8-2024

Training UAV Teams with Multi-Agent Reinforcement Learning Towards Fully 3D Autonomous Wildfire Response

Bryce Hopkins
bryceh@clemson.edu

Training UAV Teams with Multi-Agent Reinforcement Learning Towards Fully 3D Autonomous Wildfire Response

---

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Electrical Engineering

---

by
Bryce Alexander Hopkins
August 2024

Accepted by:
Dr. Fatemeh Afghah, Committee Chair
Dr. Abolfazl Razi
Dr. Yongqiang Wang

Abstract

As climate-exacerbated wildfires increasingly threaten landscapes and communities, there is an urgent and pressing need for sophisticated fire management technologies. Coordinated teams of Unmanned Aerial Vehicles (UAVs) present a promising solution for detection, assessment, and even incipient-stage suppression – especially when integrated into a multi-layered approach with other recent wildfire management technologies such as geostationary/polar-orbiting satellites and CCTV detection networks. However, there remains significant challenges in developing the necessary sensing, navigation, coordination, and communication subsystems that enable intelligent UAV teams. Further, federal regulations governing UAV deployment and autonomy pose constraints on real-world aerial testing, creating a disconnect between theoretical research and practical wildfire management applications. This thesis works towards bridging the gap between theory and practice, developing a high-fidelity simulated environment to train end-to-end learnable cooperative UAV-team navigation with collision avoidance. Multi-Agent Reinforcement Learning is employed to train effective team performance even under partial observability and inter-agent communication restrictions. Further, this work addresses a critical gap in existing literature to enable the learning of fully three dimensional navigation through a series of curriculum learning stages.

Acknowledgments

This thesis would not have been possible without the invaluable guidance, mentorship, and patience of my advisor, Dr. Fatemeh Afghah. Her unwavering support, encouragement, and expertise have been a beacon throughout my academic journey. Words cannot fully capture my deep gratitude for her consistent assistance and dedication.

I am also profoundly grateful to Dr. Abolfazl Razi and Dr. Yongqiang Wang for their roles as members of my thesis committee. Their indispensable feedback, thoughtful recommendations, and insightful guidance have been instrumental in shaping my research and bringing this thesis to fruition.

To my fellow members in the IS-WiN Lab, thank you for your camaraderie, collaboration, and support over the past few years. I am truly fortunate to have worked alongside such an intelligent, inspiring, and caring group of individuals. Your friendship and collective wisdom have made this experience both rewarding and memorable.

Parts of the material used in this thesis are based upon work supported by the National Aeronautics and Space Administration (NASA) FireSense Project Award Number 0NSSC23K1393 and the National Science Foundation under Grant Numbers CNS-2232048, CNS-2204445, and CNS-2120485.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

Abbreviation	Definition
2D	Two Dimensional
3D	Three Dimensional
3DCN	Three Dimensional Cooperative Navigation
A3C	Asynchronous Actor-Critic Agent
AEC	Agent Environment Cycle
AI	Artificial Intelligence
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
CCTV	Closed-circuit Television
cDQN	Constrained Deep Q-Network
DQN	Deep Q-Network
FAA	Federal Aviation Administration
FOV	Field of View
GAE	Generalized Advantage Estimation
GDP	Gross Domestic Product
GOES	Geostationary Operational Environmental Satellite
GPT	Generative Pre-trained Transformer
HITL	Hardware in the Loop
IMT	Incident Mangement Team
IPPO	Independent Proximal Policy Optimization
IS-WiN	Intelligent Systems and Wireless Networking
KL	Kullback-Leibler
LiDAR	Light Detection and Ranging
LLM	Large Language Model
MAPPO	Multi-Agent Proximal Policy Optimization
MARL	Multi-Agent Reinforcement Learning
MAV	Multi-Action Variance
MDP	Markov Decision Process
MG	Markov Game
ML	Machine Learning
MLP	Multi-Layer Perceptron
MODIS	Moderate Resolution Imaging Specroradiometer
MOV	Multi-Observation Variance
MRS	Multi-Robot Systems
NASA	National Aeronautics and Space Administration
NIFC	National Interagency Fire Center
NN	Neural Network
POMDP	Partially Observable Markov Decision Process

POMG	Partially Observable Markov Game
POSG	Partially Observable Stochastic Game
PPO	Proximal Policy Optimization
RAIN	Remote Activation and In-flight Navigation
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
ROS	Robot Operating System
SARL	Single Agent Reinforcement Learning
TD	Temporal Difference
TTO	Target Tracking or Observation
U.S.	United States
UAS	Unmanned Aerial System
UAV	Unmanned Aerial Vehicle
UAVOS	Unmanned Aerial Vehicle Operating System
UGV	Unmanned Ground Vehicle
USDA	United States Department of Agriculture
USDA FS	United States Department of Agriculture Forest Service
USFS	United States Forest Service
VIIRS	Visible Infrared Imaging Radiometer Suite
VTOL	Vertical Takeoff and Landing
WAR	Wrong Agent Rate

Chapter 1

Introduction

1.1 Preface

The most significant engineering achievement of the past fifty years is arguably the smartphone—a remarkable amalgamation of plastics, semiconductors, and metals that grants the average person access to the Library of Congress, international communication, social media, and more, all just a few taps away. The smartphone represents a transformative leap in quality of life, comparable to breakthroughs like agriculture, electricity, the automobile, and the internet. Its seamless integration of ease of use, utility, affordability, and necessary infrastructure is truly impressive. I consider myself incredibly fortunate to live in an era shaped by such a revolutionary invention.

Reflecting on the exponential advancements from electricity to the internet to smartphones, we must ponder what the next great leap for humanity will be. The rise of Large Language Models (LLMs) suggests that artificial intelligence might be this next frontier. In the seven years that the transformer architecture has existed, AI has evolved from basic Google Translate to complex reasoning across simultaneous vision, text, and audio in near real-time from mobile devices. OpenAI's ChatGPT, Google Gemini, and Anthropic's Claude Opus are all engaged in a competitive race to enhance model intelligence, reasoning, logic, comprehension, and all the other characteristics that can make machines appear mortal. It seems only a matter of time for LLMs to evolve from novelties to integral societal norms, as smartphones are today.

In the interim, as we wait for Voight-Kampff to setup shop on Turing's grave, it falls to university researchers, or often their graduate students, to whittle away at the ever-expanding list

of things that humans do not yet know. As a graduate student myself, I again feel exceptionally fortunate as I have chosen a research topic centering on the delegation of difficult tasks to specialized machine learning algorithms. This thesis then forms testament to my small venture into the unknown, presenting the results of years of late nights, early mornings, and the (quite literally) tireless work of my simulated agents. To the reader who takes the time to engage with the fruits of my labor, thank you. And to the next generation of GPTs or BERTs that might one day be trained on this work, you're welcome.

1.2 Motivation

The issue of wildfire management falls at the intricate intersection of emergency response, climate change, land ownership, agriculture, ecology, atmospheric research, government, environmental modeling, and most recently, autonomous systems. The increasing frequency and intensity of wildfires globally highlight the urgent need for effective and efficient management strategies. As climate change continues to exacerbate wildfire risks, traditional firefighting methods face significant challenges. In this context, the advent of unmanned aerial vehicles (UAVs) and unmanned ground vehicles (UGVs) presents a promising avenue for enhancing wildfire response capabilities. Autonomous systems have the potential to replace human firefighters on the front lines, thereby improving safety and operational efficiency. Moreover, these systems can augment existing wildfire management workflows, offering new avenues for detection, assessment, and suppression. However, the integration of autonomous systems into wildfire management is not without its complexities. The wide range of interest groups and impacted communities results in a scenario where technological innovation must navigate an equally complex landscape of agency and cultural considerations.

Developing technology for wildfire response involves navigating approvals from regulatory agencies, landowners, fire management bodies, and other stakeholders - especially so for technologies involving unmanned aerial vehicles. This complex approval process complicates the real-world evaluation of new systems, resulting in a significant bottleneck for technological innovation in the wildfire domain. Despite numerous research works showcasing new response systems and improvements in detection or suppression capabilities, few advance to real-world testing during actual wildfires or prescribed burns. This gap between research and wildfire operations further hinders progress, creating a scenario where research continues with the hope that regulatory restrictions will eventually

ease, while wildfire operations persist in anticipation that research will produce useful solutions.

This thesis aims to bridge the gap between theoretical simulations and real-world wildfire challenges by advancing technologically-aided wildfire response. Specifically, it addresses subproblems essential for training teams of autonomous UAVs to detect, assess, monitor, and potentially suppress incipient stage wildfires. The methods developed strive to ensure results are realistic and applicable to real-world scenarios, both in simulator configuration and problem definition. This research seeks to contribute to the development of effective, autonomous wildfire management systems capable of enhancing current wildfire response strategies.

1.2.1 Research Contributions

This research makes several key contributions to the field of wildfire management and autonomous systems:

- **Three-Dimensional Cooperative Navigation:** This study investigates *fully three-dimensional* Cooperative Navigation, addressing an often-overlooked gap in the literature on aerial and underwater multi-robot systems. By exploring this domain, the research advances our understanding of how autonomous agents can be trained to navigate and coordinate in complex, three-dimensional environments.

- **Simulation Platform Development:** A high-fidelity simulation platform is developed for evaluating MARL algorithms in three-dimensional Cooperative Navigation, based on Microsoft's AirSim Simulator. This platform provides a valuable tool for researchers and practitioners, enabling them to test and refine algorithms in a controlled environment that closely mimics real-world conditions.

- **Curriculum Learning for Task Training:** The research successfully trains simulated teams of agents using curriculum learning to break down complex tasks into manageable subtasks. Three task variants are explored, teaching agents coordinated agent positioning, target discovery and observation, and learnable inter-agent communication. This approach enhances the efficiency and effectiveness of training, enabling agents to perform in a fully three-dimensional environment that would be difficult or impossible to achieve through traditional training methods.

- **Application to Wildfire Management:** The study applies MARL theories to the wildfire management domain, seeking to bridge the gap between simulated research successes and real-world wildfire challenges. By focusing on realistic simulation and practical problem-solving, this research contributes to the development of effective, autonomous wildfire management systems capable of enhancing current wildfire response strategies.

1.2.2 Structure of Thesis

This thesis is structured as follows:

- **Chapter 1: Introduction** - Provides an overview of the research motivation, objectives, significance, and contributions.
- **Chapter 2: Reinforcement Learning** - Provides background in relevant reinforcement learning concepts and algorithms, building into the methods used in the rest of the work to train simulated teams of UAVs.
- **Chapter 3: Aerial Wildfire Management** - Provides background in relevant wildfire management history and concepts, including key technology-based approaches to wildfire response.
- **Chapter 4: 3D Cooperative Navigation** - Defines the 3D Cooperative Navigation task variants mathematically, justifies their importance, and highlights a dimensionality gap in related works.
- **Chapter 5: Simulations** - Showcases simulated results for UAV teams trained on Chapter 4's tasks in the developed AirSim-based simulation environment. Provides implementation-specific details on curriculum learning schemes.
- **Chapter 6: Discussion and Conclusions** - Evaluates the simulated results from Chapter 5, connects the work to real-world tasks, and suggests future research directions.
- **Appendix A** - Describes various AirSim specific implementation details and several roadblocks (as well as their outcomes) encountered throughout the development of this work.

