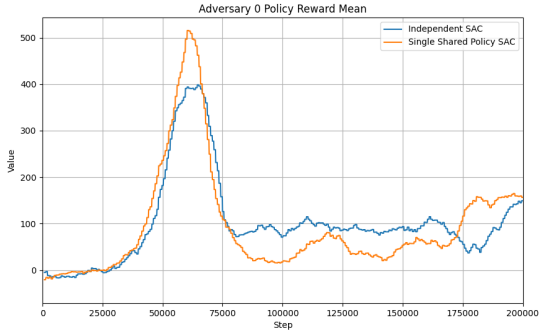

Figure 5: Agent Reward for each SAC training configuration, cooperative-competitive setting

adversaries and analyzed the actor and critic losses for each policy. Comparison was done for a collaborative only setting and for the cooperative-competitive setting. Reward comparison is presented in figures 3 and 5.

In the collaborative only setting, the single policy reaches a much higher episodic reward compared to the individual policies. This is likely because the single-policy is a centralized training method, which means that per each training iteration a single policy is fed with 4 times the data (on both actor and critic networks). Further, an often cited drawback to a single policy is that each agent is not able to learn smaller niches relative to the larger policy. In a simple setting where each agent can benefit from the same actions this does not create a drawback compared to the individual policies.

Actor loss in the collaborative setting, shown in figure 4 supports the idea that the single policy is able to learn faster than the individual policy. After initial learning (approx. 80,000 steps) the loss of the single policy decreases dramatically on a log scale indicating that the test policy is close to the optimal policy given a current understanding of the Q-function.

In the cooperative-competitive setting, figure 5, there is only a trivial difference between training configurations.

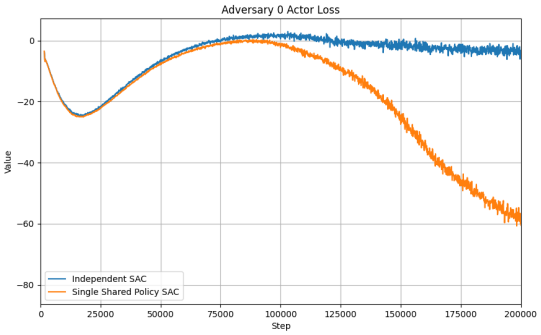

Figure 4: Actor loss for each SAC training configuration, cooperative setting, log scale

Since both the agents and targets are training in this environment, we speculate that the increased training efficiency of a single policy is negated in this setting. An individual policy does not perform better than repeating a single policy across all agents in any of the tested cases - this outcome may serve as a lesson learned when implementing RL policies for many more UAVs, since training is more efficient when making a single policy.

Cooperative Only vs. Cooperative Competitive Environment

We analyzed cases where agents pursued targets that either moved randomly or attempted to avoid capture based on a competing RL algorithm. Results shown in figure 6 show how a trained target learns to avoid agents, resulting in a decrease in the mean reward of the agent. Compared to the random use case, agents only get better at tracking targets as is expected from a reinforcement learning problem. However, if the targets are learning as well, they develop strategies to avoid detection and the reward dips back down. The reward for both agents and targets in the collaborative-competitive setting eventually does stabilize: in a perfectly optimal policy, this stabilization would be reflective of the built-in advantage present in the environment - in this case, it seems to reflect that the environment favors the agents ability to track targets. Another possibility is that the agents have given more training time the targets will learn to better avoid agents. Due to time and resource constraints we were not able to train longer than approximately $2e5$ steps for all configurations, thus comparison plots were limited to $2e5$ steps. As shown in figure 7 the increase in agent loss corresponds to a decrease in the target loss function. The two loss functions are not directly opposite, but this figure illustrates how the agents and targets learn against each other. Reward functions and loss functions oscillate to favor each group compared to the monotonic improvement often present in cooperative settings

Partial Observability

Partial Observability is a large potential complication in the multi-agent setting. At a high level, if a single agent is un-