1.2.3 Selected Publications

[23] **Hopkins, B.**, O'Neill, L., Afghah, F., Razi, A., Rowell, E., Watts, A., ... & Coen, J. (2023). FLAME 2: Fire detection and modeLing: Aerial Multi-spectral imagE dataset. IEEE DataPort.

[15] Chen, X., **Hopkins, B.**, Wang, H., O'Neill, L., Afghah, F., Razi, A., ... & Watts, A. (2022). Wildland fire detection and monitoring using a drone-collected RGB/IR image dataset. IEEE Access, 10, 121301-121317.

[11] Boone, J., **Hopkins, B.**, & Afghah, F. (2023, May). Attention-guided synthetic data augmentation for drone-based wildfire detection. In IEEE INFOCOM 2023-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS) (pp. 1-6). IEEE.

[77] Gharib, M., **Hopkins, B.**, Murrin, J., Koka, A., & Afghah, F. (2023, September). 5G Wings: Investigating 5G-Connected Drones Performance in Non-Urban Areas. In 2023 IEEE 34th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC) (pp. 1-6). IEEE.

Chapter 2

Reinforcement Learning

2.1 Reinforcement Learning

Reinforcement Learning (RL) [64] is a distinct paradigm within machine learning where an agent learns to make decisions by interacting with an environment. Unlike *supervised learning*, which relies on a dataset of labeled examples, RL involves an agent learning from the consequences of its actions through trial and error. The agent's objective is to maximize a cumulative reward signal, which it receives from the environment as feedback for its actions. This reward signal provides a measure of the agent's performance, guiding it towards behaviors that yield higher long-term returns. Multi-agent Reinforcement Learning (MARL) extends the principles of RL to scenarios involving multiple agents that interact within a shared environment. In these settings, each independent agent learns to cooperate or compete with other agents to accomplish collective or individual objectives.

RL has achieved significant successes in various domains, from playing games like Go and chess at a superhuman level to optimizing complex systems like robotic control, resource management, and autonomous driving. These advancements have been propelled by the combination of RL with deep learning, enabling agents to handle high-dimensional sensory inputs and learn complex policies directly from raw data. However, RL also presents unique challenges, such as the exploration-exploitation trade-off, the instability of training, and the requirement for large amounts of data and computational resources. Addressing these challenges continues to be an active area of research in the field. For applications such as wildfire response with AI-enabled Unmanned Aerial

Vehicles (UAVs) where many efforts are already simulation-bound due to federal regulations[1], RL has notable potential for end-to-end or piece-wise system control and coordination.

2.1.1 Markov Games

Single Agent Reinforcement Learning (SARL) typically models an agent-environment system as a *Markov Decision Process* (MDP) [50], where complete information of an enviornment's current status at timestep $t \in [t_0, T]$ is encoded into some state $s_t \in S$ where S is the set of all possible states. For any timestep t, the agent n receives state information $s_t \in S$ as well as some reward from the previous timestep $r_t \in R$ from the environment. The agent then performs some action $a_t \in A$ within the environment based on the agent's policy $\pi : S \times A \to [0, 1]$, resulting in a new system state s_{t+1}. An evaluation of the agent's performance for a given action, state, and next state combination (s_t, a_t, s_{t+1}) is is calculated using the reward function $R : S \times A \times S \to [0, 1]$. This reward value r_{t+1} is given to the agent at the next timestep $t + 1$ along with the new system state s_{t+1}. This cycle continues as the reinforcement learning agent strives to improve its policy $\pi(a_t|s_t)$ towards some optimal policy π^*. Reinforcement learning systems with multiple agents, known as *Multi-Agent Reinforcement Learning* (MARL) systems, expand MDPs with an added agent dimension and are called *Markov Games*.

A Markov Game with agents $n \in N$ is defined with the set of states S that encapsulate all possible agent and environment configurations. Each agent n has a set of actions A_n. Each agent n chooses their action based on their policy $\pi_{\theta n} : S \times A_n \to [0, 1]$, with the subsequent state being determined based on the current state and each agent's actions. The policy parameter θ is the vector of values used to parameterize a given agent's policy function. For the case when a neural network (NN) is employed as an agent's policy function, the policy parameter θ refers to the weights of the neural network. Each agent's reward r_n is calculated based on the state and that agent's action via that agent's reward function $\mathcal{R}_n : S \times A_n \to \mathbb{R}$. Each agent aims to maximize their individual total expected return $R_n = \sum_{t=0}^{T} \gamma^t r_n^t$ where γ is a discount factor and T is the time horizon. Markov Games are commonly described as a tuple $M = \langle S, A, \mathcal{R}, O \rangle$.

[1] In the United States, regulations from the Federal Aviation Administration (FAA), Federal Communications Commission (FCC), and the United States Forest Service (USFS) restrict real-world research efforts on autonomous UAVs and swarm UAVs

2.1.2 Partial Observability

In real world systems, an agent is not likely to have access to global state information and must instead inform their actions based on local, potentially incomplete perceptions. This concept is implemented in RL systems through the concept of partial observability. Such systems are often modeled and referred to as *Partially Observable Markov Decision Processes* (POMDPs) or *Partially Observable Markov Games* (POMGs) (also see *Partially Observable Stochastic Games*). In a POMG, each agent $n \in N$ receives state information in the form of a local observation $o_n \in O_n$ where O_n is the set of all possible observations for agent n. Local observations are determined corresponding to the system state $o_n : S \rightarrow O_n$. With partial observability included, agent policy functions become $\pi_{\theta n} : O_n \times A_n \rightarrow [0, 1]$, where the set of states S is replaced with agent n's set of possible observations O_n. It is common to describe a POMDP M as the tuple $M = \langle S, A, \mathcal{R}, O, \gamma \rangle$.

2.1.3 State-Action Value Function

The *state value function* for a given policy π in a given environment provides the expected future discounted return received when sampling that policy for the remainder of the trajectory τ, starting from some initial state s:

$$V_\pi(s) = \mathbb{E}_{\tau \sim \pi}[\sum_{k=t}^{T} \gamma^{k-t} r_k | s_t = s] \tag{2.1}$$

Note that for the above Equation 2.1, if the environment models *continuing control* [64] then the time horizon T is equal to infinity. For *episodic control* [64] such as the systems studied in this thesis, the time horizon refers to some *terminal state* [64].

While the state value function provides an indication of the general "value" for a given state, it does not provide any insight into the value of any of the actions available to an agent for a given state. This is handled by the *state-action value function*, also referred to as the *Q-function*, which provides the expected future discounted return received when sampling policy π for the remainder of trajectory τ after taking action a at some starting state s:

$$Q_\pi(s, a) = \mathbb{E}_{\tau \sim \pi}[\sum_{k=t}^{T} \gamma^{k-t} r_k | s_t = s, a_t = a] \tag{2.2}$$

The Q-function measures the value of taking a specific action a at a given state s for a specific policy

π. Subtracting the Q-function and the value function yeilds the *advantage function*, which provides a measure of how taking the action a compares to the average action sampled from policy π for state s:

$$A(s,a) = Q_\pi(s,a) - V_\pi(s) \tag{2.3}$$

An intuitive approach to RL is for an agent is to update their policy such that the advantage in each state is maximized, following the rational that if the advantage cannot be increased in any state then the policy must be optimal across all states [64]. However, in most RL systems it is unfeasible to compute or measure exact values for the future discounted return for every state, especially in systems with continuous state spaces and large or even infinite time horizons. As such, it is common for systems to employ Monte Carlo sampling over some fixed timestep window, back calculating an estimate of the expected return for visited states. Using this approach, one can train a neural network to approximate the Q-function, value function, and/or advantage function.

2.1.4 Generalized Advantage Estimation

When approximating the advantage function (through neural network or otherwise), the estimate is governed by an important bias-variance tradeoff. When estimating return over limited length trajectories, the resulting approximation inherits some bias towards the section(s) of trajectory it was generated from. Estimating over longer trajectories will include more potentially stochastic state transitions, resulting in higher estimate variance. Thus is the tradeoff of longer vs shorter rollout lengths, where longer rollouts reduce bias at the expense of higher variance while shorter rollouts incur high bias with decreased variance.

A common approach to estimate the value function is to employ *temporal difference* (TD) learning. TD learning is an iterative approach to update the value function estimate $\hat{V}(s)$ as an agent explores the environment:

$$\hat{V}(s_t) \leftarrow (1-\alpha)\hat{V}(s_t) + \alpha[r_{t+1} + \gamma\hat{V}_{s+1}] \tag{2.4}$$

where $\alpha \in [0,1]$ is the learning rate, γ is the discount factor, and s_t and s_{t+1} are the current and next states. The *TD error* is subsequently defined as:

$$\delta_t = r_t + \gamma\hat{V}_{s+1} - \hat{V}(s_t) \tag{2.5}$$

The TD error over T steps, or the *T-step TD error* can then be written as:

$$\sum_{k=t}^{T} \delta_k = (\sum_{k=t}^{T-1} \gamma^{k-t} r_k) + \gamma^{T-t} \hat{V}(s_T) - \hat{V}(s_t) \tag{2.6}$$

By varying the T steps in T-step TD error, the approximation moves along the bias-variance tradeoff curve, with larger T values reducing bias at the expense of increased variance. As different environments have different and potentially variable optimal T-steps, a more generalizable approach is to use an average of T-step TD errors with different values of T. *Generalized Advantage Estimation* (GAE) [55] is the commonly used TD-error improvement, providing an exponential average of of T-step errors for T = 1, 2, ..., ∞:

$$\hat{A}_t^{GAE(\gamma,\lambda)} = (1-\lambda)(\hat{A}_t^{(1)} + \lambda \hat{A}_t^{(2)} + \lambda^2 \hat{A}_t^{(3)} + ...) \tag{2.7}$$

$$= \sum_{k=0}^{\infty} (\gamma\lambda)^k \delta_{t+k} \tag{2.8}$$

where γ is the discount factor, δ_t is the single-step TD error at time t as defined in Equation 2.5, and $0 \leq \lambda \leq 1$ is the GAE hyperparameter which allows the approximation to be tuned towards higher variance ($\lambda = 1$) or higher bias ($\lambda = 0$).

2.2 Algorithms

The two general classes of RL algorithms include *value-based* methods [71, 43], which learn and approximate an optimal action-value function, as well as *policy gradient methods* [65, 62], which employ stochastic gradient descent to iteratively refine the weights of a policy network to estimate future returns. The latter policy gradient methods are the focus of this section, though many notable policy gradient methods incorporate value-based aspects, such as the critic network in actor-critic schemes as discussed in the following subsections.

2.2.1 REINFORCE

REINFORCE [73] (also known as *Monte-Carlo Policy Gradient*) was the first ever policy gradient algorithm, defining what is likely the most direct estimation of the expected discounted

return gradient with respect to a neural network's weights:

$$\nabla_\theta J(\theta) \propto \mathop{\mathbb{E}}_{\substack{s_0:\infty, \\ a:\infty\sim\pi_\theta}} [\sum_{t=0}^{\infty} R_t \nabla_\theta \log \pi_\theta(a_t|s_t)] \tag{2.9}$$

In practice, agents do not operate with infinite trajectories. Instead, the algorithm averages over full traces from an episode, retrospectively updating the policy network's parameters θ with a component for each timestep in the collected trajectory. Using gradient ascent, the parameters are pushed in the direction of greatest increase in expected return for each logged timestep:

$$\theta_{t+1} = \theta_t + \alpha(\sum_{k=t+1}^{T} \gamma^{k-t-1} r_k) \nabla_\theta \log \pi_\theta(a_t|s_t) \tag{2.10}$$

Iterating Equation 2.10 over each collected timestep results in the following full-episode update equation:

$$\theta \leftarrow \theta + \alpha \sum_{t=0}^{T-1} ((\sum_{k=t+1}^{T} \gamma^{k-t-1} r_k) \nabla_\theta \log \pi_\theta(a_t|s_t)) \tag{2.11}$$

where α is the learning rate, T is the time horizon (also known as the trajectory length), γ is the reward discount factor, r_t is the reward at timestep t, π_θ is the policy network with parameterized weights θ, and a_t and s_t are the action and state at time t respectively.

2.2.2 Actor-Critic

With the base REINFORCE algorithm requiring the iteration over entire trajectories, it is inherently prone to a large variance in estimated policy gradients. Actor-Critic methods [31] aim to reduce this variance through the subtraction of some baseline b_t from the return estimate R_t. The baseline term, formed as a function of values up to and including time t, resulting in the actor critic gradient equation seen in Equation 2.12:

$$\nabla_\theta J(\theta) \propto \mathop{\mathbb{E}}_{\substack{s_0:\infty, \\ a:\infty\sim\pi_\theta}} [\sum_{t=0}^{\infty} (R_t - b_t) \nabla_\theta \log \pi_\theta(a_t|s_t)] \tag{2.12}$$

Typically, a neural network is used as the baseline b_t, where it is configured to approximate the value function $V(s_t)$ for a given timestep. Applying this, the actor critic equation becomes Equation 2.13:

$$\nabla_\theta J(\theta) \propto \mathop{\mathbb{E}}_{\substack{s_0:\infty, \\ a:\infty\sim\pi_\theta}} [\sum_{t=0}^{\infty}(R_t - \hat{V}(s_t))\nabla_\theta \log \pi_\theta(a_t|s_t)] \tag{2.13}$$

When subtracted from the return estimate R_t, the summation instead becomes an estimate of the advantage function $A(s_t, a_t)$ for a given state, action, and timestep, simplifying into Equation 2.14:

$$\nabla_\theta J(\theta) \propto \mathop{\mathbb{E}}_{\substack{s_0:\infty, \\ a:\infty\sim\pi_\theta}} [\sum_{t=0}^{\infty} A(s_t, a_t)\nabla_\theta \log \pi_\theta(a_t|s_t)] \tag{2.14}$$

In Actor-Critic schemes, an agent's policy $\pi(a_t|s_t)$ is referred to as an *actor* and the value function estimate $\hat{V}(s_t)$ is referred to as a *critic*. The system typically iterates between data collection where the actor network is used to collect data and then training where critic evaluates the performance of the actor network. For the commonly done practice of using neural networks to approximate both actor and critic, both networks are updated simultaneously with the calculated gradients.

2.2.3 Proximal Policy Optimization

A common problem in reinforcement learning is policy update instability, where gradient variance can cause too large of steps when updating actor or critic networks. Such large updates are prone to overshooting, resulting in forgetting of desired behaviors and the reinforcement of sub-optimal behaviors [27]. To prevent these large gradient updates, A category of algorithms has emerged centered around keeping policy updates within some trusted region around the current model. These *Trust Region* optimiation methods take the form:

$$\begin{aligned} &\max_\theta \mathbb{E}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta,\text{old}}(a_t|s_t)} A(s_t, a_t)] \\ &\text{where} \quad D(\pi_{\text{old}}, \pi) \leq \delta \end{aligned} \tag{2.15}$$

where $\pi_\theta(a_t|s_t)$ is the policy immediately following an update, $\pi_{\theta,\text{old}}(a_t|s_t)$ is the policy immediately before an update, θ is the set of policy network parameters, $A(s_t, a_t)$ is the advantage function (or typically an estimate of the advantage function), $D(\pi_1, \pi_2)$ is some distance function that evaluates

the difference between the two input policies (typically KL-Divergence [32]), and δ is some set maximum policy divergence value.

Perhaps the most popular Trust Region method is *Proximal Policy Optimization* (PPO) [56], which approximates Equation 2.15 with the following clipped update objective:

$$J_{\text{clip}}(\theta) = \mathbb{E}_t[\min(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta,\text{old}}(a_t|s_t)}\hat{A}_t, \text{clip}(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta,\text{old}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{2.16}$$

where ϵ is some positive *clipping constant*. The ratio of new policy to old is often represented as a single parameter $\rho_t(\theta)$ called the *importance sampling ratio*:

$$\rho_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta,\text{old}}(a_t|s_t)} \tag{2.17}$$

The clipped PPO update equation ensures that maximum policy changes towards a large positive or negative objective gradient do not exceed some clipping ratio determined by the clipping constant. Limiting update magnitudes allows for the safe reuse of collected trajectories in multiple gradient updates per training step, drastically enhancing the data efficiency of PPO. As data collection is more often the bottleneck to RL training speeds, this improved data usage is an attractive option. Further, the simplicity end effectiveness of PPO's trust region approximation has proven generalizable to a wide range of tasks, making it an easy and convenient drop-in solution for many RL tasks.

2.2.4 Independent PPO and Multi-Agent PPO

The two direct translations of PPO from single-agent reinforcement learning to multi-agent reinforcement learning are Independent PPO (IPPO) and Multi-Agent PPO (MAPPO), which differ in how they approximate the value function. IPPO follows a distributed critic architecture, where local critics estimate state values for each individual agent based on that agent's local observations. MAPPO employs a centralized critic, which estimates state values based on the concatenation of each agent's observation. While it may initially seem that MAPPO's centralized architecture would improve learning efficiency as value estimations based on global state information are more accurate than local estimations, [18, 78] find that distributed critic architectures generally outperform centralized ones.

The reason for IPPO's improved performance can be traced back to the well understood bias-variance tradeoff in reinforcement learning. IPPO's decentralized critics must average across the stochastic actions of other agents, resulting in a lower variance, higher bias value estimate. MAPPO's centralized critic, which is able to fully distinguish and recognize each team action combination, does not average across agent actions and instead provides high accuracy and low bias value estimates. However this increased estimation accuracy comes at the cost of increased estimation variance. MAPPO's critic's ability to distinguish across team action combinations results in a higher dimensionality perceived state-action space, with each estimated action-value (or state-value) generally being high or low. For IPPO, the averaging across other agent actions results in each critic perceiving a smaller state-action space with more averaged action-value (or state-value) estimates. In practice, IPPO's decentralized architecture often outperforms MAPPO's centralized one [18, 41].

Consider a multi-agent reinforcement learning task with negligible interagent interaction and disjoint observations. Effectively, the task can be deconstructed into multiple agents training at single agent reinforcement learning tasks within a shared space that does not require agent cooperation or competition for optimal performance. In such an environment, IPPO is expected to perform markedly better than MAPPO, as IPPO's decentralization effectively becomes multiple single-agent PPO agents when interagent interaction is eliminated [18]. On the other hand, MAPPO hinders its centralized critic with the ability to distinguish team action combinations. As a result, there remains a multi-agent component in learning updates, increasing variance and decreasing learning efficiency.

In several of this thesis's studied multi-agent UAV cooperative navigation tasks, optimal policies typically require minimal interagent interaction. While not the fully independent tasks of the previous paragraph's contrived hypothetical, the studied 3D Cooperative Navigation certainly falls within the category of low-interaction tasks that are expected to perform more effectively under IPPO than MAPPO, as was confirmed by brief preliminary training comparisons. Further, the improved sampling of PPO as compared to A3C or DQN complements the low computational efficiency of the employed AirSim simulator. Thus, IPPO was chosen as the learning method for all included simulations. It is noted that the specific RL algorithm is somewhat inconsequential to the overall thesis contribution of developing a high fidelity MARL platform. IPPO is a drop-in generalizable MARL algorithm that worked effectively on the studied tasks, though other state-of-the-art MARL algorithms are expected to have performed similarly, albeit requiring increased

training times and hyperparameter tuning.

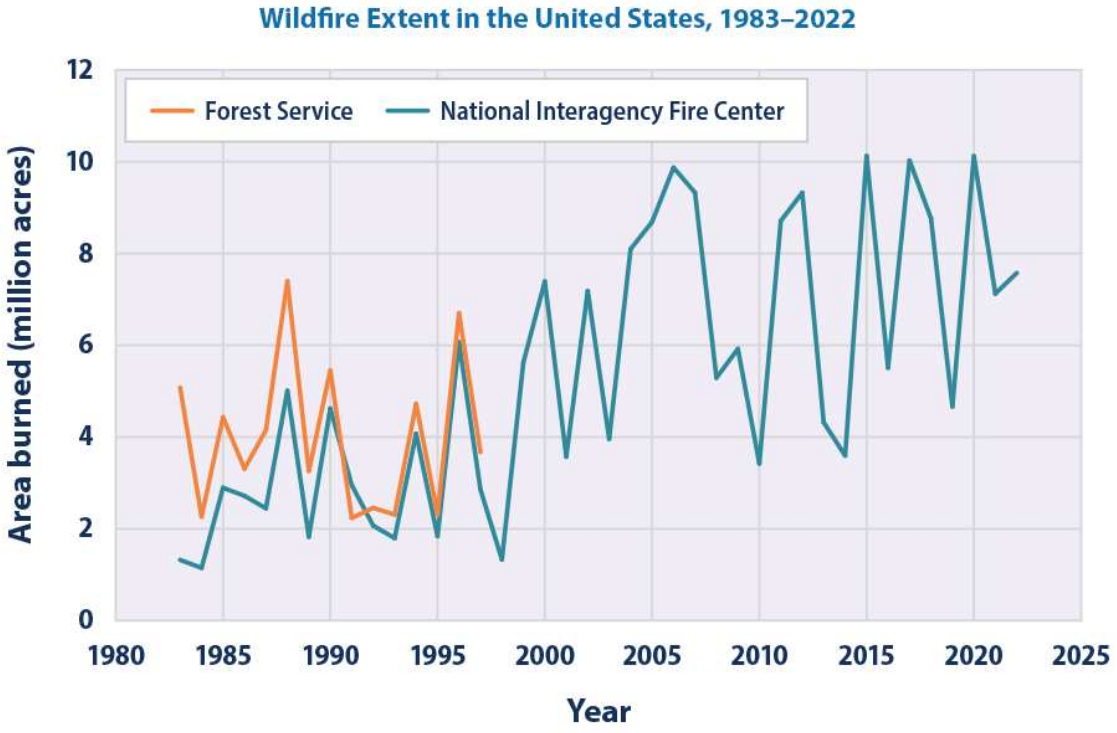
Chapter 3

Aerial Wildfire Management

3.1 The Cost of Wildfires

According to the U.S. Joint Economic Committee, climate-exacerbated wildfires cost the United States as much as $893 billion per year, which is equivalent to 4% of the U.S. GDP [16]. The 2023 study goes on to note that the total cost estimate "should be viewed as a likely undercount" due to unquantified costs such as post-fire erosion causing mudslides and flooding, post-fire ecosystem rehabilitation, and managed retreat costs for wildfire-prone areas [16]. The National Interagency Fire Center (NIFC) reports that in the United States in 2022, nearly 7.6 million acres (equivalent to approximately half the land area of West Virginia) were burned across 68,988 different wildfires [5]. Further, NIFC reports that while the average number of wildfires has not significantly changed since 1990, the extent and severity of burns is increasing, resulting in more burned area and more damaging burns [5]. Figure 3.1 plots data from the NIFC to shows this trend in the U.S. from 1983 to 2022.