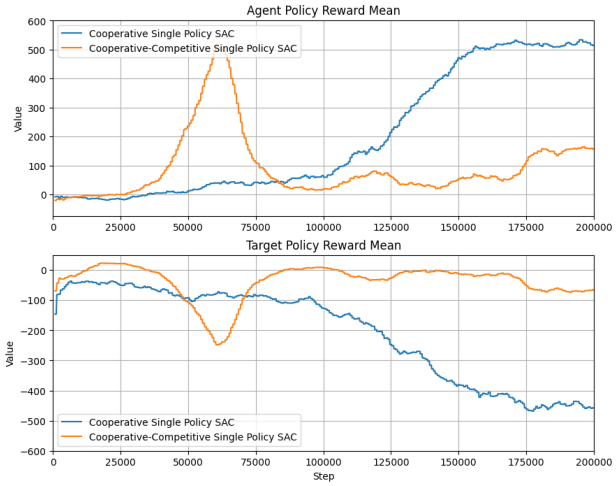

Figure 6: Agent and Target Reward for each cooperative and cooperative-competitive settings

aware of the actions or position of other agents, it can't associate their state with changes in reward; therefore, we'd expect learning in a partially observable setting to be harder and slower.

Figure 8 shows the mean reward of the shared single policy case, trained with a SAC algorithm for various observability radii. The observability radius defines how close an object must be to the current agent for it's position to be included in that agent's observation as explained further in the methodology section.

With full-observability, the agents quickly learns a high reward policy. In contrast, with a 0.5 observability radius, the agents do seem to learn, increasing from 0 to 50 over 300k steps; however, learning is much slower. With an observability radius of 0.3, the agents no longer seem to learn over this training size. It is easily evident how important knowledge of the full environment is to quickly learning an optimal policy in this simple setting. We can extrapolate to a complicated UAV tracking setting to theorize that it is critically important that agents are able to use as much environment data as possible when forming policies.

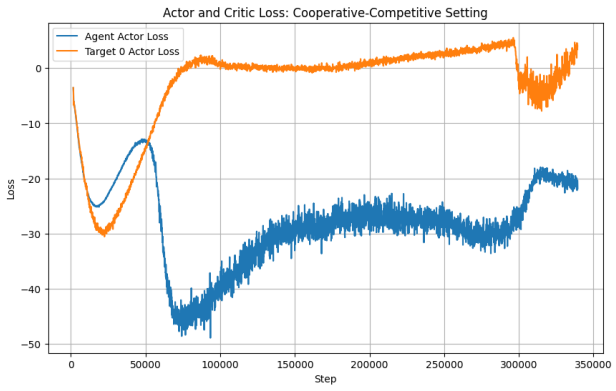

Figure 7: Agent and target loss in the cooperative-competitive setting

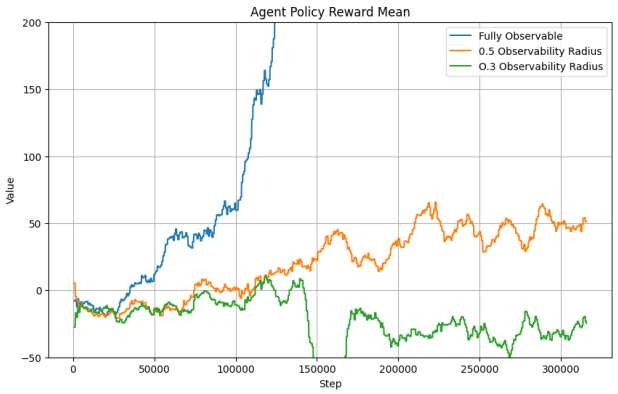

Figure 8: Mean reward for single shared SAC policy when trained with different levels of observability

PPO vs. SAC

We intended to test multiple configurations of PPO against the SAC results shown in the previous sections. In simulation, our implementation of the PPO algorithm extended from rllib failed to learn in a meaningful way even across many more iterations than required for SAC. Figure 9 shows the training reward across $1.2e6$ trials. It is evident from the mean reward that PPO is failing to learn to track the agents while SAC achieved orders of magnitude better performance in far shorter time.

While it is possible that SAC is a more suitable algorithm to this environment, the lack of result from PPO indicates that the algorithm requires configuration/tuning fixes in order to achieve better results. We would expect an optimized PPO algorithm to perform far better than achieved in the presented results.

A brief look at the training metrics of the independent PPO algorithm supports the assertion that the policy is not being learned. Figure 10 shows the loss of the value func-

Figure 9: Mean reward for Independent SAC and Independent PPO policies for a collaborative only setup.

tion estimator for each step of PPO training. One would expect the loss to decay towards 0 as the value function approximation improved. The lack of convergence may be explained by a poor value function clip (set at 10) for this trial, a learning rate that is too high/low, or a KL factor that is too high - meaning the policy is changing too much from iteration to iteration. We were unable to investigate these potential changes within the time and resource constraints of the project.