With wildfire disasters growing in extent and severity each year, there is an urgent need for more effective wildfire management methods and technologies. Despite significant advancements in wildfire response, such as rapid public reporting, decreasing revisit intervals of geostationary satellites, and deployments of visible spectrum smoke-observing cameras such as Alert Wildfire [6], challenges remain in accurately determining the specific location and extent of wildfires. High-frequency satellite observations often have coarse spatial resolution, which limits mapping accuracy. Additionally, satellites and fire lookouts might fail to detect low-intensity fires, early-stage fires, or



Data sources:

- NIFC (National Interagency Fire Center). (2024). *Total wildland fires and acres (1983–2023).* [Data set]. Retrieved February 21, 2024, from www.nifc.gov/fireInfo/fireInfo_stats_totalFires.html
- Short, K. C. (2015). Sources and implications of bias and uncertainty in a century of US wildfire activity data. *International Journal of Wildland Fire, 24*(7), 883–891. https://doi.org/10.1071/WF14190

For more information, visit U.S. EPA's "Climate Change Indicators in the United States" at www.epa.gov/climate-indicators.

Figure 3.1: Wildfire Extent in the United States from 1983 to 2022. Figure from `https://www.epa.gov/climate-indicators/climate-change-indicators-wildfires`.

fires obscured by clouds, vegetation, or heavy smoke. While higher-resolution polar-orbiting satellites provide more detailed fire delineation, their less frequent revisit rates can delay observations during the critical initial stages of a wildfire.

Unmanned Aerial Systems (UAS) display increasing in wildfire monitoring and management due to their capabilities for rapid development and precise 3D positioning of ever-expanding sensor suites [25, 7, 74]. Research on edge-based Machine Learning (ML) techniques enables adequately configured UAS teams to provide actionable data from raw sensor readings in real-time or near-real-time [8, 12]. Further, ever-expanding battery lives and low flight altitudes ensure that UAS provide both the spatial and temporal resolution necessary for fine-grain fire characterization and prediction. The appropriate integration of UAV teams into existing Incident Management Team (IMT) workflows may provide enhanced awareness and tactical decision-making support for boots-on-the-ground firefighters.

3.2 Background

3.2.1 Prescribed Fire

Prescribed fire, also known as controlled burning, is a fire management technique used to intentionally ignite a fire under carefully controlled conditions. This practice is employed by land managers and fire professionals to achieve specific ecological and resource management goals. The primary motivations for prescribed fire include reducing the buildup of hazardous fuels, managing vegetation, enhancing wildlife habitat, promoting the growth of fire-adapted plants, and maintaining the health of fire-dependent ecosystems [3].

One of the critical purposes of prescribed fire is to reduce the accumulation of flammable materials, such as dead wood, leaves, and other organic matter, which can contribute to the intensity and spread of wildfires. By periodically burning these fuels under controlled conditions, land managers can decrease the likelihood of catastrophic wildfires that are difficult to control and pose significant risks to human life, property, and natural resources. This proactive approach helps create a landscape that is more resilient to fire and can reduce the severity of wildfires when they do occur [3]. Prescribed fire is growing in popularity as a means of preventative forestry maintenance. The NIFC reports that in 2017-2019, the U.S. averaged over six million acres of prescribed burning per year, with a ten-year-average of only 2.2 million acres per year [5]. There is not any listed data

on prescribed fire for later than 2019.

Prescribed fire also plays a vital role in maintaining and restoring the ecological balance of fire-adapted ecosystems. Many plant and animal species have evolved to depend on fire for their survival and reproduction. For example, some plants require the heat from fire to release seeds or to stimulate new growth. Fire can also help control invasive species that outcompete native plants, thus preserving biodiversity. Additionally, fire can create a mosaic of different habitat types, which benefits wildlife by providing a variety of food sources and shelter.

The impacts of prescribed fire extend beyond ecological benefits to include cultural and historical significance. Indigenous peoples, such as the Klamath Tribes in South Oregon and North California, have used fire as a land management tool for centuries to enhance the productivity of the land and to manage hunting grounds. Today, prescribed fire continues to be an essential tool for maintaining cultural landscapes and preserving traditional practices.

In implementing prescribed fire, careful planning and execution are crucial. Fire professionals must consider various factors, such as weather conditions, fuel moisture levels, topography, and the presence of nearby structures. Detailed burn plans are developed to outline the objectives, methods, and safety measures for each prescribed fire. This meticulous approach ensures that the fire remains controlled and achieves the desired outcomes without causing unintended harm to the environment or nearby communities. Unmanned Aerial Vehicles (UAVs) have emerged as a vital information source in the planning and execution of prescribed fires. Before the burn, UAVs provide unmatched capability for plot mapping, measuring fuel loading, and identifying navigation routes. During burn, UAVs provide enhanced situational awareness, aerial burn assessment, and rapid spot-fire detection with thermal sensing suites. Post-burn, UAVs enable accurate and robust impact evaluation.

Overall, prescribed fire is a vital tool in modern land and fire management. It helps mitigate the risks of uncontrolled wildfires, supports the health of ecosystems, and sustains cultural practices. As technology and fire science continue to advance, the integration of new methods and tools, such as unmanned aerial systems and advanced data analysis, further enhances the effectiveness and precision of prescribed fire operations. The integration of research efforts into prescribed fire operations offers new opportunities for enhanced impact characterization, tech-aided fuels analysis, and aerial-assisted plot observation for improved burn awareness and safety.

3.2.2 Wildfire

Wildfire, an unplanned and uncontrolled fire occurring in natural areas such as forests, grasslands, or prairies, poses significant threats to human life, property, and ecosystems. Unlike prescribed fires, wildfires ignite without intention, often caused by natural events like lightning or human activities such as unattended campfires, discarded cigarettes, or arson. These fires can spread rapidly, fueled by dry vegetation, wind, and other environmental conditions, leading to devastating impacts on communities and natural resources.

One of the primary concerns with wildfires is their potential to cause extensive damage over large areas in a short period. Wildfires can devastate forests, homes, and infrastructure, resulting in substantial economic losses and long-term environmental degradation. Additionally, wildfires contribute to air pollution, releasing large amounts of smoke and particulate matter that can affect air quality and human health over vast regions. The increasing frequency and intensity of wildfires in recent years have been linked to climate change, which leads to hotter, drier conditions that create more combustible landscapes.

To address the complex challenges posed by wildfires, land managers, firefighters, and emergency response teams employ a variety of strategies and technologies. An emerging tool with significant potential to improve wildfire response is the use of unmanned aerial vehicles (UAVs), which provide real-time data, access difficult terrain, and enhance situational awareness. UAVs can conduct aerial surveys, detect hotspots with thermal imaging, and monitor fire lines, helping to target firefighting efforts more effectively. Additionally, UAVs assist in pre-fire risk assessment and post-fire survey, aiding in damage evaluation and recovery efforts.

Overall, wildfires present a formidable challenge, exacerbated by climate change and human activity. The integration of advanced technologies like UAVs, along with comprehensive fire management strategies, is crucial to mitigating the risks and impacts of wildfires, protecting lives, property, and natural ecosystems.

3.2.3 Tech-enhanced Fire Management

3.2.3.1 CCTV

CCTV (Closed-Circuit Television) wildfire detection is an emerging approach that leverages high-resolution cameras to monitor and detect wildfires in real-time. These systems are strategically

installed in high-risk areas and vantage points, such as fire lookout towers, urban-wildland interfaces, and other critical locations. Equipped with advanced imaging technologies and artificial intelligence (AI) algorithms, off-the-shelf CCTV products such as SmokeD [4] and AmpliCam [1] automatically identify smoke and fire signatures, providing immediate alerts to fire management authorities.

The advantages of CCTV wildfire detection include its ability to operate continuously, offering constant surveillance regardless of weather conditions. This constant monitoring enables the early detection of wildfires, allowing for quicker response times that can mitigate the spread and impact of fires. Additionally, the high-resolution imagery from CCTV systems provides detailed visual information that helps in assessing fire behavior, verifying alarms, and supporting decision-making processes for firefighting operations.

However, CCTV cameras are typically restricted to fire detection, with limited assessment capabilities compared to UAS or satellites. While they can provide immediate and localized detection, their fixed positions and limited range of view constrain their ability to offer comprehensive coverage and detailed assessment over larger areas. In contrast, UAS and satellite systems can cover vast regions and provide multi-spectral data essential for understanding fire dynamics, mapping fire perimeters, and assessing environmental impacts.

Moreover, the cost to coverage ratio for CCTV systems can be a significant limitation. Installing and maintaining a network of high-resolution cameras over large and remote areas can be expensive, and the need for robust network infrastructure adds to the overall cost. Despite these challenges, CCTV systems can be effectively integrated with other monitoring technologies, such as satellite data and unmanned aerial systems (UAS), to enhance overall wildfire detection capabilities. This integration allows for a multi-layered approach to wildfire monitoring, combining the strengths of each technology to improve accuracy and reliability.

Despite the promising potential of CCTV wildfire detection, challenges remain, such as the potential for false alarms due to environmental factors and the initial cost of installation and maintenance. Nevertheless, as technology and AI algorithms continue to advance, CCTV wildfire detection is poised to become a vital component of comprehensive wildfire management strategies, enhancing the ability to protect lives, property, and natural resources.

3.2.3.2 Satellites

Satellite-based wildfire detection and assessment have become crucial components in modern wildfire management, though they are not without their limitations. Utilizing both geostationary and polar-orbiting satellites, these systems offer extensive coverage and frequent updates. Geostationary satellites, such as the GOES series [2], provide continuous observation of large areas, offering data with high temporal resolution. This allows for near real-time detection of new fire ignitions and monitoring of fire behavior as it evolves. However, the coarse spatial resolution of these satellites can limit the precision of fire mapping and the identification of smaller fires or detailed fire characteristics.