Agent Clumping

In visualizing results, we observed a phenomena not captured in the reward function that would have an impact on real-world success metrics for a UAV tracking problem. As shown in figure 11, we observed cases where all the agents would cluster around a single target while a second target remained unobserved. The agent reward in this episode was approximately 1400, indicating that agents were still being rewarded for only following a single target.

To combat this problem, we introduced a second reward modification that rewarded agents based on the average distance between them and penalized each agents any time two agents collided - all this intended to encourage spreading out over clumping.

$$R_{agent,new} = R_{agent} + 0.05 * \frac{\sum_{a' \in agents} ||pos_{a'} - pos_{agent}||_2}{NumAgents - 1} - \sum_{a' \in agents} 10 * \mathbf{1}[pos_{agent} = pos_{a'}], a \neq agent \quad (5)$$

Figure 12 shows the new reward for a single shared policy trained with the new reward function. It is important to note that the because we changed the reward function, using mean reward is not an apples-to-apples comparison. Both policies show clear improvement in reward function over time; the new reward may require further training to converge to an optimal policy, but an idea of performance is possible from the current state. An evaluation based on agent

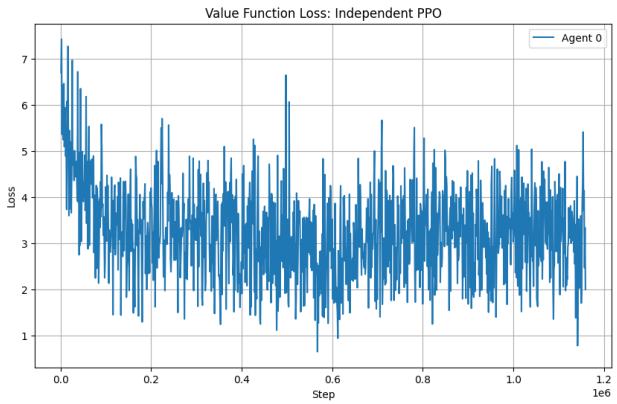

Figure 10: Value Function loss for Agent 0 in IPPO training

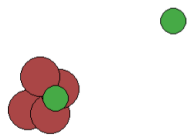

Figure 11: Snapshot from single SAC policy with high reward where agents are all tracking a single target

spread and pct of time a target is covered would more accurately capture the performance of a policy relative to a UAV tracking use case.

Focusing instead on the visualizations, we can anecdotally analyze the performance of the new reward function. In many evaluations agents behaved as shown in figure 13 where they drifted towards maximum separation rather than tracking targets. This indicates that the new reward function had too large a scale factor on the spreading metric.

One positive takeaway is that there is merit to increasing spread of agents by modifying the reward function. The new reward function also requires more coordination, since only a single agent should be near each target. This may motivate the use of a more adaptable policy such as Independent SAC, a fully centralized framework, or a separate actor, joint-critic framework.

6 Conclusion and Future Work

Using the simple tag environment this project was able to investigate many of the intricacies of a MARL problem in a way that may be extended to the more complex problem of tracking and surveillance using a swarm of UAVs.

Firstly, we were able to see that an independent SAC and a single-policy SAC are able to achieve suitable tracking performance. Furthermore, the single-policy is able to outperform independent SAC in both sample efficiency and mean reward. This implies that if the environment does not require

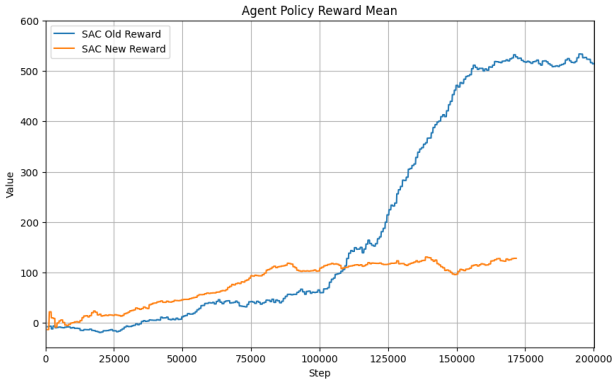

Figure 12: Mean agent reward for single shared policy SAC, old and new reward functions

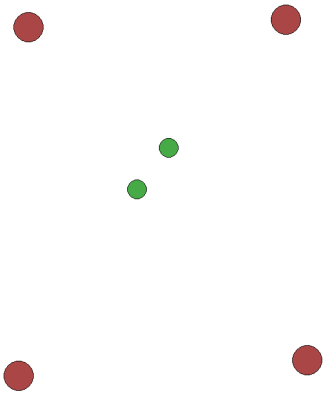

Figure 13: Snapshot from single SAC policy with modified reward function

advanced coordination, it may be suitable to use a single agent RL policy on every UAV. This is an surprising and important result in that simple tasks may save significant time and money if they deem the environment simple enough to succeed without coordination between UAVs. Further study should determine at what level of coordination a centralized critic or independent SAC may become beneficial.