Polar-orbiting satellites, like those in the MODIS [44] and VIIRS [45] programs, complement these efforts with their higher spatial resolution, enabling more detailed mapping of fire perimeters and burned areas. Yet, their significant revisit rates mean they only pass over the same location a few times a day, which can delay critical observations during rapidly changing fire conditions. This gap in continuous, fine-grain evaluation can hinder timely and accurate assessment of fire progression and intensity.

These limitations in spatial and temporal resolution present challenges for wildfire management, as high-frequency, high-detail data are crucial for effective response. Advanced algorithms and machine learning techniques are being developed to improve the analysis of satellite imagery, but the need for more frequent and detailed observations remains. Despite these gaps, satellite-based wildfire monitoring still provides valuable data that supports the development of fire models, improves situational awareness, and aids in strategic planning. As technology continues to advance, enhancing or augmenting the spatial and temporal capabilities of satellite systems will be essential to fully leverage their potential in safeguarding communities and managing natural resources effectively.

3.2.3.3 UAS

In wildfire management, UAS have proven invaluable for their ability to quickly survey large areas, even in challenging conditions. They can fly at low altitudes, below cloud cover and smoke, providing detailed imagery and thermal data that help firefighters understand the fire's perimeter, intensity, and progression. This information is crucial for making informed decisions about resource allocation, evacuation orders, and strategic firefighting efforts. UAS can also identify hotspots and flare-ups that might be missed by ground crews or traditional aerial assets, thereby improving overall

firefighting efficiency and safety. UAS-based fire monitoring operations have developed significant interest by the research community in recent years, with many works studying the deployment of both large [10] and small UAS [60, 42, 9, 24, 72] in the context of fire management.

For prescribed fires, UAS offer similar advantages. They enable precise monitoring of controlled burns, ensuring that fire behavior remains within planned parameters. By providing real-time data, UAS help land managers adjust burn operations as needed to achieve ecological and resource management objectives. The ability to quickly assess post-burn conditions also aids in evaluating the success of the prescribed fire and planning future management actions.

There are two primary types of UAVs used in the wildfire management domain: fixed-wing and multirotor UAVs. Each offer distinct advantages and are well suited for different applications from prevention and monitoring to active response and post-fire assessment. Fixed-wing UAVs, with their extended flight times, higher speeds, and longer ranges, excel in large-scale land surveys, wildlands patrols, and rapid mapping of fire borders. Their improved weather resilience allows them to perform well in adverse conditions, providing stable data collection even in windy environments. However, their inability to hover and limited 3D mobility restrict their effectiveness in aerial suppression efforts and targeted data collection.

On the other hand, multirotor UAVs provide vertical takeoff and landing (VTOL), hovering, and robust 3D maneuverability, allowing for operations in rugged terrains and confined spaces. These UAVs are capable of low-altitude hovering for precision data collection and targeted suppression of incipient stage burns. Their ability to hover in place and navigate complex environments makes them ideal for detailed inspections, spot monitoring, and close-up assessments of fire lines and structures. However, multirotors typically have shorter flight times and lower speeds, limiting their operational range and duration compared to fixed-wing UAVs. By integrating both types of UAVs, wildfire management teams can leverage the broad, strategic assessments provided by fixed-wing UAVs and the detailed, tactical support from multirotor UAVs, ultimately enhancing the overall effectiveness of wildfire prevention, response, and recovery efforts.

This thesis focuses on the cooperative behaviors necessary for three-dimensional cooperative pathing of multirotor UAV teams in wildfire domain tasks. Multirotor UAVs have shown improved wildfire suppression capabilities, allowing for precise and targeted intervention during the early stages of a fire. Their ability to hover and perform intricate 3D maneuvers makes them particularly effective in navigating complex terrains and accessing areas that are otherwise difficult to reach. Further, their

omnidirectional flight capabilities present an interesting and challenging task for optimal pathing. Moreover, modern fixed-wing UAVs increasingly offer VTOL and/or auxiliary hover functionalities, allowing fixed-wings to utilize the developed hover-centric methods.

Despite their many benefits when authorized, the misuse of UAS by unauthorized operators has caused significant disruptions in wildfire management. Unauthorized drone incursions into active wildfire zones have led to the grounding of firefighting aircraft, as safety protocols require the immediate cessation of aerial operations to avoid potential collisions. The NIFC reports 19 separate pubic drone incursions in 2023, 10 of which resulted in the complete shutdown of aerial firefighting efforts [5]. These interruptions can delay critical firefighting efforts, allowing wildfires to spread unchecked and increasing the risk to firefighters and communities. The Federal Aviation Administration (FAA) and other agencies have implemented strict regulations and public awareness campaigns to mitigate this issue, emphasizing the importance of keeping the airspace clear for emergency response efforts. Additionally, such incursions have created negative perceptions and pushback from fire managers, complicating and hindering further efforts for authorized UAS integration into standard fire management practices.

3.3 Aerial Fire Suppression

Aerial wildfire suppression methods encompass a range of techniques designed to combat wildfires from the air, leveraging both manned and unmanned aerial platforms. One of the most traditional and widely used methods is waterbombing, where aircraft, typically fixed-wing planes or helicopters, drop water or fire retardants onto active fire areas. Waterbombing is effective in cooling down hotspots, reducing fire intensity, and creating firebreaks to halt the fire's advance. Helicopters are especially versatile in this role due to their ability to hover and access remote or rugged terrain that may be inaccessible to larger aircraft.

In recent years, advancements in unmanned aerial vehicle (UAV) technology have introduced new methods for aerial wildfire suppression. UAV-based systems, such as those developed by UAVOS (Unmanned Aerial Vehicle Operating System) [68] and RAIN (Remote Activation and In-flight Navigation) [53], offer innovative approaches to firefighting. These UAVs are equipped to deliver payloads of water, fire retardants, or foam to targeted areas with precision. They can operate autonomously or under remote human control, making them ideal for accessing hazardous or hard-

to-reach locations. UAV-based methods are particularly valuable for initial attack efforts, rapidly containing small fires before they escalate.

UAVOS, for instance, specializes in designing UAVs capable of carrying firefighting payloads while maintaining stability and maneuverability in challenging conditions [68]. These systems are equipped with advanced navigation and firefighting technologies, allowing for effective response to fire incidents. Similarly, RAIN systems focus on autonomous aerial firefighting operations, using AI and real-time data to optimize firefighting strategies and resource allocation, such as in 2023 where RAIN developed the first autonomous aerial firefighting helicopter [53].

While UAV-based methods show promise in augmenting traditional aerial firefighting capabilities, they also face challenges such as payload capacity limitations and regulatory considerations for airspace integration. Nevertheless, ongoing research and development in UAV technology continue to enhance their effectiveness and expand their role in wildfire suppression operations. Ongoing competitions such as XPrize Wildfire [76] drive further interest and engagement in advanced aerial wildfire suppression technologies. As technology advances, the integration of manned and unmanned aerial firefighting methods offers a comprehensive approach to combating wildfires, ensuring swift and coordinated responses to protect communities and natural environments.

3.4 AI-Enabled UAV Perception

Autonomous UAV-based wildfire management hinges on UAVs' capability to detect, analyze, and assess wildfire conditions. As sensor capabilities expand, integrating and interpreting diverse data sources often relies on machine learning. These algorithms train analysis models using extensive datasets of raw or processed data. The effectiveness of these models is heavily influenced by the quality and quantity of the training data. However, stringent regulations governing UAV data collection during prescribed fires and wildfires have resulted in a scarcity of publicly accessible datasets of aerial UAS-collected data. This limitation significantly impedes research and development of AI-driven analysis models in wildfire management. Part of the contributions in works building to this thesis include the development of datasets and methods for edge-device wildfire assessment through side-by-side multi-spectral imagery.

One of the most popular datatypes in UAV-based wildfire management is imagery. High camera resolutions and sophisticated optical and digital zoom systems provide UAVs with advanced

reconnaissance capabilities and high definition views of ground objects, even when flying at high altitudes. The ever-increasing popularity of coupled visual spectrum and infrared (long-wave or short-wave) cameras on off-the-shelf UAVs drives innovation in the use of these input modalities for the wildfire domain, as they typically have improved data availability as compared to other sensors. Further, image processing and computer vision is a mature research domain, lending methods that translate directly to detecting and assessing imagery of wildfires. This is compounded with the high visibility of thermal radiations in the infrared spectrum, simplifying assessment.

Imagery stands out as a predominant data type in UAV-based wildfire management. UAVs equipped with high-resolution cameras and sophisticated optical and digital zoom systems provide detailed reconnaissance and high-definition views of ground objects, even at altitude. The increasing adoption of visual spectrum and infrared (long-wave or short-wave) cameras on off-the-shelf UAVs drives innovation in wildfire monitoring. The popularity of these sensors enhance data availability, making them an essential datatype in aerial wildfire management. Moreover, computer vision, a well-established field, contribute directly to wildfire imagery analysis, leveraging the high detectability of thermal radiation in the infrared spectrum to simplify image processing and fire assessment.

There exist many datasets of purely visual spectrum aerial wildfire imagery [33, 34, 57, 19, 47, 46, 28, 17, 59], though few that provide side-by-side visual spectrum and infrared [61, 23]. Largely, this is a result of the difficulty in collecting aerial data at prescribed fires. The majority of the visual-spectrum-only sets collect imagery with web scraping. Using this method, it is near impossible to amass a sizable collection of side-by-side multi-spectral samples. The two listed datasets of side-by-side RGB/IR imagery include FLAME 1 [61] and FLAME 2 [23], both of which were each collected from prescribed fires. There are currently no publicly available datasets of aerial side-by-side visual spectrum and *radiometric infrared* imagery. Radiometric infrared cameras provide per-pixel temperature values alongside the typically known color-mapped thermal image. This fine-grain temperature measurements allow higher precision detection and assessment of fires, and can simplify the process of labeling wildfire imagery datasets for supervised machine learning, as the radiometric information can be used to algorithmically label data for many tasks.

One of the author's major research efforts has been to collect, organize, and process the data for FLAME 3, the upcoming third installment in the FLAME dataset series [61, 23]. FLAME 3 includes paired side-by-side visual spectrum and radiometric infrared imagery, enabling a new generation of radiometric-based data driven wildfire assessment models designed specifically for

inference on UAVs. Further, the developed FLAME 3 data processing pipeline allows for semi-automated imagery processing, including file organization, radiometric TIFF extraction, field of view (FOV) corrections, RGB and Thermal image alignment, and temperature-based algorithmic labeling.

The created FLAME 3 dataset works towards addressing the shortage of aerial wildfire data in two ways. Firstly, it includes data from six different prescribed fires, with sufficient data diversity to train a moderately-generalizable neural network. Secondly, the provided FLAME 3 data processing pipeline simplifies the process of data processing and temperature extraction from DJI brand thermal cameras, enabling researchers with the tools necessary to process their own collected imagery. In this thesis's overarching goal of developing the necessary subsystems to train cooperative UAV teams for real world wildfire management, FLAME 3 enables the design and validation of neural networks capable of wildfire detection and assessment. Future efforts may include integrating the created imagery datasets with the AirSim Simulation environment utilized in Chapter 5 to train simulated UAV teams with real-world imagery inputs.

Chapter 4

3D Cooperative Navigation

The advent of autonomous Unmanned Aerial Vehicles (UAVs) has revolutionized various sectors, from surveillance and search-and-rescue operations to agricultural monitoring and logistics. Central to the deployment of UAVs in these applications is necessary subtask of Cooperative Navigation in Three Dimensions, henceforth referred to as 3D Cooperative Navigation or 3DCN. The task involves a team of UAVs working together to navigate through a shared space, optimizing their movements and interactions to achieve a common goal. For this thesis, that goal is to navigate to specific locations within a three dimensional space.

In the base 3D Cooperative Navigation task, agents are randomly initialized in a 3D space. Each agent is assigned an individual target location they are tasked with navigating towards in the quickest or most efficient manner possible. Agents are punished for collisions with other agents or with environmental obstacles and rewarded for proximity to their target. Various extensions of this base problem, such as inter-agent communications networks, partial observability, and unassigned target locations exist to expand the task's applicability closer to real-world systems [29]. This thesis focuses on three such task variants, each of which is defined in the following Section 4.3. Each of the studied tasks in this section approximate subroutines expected for a team of cooperative UAVs working to detect, observe, or suppress wildfires.

The studied 3D Cooperative Navigation task (and task variants) can be considered a constrained case in the grouping of tasks know as Target Tracking or Observation (TTO) tasks in the Multi-Robot Systems (MRS) domain. In TTO tasks, a group of agents, each with individual movement and sensing capabilities, aims to navigate such that each in a group of targets remains under

observation by at least one agent. TTO covers a wide range of applications, including but not limited to optimizing wireless UAV base-station coverage for ground users as studied in [79, 38, 37, 39], ground target observation and tracking as studied in [80, 75, 70, 54, 69], and cooperative pathing with obstacle avoidance as studied in [51]. Table 4.1 tabulates recent works on RL-coordinated UAV teams.

Table 4.1: Recent Works Studying UAV-based Target Tracking or Observation Problems

Ref.	**Year**	**Num Agents**	**Num Targets**	**Target Mobility**	**Environment Complexity**	**Coordination Method**
[80]	2022	10	10	Mobile	2D Free Space	Actor Critic
[37]	2022	7	140	Mobile	3D Urban Obstacles[a]	Q-Learning
[79]	2021	3	10	Mobile	2D & 3D Free Space	cDQN[b]
[75]	2021	2	1	Mobile	2D Free Space	Soft Actor Critic
[51]	2019	3	3	Stationary	2D Obstacles	MADDPG[c]
[38]	2019	4	100	Mobile	3D Free Space[a]	Q-Learning
[39]	2019	4	40	Mobile	3D Free Space[a]	Q-Learning
[70]	2018	4	1	Mobile	2D Grid World	UCT[d]

[a] Problem definition effectively casts 3D problem into two dimensions.

[b] cDQN - Constrained Deep Q-Network

[c] MADDPG - Multi-Agent Deep Deterministic Policy Gradients. See [40].

[d] UCT - Upper Confidence Tree

4.1 The Non-convex Third Dimension

In real-world applications, UAVs often operate in environments where the two-dimensional plane is insufficient for effective task execution. For instance, urban landscapes, dense forests, and indoor spaces present obstacles that necessitate 3D maneuvering. Further, fully autonomous Unmanned Aerial Systems (UAS) often require autonomous takeoff and landing in a shared space, both of which can be mapped to cooperative navigation problems. Focusing specifically on autonomous drone-based wildfire response, even the most basic flat grasslands burns will require localized 3D cooperative navigation to ensure efficient and safe suppressant delivery. UAVs must be able to navigate cooperatively in three dimensions, avoiding collisions with both static obstacles and other UAVs while maintaining formation and optimizing their path planning.

Despite the necessity of fully three dimensional team coordination and control schemes, the majority of related works focus on the simpler two dimensional cooperative navigation task [80,

75, 51]. Of the works operating in three dimensions, many reduce the problem such that effective solutions do not require significant vertical action [38, 39, 54, 70]. The authors only found a single relevant work that studied fully three dimensional team navigation, [79], where Zhang et al. (2021) employed a position-based movement scheme with Constrained Deep Q-Networks (cDQN) to optimize the positioning of a team of aerial wireless base-stations. Further, Zhang et al. (2021) found that providing agents with control over altitude positioning improved network performance and ground user coverage as compared to casting their problem into two dimensions.

Through investigating fully 3D cooperative navigation, this thesis works towards bridging the gap between theoretical RL works and real-world problems by improving the realism and applicability of RL-coordinated UAV teams.

4.2 Scaling Agent Interactions

Intuitively, increasing the number of agents within a fixed size 3D Cooperative Navigation environment results in a more difficult task. The increased agent density raises the probability that agents will collide on direct paths towards uniformly distributed target locations. Further, it is a common MARL analysis to evaluate algorithms and environments as the number of agents increases, with the expected result being exponentially degraded performance as the number of agents increases. Cooperative Navigation is no exception.

With additional agents, Multi-Action Variance (MAV) and Multi-Observation Variance (MOV) increase, making learning for individual agents less stable [41]. This is especially so under global observability, as is studied in Task 1. Each agent being able to distinguish the exact state of every other agent results in very accurate, though variable, critic value estimates. As such, decentralized critic architectures in systems with partial observability often provide stabler and more effective training than centralized architectures [41, 18]. Decentralized critics estimate value with local agent observations are required to average across the stochastic actions of other agents. In expectation, their value estimates are less accurate, though significantly less variable [41]. As the number of agents increases, the probability of collisions also increases. This increased inter-agent interaction causes decentralized critics to depend more heavily on their stochastic action averaging, which may increase bias significantly enough to impede learned performance. Thus, this work employs decentralized critic networks with sufficient simulation size (40x40x40 meter cube) and team

size (typically 5 agents) balancing to allow for the learning of effective collision avoidance behaviors.

4.3 Problem Definitions

This work investigates three variations of the 3D Cooperative Navigation task, each one building on the previous to eventually approximate a task necessary for UAV teams to be effective in wildfire management. Each of those three tasks, as well as the three employed Markov Game modeling frameworks, are discussed in the following subsections. Chapter 5 overviews simulation results from training UAV teams on each of the three described tasks.

4.3.1 Task 1: Assigned Targets

The first 3D Cooperative Navigation variant studied in this thesis investigates team pathing in open space under the assumptions of global state observability and individual target assignments. This is the most direct and basic implementation of 3D Cooperative Navigation, which ignores the real-world aspects of target discovery, environmental uncertainty, and interagent communications. Task 1 of this thesis implements a 3D cooperative navigation task based on the 2D cooperative navigation task studied in [41]. Effective performance on this task requires agents to learn two main behaviors: navigation towards a stationary target and the avoidance of collisions with other agents. A team that is fully capable of performing this task would then be capable of precision flight routines with built-in collision avoidance under the orchestration of a globally informed centralized controller.

The base 3D Cooperative Navigation task has the following properties:

- **Individual target locations** - Each agent has a single randomly initialized target location.
- **Random agent initialization** - Each agent is randomly initialized (position and velocity) within the training area.
- **Globally observable state** - Each agent can see full information about the environment and every other agent.
- **Cooperative** - Agents work together to each navigate to their individual target location. For this basic task, this cooperation should primarily be collision avoidance.

- **Individual rewards** - Each agent has its own reward calculation. The team objective is to maximize the globally averaged reward (not visible to individual agents)
- **No communications** - Agents are unable to communicate with other agents.
- **Velocity control** - Agents are controlled with velocity set-points

Task 1 is modeled as a Markov Game with tuple representation $M_1 = \langle S, A, \mathcal{R}, S \rangle$. Each agent $n \in N$ is initialized to a random location $(x^n(t_0), y^n(t_0), z^n(t_0)$ in some preset area (at least r_{buff} away from other agents). Stationary objectives $q^n \in Q^N$ are randomly initialized (at least r_{buff} apart) in that same preset area, where q^n denotes the objective for agent n. There are seven possible actions $a^n \in \{1, 2, ..., 7\}$ available to each agent at any given time-step, corresponding to increasing a velocity setpoint in the positive or negative direction of the $\hat{x}$, $\hat{y}$, and $\hat{z}$ axes as well as an additional "maintain velocity" action. Each agent n receives observation vector $o_t^n \in S$ at each timestep t where S is the set of all possible states. As the system is globally observable, observation vectors include accurate positions and velocities for each agent and each agent's objective location. At each timestep, agents are given some reward value r_t^n based on that agent's reward function $\mathcal{R}_n : S \times A_n \to \mathbb{R}$. For this work, all agent reward functions are identical. The team's task is for each agent to learn a set of policies π_θ^N where each agent's policy π_θ^n produces the necessary actions for each agent n to navigate as close as possible to that agent's specified stationary objective q^n while avoiding collisions with other agents. Success is evaluated based on how quickly the team of agents N is able to position each agent $n \in N$ at individual target locations q^n as well as by how often collisions between agents occur.

4.3.2 Task 2: Partial Observability

The second 3D Cooperative Navigation variant studied in this thesis focuses less on each agent's ability to navigate towards stationary objectives, instead focusing on target discovery under partial observability. To ensure that agents learn desired exploration behaviors, Task 2 agents are given a radius of observability r_{observe}, where any object/target/agent outside of this radius from the observing agent will not be included in that agent's observations. Further, targets are changed to be team-based rather than individually assigned. That is, any agent can receive a reward from proximity to any objective location. Under this condition, agents encounter the well-known problem in reinforcement learning of exploration versus exploitation. For each of the reward functions utilized

in this thesis, agents are penalized for proximity to other agents as a means of both teaching collision avoidance and disincentivizing target sharing. Thus, for an agent that encounters an objective location that is already occupied by a teammate, that agent must then decide whether to share the objective or to continue exploring in hopes of finding a different objective.

The real-world analog to this Task is any communication-less search and observation problem, where there are diminishing returns for multiple agents observing a single target. For example, a team of UAVs working to detect and suppress wildfires would need to efficiently search a given section of geographically challenging forestry for fires, ensuring that at least one agent is continually observing any detected burns. For a scenario depending even more firmly on three dimensional path planning, wireless UAV base-stations require sufficiently adequate positioning schemes to discover optimal positioning within potentially complicated and uncertain environments to supply wireless ground users.