Based on the study of cooperative and cooperative-competitive environments, we can theorize that tracking agents require a different policy and achieve a lower level of tracking success when the targets are capable of learning to avoid detection. In the simple setup, tracking agents held a steady state advantage over targets. This result bears more detailed analysis, since in the real-world the observability of targets and their action-space may be far more varied compared to the standard assumptions of the simple environment.

From the simple environment we are also able to experiment with reward function modification, the complexities of partial observability, and different algorithms such as independent PPO. Future work should focus on tuning these modifications to gain more insight into their performance under well configured parameters. Centralized-critic, and hierarchical multi-agent frameworks under an SAC algorithm would also be insightful future tests that build off this work.

References

Arranz, R.; Carramiñana, D.; Miguel, G. d.; Besada, J. A.; and Bernardos, A. M. 2023. Application of Deep Reinforcement Learning to UAV Swarming for Ground Surveillance. *Sensors*, 23(21).

Bhagat, S.; and Sujit, P. 2020. UAV target tracking in urban environments using deep reinforcement learning. In *2020 International conference on unmanned aircraft systems (ICUAS)*, 694–701. IEEE.

Gronauer, S.; and Diepold, K. 2022. Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review*, 55(2).

Haarnoja, T.; Zhou, A.; Hartikainen, K.; Tucker, G.; Ha, S.; Tan, J.; Kumar, V.; Zhu, H.; Gupta, A.; Abbeel, P.; and Levine, S. 2019. Soft Actor-Critic Algorithms and Applications. arXiv:1812.05905.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Lowe, R.; Wu, Y. I.; Tamar, A.; Harb, J.; Pieter Abbeel, O.; and Mordatch, I. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30.

Matheron, G.; Perrin, N.; and Sigaud, O. 2020. *Understanding Failures of Deterministic Actor-Critic with Continuous Action Spaces and Sparse Rewards*, 308–320. Springer International Publishing. ISBN 9783030616168.

Moritz, P.; Nishihara, R.; Wang, S.; Tumanov, A.; Liaw, R.; Liang, E.; Elibol, M.; Yang, Z.; Paul, W.; Jordan, M. I.; and Stoica, I. 2018. Ray: A Distributed Framework for Emerging AI Applications. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 561–577. USENIX Association.

Pan, X.; Liu, M.; Zhong, F.; Yang, Y.; Zhu, S.-C.; and Wang, Y. 2022. MATE: Benchmarking Multi-Agent Reinforcement Learning in Distributed Target Coverage Control. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Pathak, D.; Agrawal, P.; Efros, A. A.; and Darrell, T. 2017. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, 2778–2787. PMLR.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347.

Terry, J.; Black, B.; Grammel, N.; Jayakumar, M.; Hari, A.; Sullivan, R.; Santos, L. S.; Dieffendahl, C.; Horsch, C.; Perez-Vicente, R.; Williams, N.; Lokesh, Y.; and Ravi, P. 2021. PettingZoo: Gym for Multi-Agent Reinforcement Learning. In Ranzato, M.; Beygelzimer, A.; Dauphin, Y.; Liang, P.; and Vaughan, J. W., eds., *Advances in Neural Information Processing Systems*, volume 34, 15032–15043. Curran Associates, Inc.

Yu, C.; Velu, A.; Vinitsky, E.; Gao, J.; Wang, Y.; Bayen, A.; and Wu, Y. 2022. The Surprising Effectiveness of PPO in Cooperative, Multi-Agent Games. arXiv:2103.01955.

Yuan, L.; Zhang, Z.; Li, L.; Guan, C.; and Yu, Y. 2023. A survey of progress on cooperative multi-agent reinforcement learning in open environment. *arXiv preprint arXiv:2312.01058*.

Zhou, W.; Liu, Z.; Li, J.; Xu, X.; and Shen, L. 2021. Multi-target tracking for unmanned aerial vehicle swarms using deep reinforcement learning. *Neurocomputing*, 466: 285–297.