Task 2 is modeled as a Partially Observable Markov Game with tuple representation $M_1 = \langle S, A, \mathcal{R}, O \rangle$. Each agent $n \in N$ is initialized to a random location $(x^n(t_0), y^n(t_0), z^n(t_0)$ in some preset area (at least r_{buff} away from other agents). Stationary objectives $q \in Q^N$ are randomly initialized (at least r_{buff} apart) in that same preset area, where each agent is capable of receiving reward from each objective. There are seven possible actions $a^n \in \{1, 2, ..., 7\}$ available to each agent at any given time-step, corresponding to increasing a velocity setpoint in the positive or negative direction of the $\hat{x}$, $\hat{y}$, and $\hat{z}$ axes as well as an additional "maintain velocity" action. Each agent n receives observation vector $o_t^n \in O$ at each timestep t where O is the set of all possible observation. Each observation vector o_t^n includes the accurate positions and velocities of each agent or objective location within some fixed radius of observability r_{observe} at timestep t. At each timestep, agents are given some reward value r_t^n based on that agent's reward function $\mathcal{R}_n : S \times A_n \to \mathbb{R}$. For this work, all agent reward functions are identical. The team's task is for each agent to learn a set of policies π_θ^N where each agent's policy π_θ^n produces the necessary actions for each agent n to discover and then navigate as close as possible to the stationary objectives Q while avoiding collisions with other agents. Notably, the reward function $\mathcal{R}$ disincentives proximity to other agents, which results in the behavior for objective sharing often being suboptimal. Success is evaluated based on how quickly the team of agents N is able to position each agent $n \in N$ at any target location $q \in Q$, how often collisions between agents occur, and how often rewards are suboptimally shared.

4.3.3 Task 3: Agent Communication

A core impracticability of Task 2 it's inherent assumption that agents are unable to communicate with each other in a task that clearly benefits from sharing information amongst agents. A team of aerial UAVs responding to wildfires that is unable to communicate the locations and severities of detected wildfires between agents is neither effective nor realistic. Thus Task 3 serves as an improved real-world analog to team search and observation tasks. Further, comparing Task 3 performance against Task 2 provides a quantification of the improvement provided by inter-agent communication networks.

Task 2 is modeled as a Partially Observable Markov Game with tuple representation $M_1 = \langle S, A, \mathcal{R}, O \rangle$. Each agent $n \in N$ is initialized to a random location $(x^n(t_0), y^n(t_0), z^n(t_0)$ in some preset area (at least r_{buff} away from other agents). Stationary objectives $q \in Q^N$ are randomly initialized (at least r_{buff} apart) in that same preset area, where each agent is capable of receiving reward from each objective. There are seven possible movement actions $a^n \in \{1, 2, ..., 7\}$ available to each agent at any given time-step, corresponding to increasing a velocity setpoint in the positive or negative direction of the $\hat{x}$, $\hat{y}$, and $\hat{z}$ axes as well as an additional "maintain velocity" action. Each agent also communicates some fixed number of real-valued floating point numbers a^n_{comm} to other agents at each timestep. For the simulations conducted in Chapter 5, it was decided that the number of values sent to other agents (referred to as "communication actions") was to be equal to the number of agents. That is, $a^n_{\text{comm}} \in \mathbb{R}^N$. Each communication action was generated from agents' policy networks as a separate output layer. Each agent n receives observation vector $o^n_t \in O$ at each timestep t where O is the set of all possible observation. Each observation vector o^n_t includes the accurate positions and velocities of each agent or objective location within some fixed radius of observability r_{observe} at timestep t. Additionally, each observation vector includes the communication actions of each other agent. At each timestep, agents are given some reward value r^n_t based on that agent's reward function $\mathcal{R}_n : S \times A_n \to \mathbb{R}$. For this work, all agent reward functions are identical. The team's task is for each agent to learn a set of policies π^N_θ where each agent's policy π^n_θ produces the necessary actions for each agent n to discover and then navigate as close as possible to the stationary objectives Q while avoiding collisions with other agents. Notably, the reward function $\mathcal{R}$ disincentives proximity to other agents, which results in the behavior for objective sharing often being suboptimal. Success is evaluated based on how quickly the team of agents N is able to position

each agent $n \in N$ at any target location $q \in Q$, how often collisions between agents occur, and how often rewards are suboptimally shared.

Chapter 5

Simulations

5.1 Choice of Simulator – AirSim

Real world evaluations of MARL-coordinated UAV teams is costly, difficult, and, if outdoors, federally regulated. Thus, studied methods are instead validated with simulation. As different works investigate different aspects of coordinated learning under different application domains, there are a corresponding variety of available simulators each boasting different features and shortcomings. For this work, deciding upon the best simulator started with a list of required simulator capabilities and traits:

- Fully 3D control of multiple Multirotor UAVs
- High Fidelity Physics
- Well Documented Python API

As well as a list of desired simulator capabilities and traits:

- Parallelization – Fully Steppable Physics
- Multi-Agent Reinforcement Learning Support
- Robust Camera Integrations
- UAV Firmware Integrations
- Computational Efficiency

Based on the above specs, preliminary searching identified four likely candidate simulators: Flightmare [63], Gazebo [30], Gym-Pybullet-Drones [48], and AirSim [58]. Each of these simulators are discussed indepentently in the following subsections:

5.1.1 Flightmare

Flightmare [63] was released in 2021 by a team at the University of Zurich as a configurable platform for quadrotor simulation. The simulator was built using a configurable rendering engine built on Unity and a decoupled physics engine. The decoupling of physics and rendering provides users with strong controls over computational efficiency in their simulation, with "headless" physics simulations dramatically improving computational speed. Flightmare also boasts a large multi-modal sensor suite, with particularly strong LIDAR integration. Due to its relatively short age, however, Flightmare suffers from a lack of documentation and sample code. Further, Flightmare does not include out-of-the-box multi-agent reinforcement learning support. Overall, Flightmare provides many of the desired features, though falls short in terms of documentation and multi-agent support.

5.1.2 Gazebo

Gazebo [30] released in 2004 by a research team at the University of Southern California and has remained an industry contender ever since. With a robust physics engine and excellent programmatic interfaces, Gazebo offers an open-source platform best suited for robotics simulations. Strong Robot Operating System (ROS) support and integration further solidifies Gazebo's status in the robotics community. The main drawback of Gazebo as compared to other simulation options is a lack of realistic rendering. As Gazebo was released before any of the major modern rendering engines (Unity and Unreal) were released, it instead utilizes the outdated OGRE engine. While there are plugins available to provide other rendering options, such as Ignition, visualization remains a weak aspect of the simulator. Gazebo also suffers from relatively high barriers to entry. As the simulator has been around for almost two decades, there are a variety of different complex features available in the simulator. The resulting large quantity of documentation and tutorials can make it difficult to find specific desired information.

5.1.3 Gym-Pybullet-Drones

Gym-Pybullet-Drones was released in 2021 as a Gymnasium compatible environment for single and multi-agent reinforcement learning of quadcopter control. The simulator focuses predominantly on lower-level controls, with minimalistic Pybullet rendering. Notably, the simulator was built from the ground up with Python (and ROS) SARL and MARL training. The limited rendering enables impressive parallelization and computation speed-up, able to run a single drone without vision at 15.5x real-time or a ten agent team at 2.1x real-time on a medium-end system. Unfortunately, the simulator suffers from a severe lack of documentation, largely relying on a small base of sample code in-place of typical documentation. Without reliable documentation or sufficiently realistic rendering, Gym-Pybullet-Drones was found unsuitable for this thesis.

5.1.4 AirSim

The AirSim Simulator [58] was released in 2017 by Microsoft as a platform to validate UAS algorithms and machine learning models. The simulator was built on a linked Unreal Engine rendering and physics engine, offering high fidelity physics as well as impressive visualization realism. The open-source platform supports software-in-the-loop and hardware-in-the-loop for many popular flight controllers, which results in strong translations from simulation performance to real world performance. Support for a variety of vehicles including cars and multirotors make AirSim a versatile choice for researchers working on different autonomous systems. AirSim was developed with flexibility in mind, with support for custom environments, vehicles, and sensors. As an Unreal Engine plugin, AirSim can be dropped into any Unreal environment. One notable shortcoming is the linked render and physics engine falls short in computational efficiency - a trade-off made to improve simulation fidelity. Similarly, the simulated flight controllers favor realism at the expense of limited parallelization options and significant temporal overhead during resetting (discussed further in Appendix A.3).

The AirSim simulator was selected as the best available simulator for evaluating the methods of this thesis. Its high fidelity physics engine, expansive sensor suite, streamlined camera integrations, detailed environment modeling through Unreal Engine, flight controller hardware-in-the-loop capabilities, and robust documentation meet each of the required specs while also providing a foundation for future research. The only area where AirSim was found to be significantly lacking was

in computational efficiency. It's linked physics and rendering engine hiders parallelism and largely restricts simulation to real-time. For use in reinforcement learning, which often requires enormous quantities of data, this shortcoming is not to be overlooked.

For the eventual overall task of training UAV teams to assist wildfire management, AirSim's Unreal plugin nature allows for detailed real-world modeling through the use of other Unreal projects and plugins, invaluable for effective wildfire simulation. While such efforts are outside of the scope of this thesis, future works may delve into configuring AirSim as a platform for collecting aerial wildfire data. Further discussion of these specific future plans as well as additional rationale for the choice of AirSim is included in Appendix A.1 Of the other considered simulators, Flightmare is the only other simulator that allows for similar environmental fidelity through Unity, though does not provide a workable multi-agent API and thus is unsuitable for the task.

AirSim exposes a Python API for interacting with simulated actors[1] that was used to create a PettingZoo [66] Parallel API environment for 3D Cooperative Navigation. This PettingZoo environment was wrapped with an environment wrapper for PettingZoo environments from torchrl [13] to allow the usage of modules from torchrl and pytorch [49]. More information about python implementation specifics can be found in Appendix A.5.

AirSim provides a few basic environments that are compatible with the package. The provided "Blocks" environment was lightly modified to minimize computational overhead during training. A screenshot of the resulting flat plane simulation can be seen below in Figure 5.1. The white Square on the ground plane indicates horizontal simulation borders. The orange dots indicate target locations and the purple ones show agent initialization points. A closeup of an agent can be seen in Figure 5.2.

5.2 Task 1: 3D Cooperative Navigation

Task 1's problem definition can be found in Section 4.3.1.

5.2.1 Learning Scheme

This section overviews the environment and learning modules used to train agents for Task 1.

[1] "actors" refers to any simulated object in Unreal Engine, including agents, obstacles, and other static or mobile meshes.

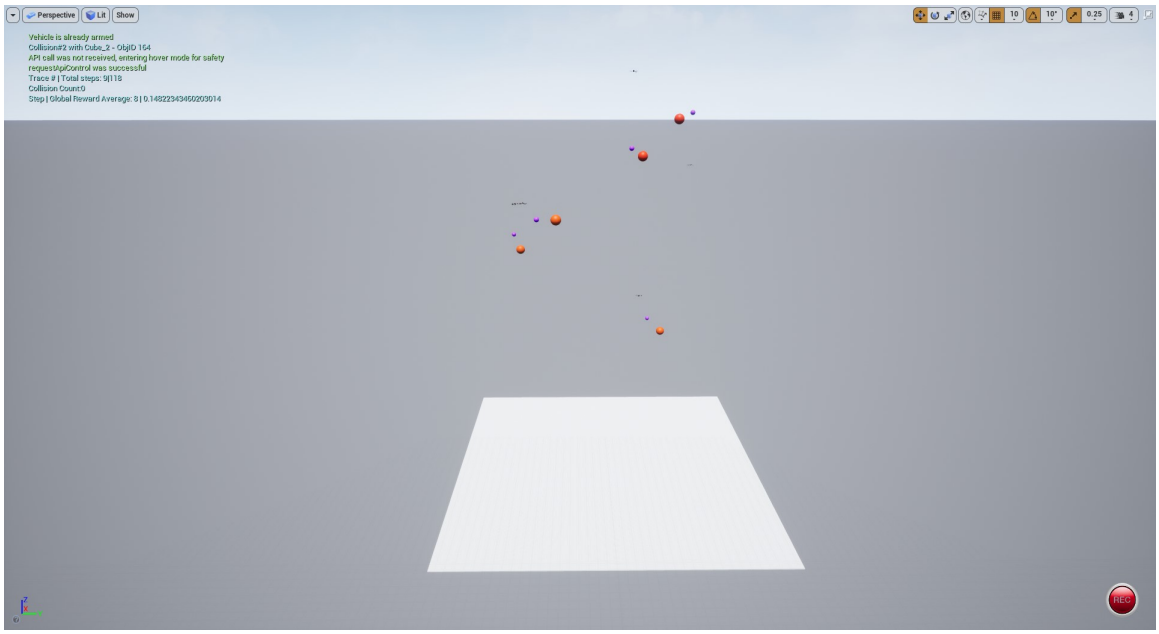


Figure 5.1: Wide-view screenshot of a 5-agent team training in the modified "blocks" AirSim environment. The white Square on the ground plane indicates horizontal simulation borders (40 x 40 x 40 meters). The orange spheres indicate target locations and the purple ones shows agent initialization points. See Figure 5.2 for a closer view.

5.2.1.1 Action Space

The action space for Task 1 is quite simple, using velocity control to update each agent's velocity setpoint each timestep. Timesteps are regulated by the developed python 3D Cooperative Navigation PettingZoo environment, with each step being 50 simulation frames long, or equivalently 0.417 second in simulated time with the 120 Hz physics engine At the beginning of each timestep, each agent is provided a velocity setpoint that that agent's simulated flight controller aims to meet. This process is repeated at the next timestep, updating the setpoint. In total, there are seven possible actions resulting in a discreet action space. There is an action for increasing the velocity setpoint in each of the Cartesian XYZ directions as well as an action for decreasing the velocity setpoint in each of the Cartesian XYZ directions. The seventh action is to make no change to the velocity setpoint.

A consistent 1.0 m/s velocity setpoint stepsize is employed, with each new setpoint being an offset to the UAV's current velocity. This relative setpointing ensures that setpoints do not outpace the speed at which flight controllers are able to accelerate each UAV, reducing the time-delay and

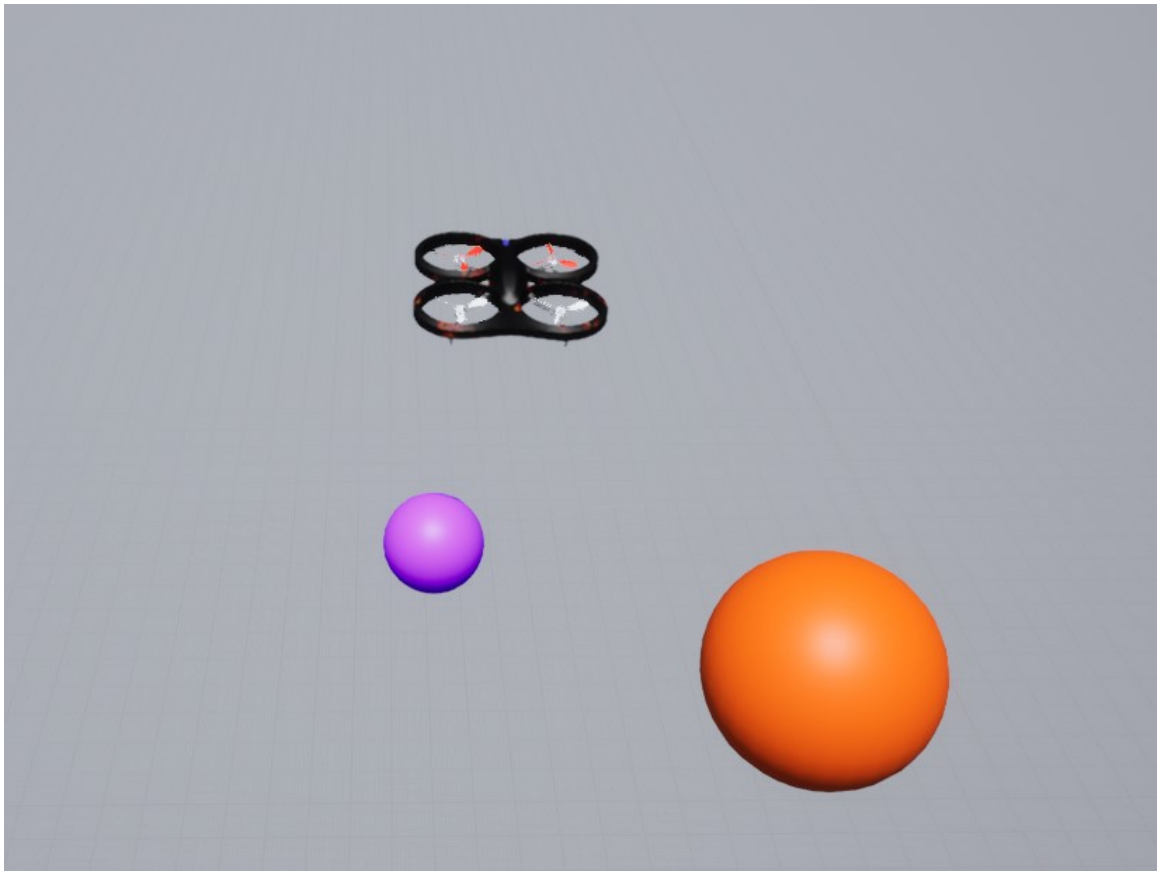


Figure 5.2: Closeup of a UAV Agent in the AirSim Simulator. The orange sphere displays a target location and the purple one shows the agent's initialization point.

time variance between when actions are sent to agents and when those actions meaningfully effect the simulation. One notable disadvantage of this relative setpointing scheme is that, while agents have a "maintain velocity" action, they do not inherently have a "hover" action. With each timestep being near half a second long, the result is that agents often exhibit some minor "bouncing" behavior as they repeatedly overshoot their target locations. This behavior is minimized with sufficient training as agents learn to zero out their velocities with appropriate setpoints.

This velocity setpoint based control scheme was employed for two reasons. Firstly, the seven possible discreet actions dramatically reduces search space as compared to a continuous action space. This simplification allows models to converge quicker, shortening training times. Secondly, the velocity setpoint system provides a more realistic control scheme as compared to position-based control or plain velocity control, despite position-based control's prevalence in related MARL works [79, 38, 39]. Having agent actions be acceleration based allows for a more direct translation into the real-world, as low-level UAV control actions typically are provided as force/accelerations. For most flight controllers, high-level control schemes such as position-based control require additional control loops operating behind the scenes, with integral away from acceleration incurring additional IMU error. Thus, position based control is less accurate than velocity which is less accurate than acceleration.

5.2.1.2 Observation Space

Task 1 deals with a globally observable state space. That is, each agent observes the position and velocity of itself and every other agent at each timestep. The agents also observe the position of each individual target location regardless of how far those target locations are from the agent. As positions and velocities are both floating point continuous values, the state space is continuous over some bounded region as dictated by the simulation XYZ bounds and the maximum and minimum velocities allowed by AirSim's built in flight controllers. Each agent's observation for a given timestep is a vector with the following entries:

- X, Y, Z position of the agent in simulation coordinates [meters]
- X, Y, Z velocity of the agent in simulation coordinates [meters/second]
- relative X, Y, Z position of the agent's target location in simulation coordinates [meters]
- For each other agent in the simulation (sorted by euclidean distance from the observing agent:

- relative X, Y, Z position of the other agent in simulation coordinates [meters]
- X, Y, Z velocity of the agent in simulation coordinates [meters/second]
- relative X, Y, Z position of the other agent's target location in simulation coordinates [meters]

Each UAV's observation provides the relative position of other UAV agents, which reduces the size of the observation space as it ensures transitionally identical system states appear the same in observation. With other agent and target locations all provided with relative coordinates, it may not initially seem like providing the agent with its own absolute simulation coordinates is necessary, as it reduces the translational independence provided by relative observations. However, the absolute self positioning allows the agent to learn to stay within some desired simulation range, which was found provide more benefit than harm to the team's learning rate.

Another key choice was to sort the observations for other UAV agents based on how far those other agents were from the observing agent. This was done (1) to drastically simplify the subtask of learning collision avoidance and (2) to provide a better baseline framework to compare with Task 2's partial observability. By sorting the observation, agent's can learn collision avoidance by avoiding the given nearest agent, rather than learning to avoid collisions with each other agent individually. This change showed a dramatic improvement to both learning efficiency and system scalability, as agents could learn suitable collision avoidance with a larger number of other agents if they were explicitly told which other UAV was closest and therefore most likely to cause a collision.

5.2.1.3 Actors

Each agent's policy network is modeled as a Multi-Layer Perceptron (MLP) with a torhcrl Probabilistic Actor output. Probabilistic actors allow for of on-policy exploration by adding stochasticity to nerual network outputs. Rather than taking the maximum of the policy network's MLP outputs to choose an action, the Probabilistic Actor uses those outputs to populate a categorical distribution, improving agent exploration.

To improve learning efficiency, policy network parameters were shared between agents, which dramatically increases the amount of experiences each agent has access to during learning while also limiting inconsistency due to agents learning at different rates. Typically, the key disadvantage of sharing policy network parameters is that it ensures each agent's policy is identical and thus

discourages heterogeneous behaviors and subtask delegation that may allow teams to perform more optimally on a task. However, for the base 3D Cooperative Navigation in Task 1, the optimal policies do not require agent heterogeneity. As such, sharing policy parameters is an easy and effective way to reduce training times and improve convergence properties.

5.2.1.4 Critics

Each agent's critic network is modeled as a Multi-Layer Perceptron (MLP) identical in shape and architecture to the agent policy networks. Following the results and analysis of [41], this thesis employs decentralized critic networks that estimate the state-value function using local agent observation-action histories rather than global state-action histories. As centralized critics are able to distinguish between team action combinations, they are able to make more accurate state-value estimations, however this reduction in bias comes at the expense of higher variance due to sharper state-value estimates [41]. Decentralized critics instead only have access to the local agent's action and must average over the stochastic possibilities of each other agent's action, resulting in reduced variance at the cost of increased bias towards the local observation history [41]. As 3D Cooperative Navigation imposes fairly minimal agent interaction requirements for optimal policies, the reductions to variance provided by a decentralized critic provide significant benefit over the more accurate state-value estimates provided by centralized critics.

5.2.1.5 Reward Function

As with any reinforcement learning task, the employed reward function is critically important to effective learning. The reward function designed for the 3D Cooperative Navigation task is composed of five terms, as tabulated in the following Table 5.1, with the total reward given to each agent being the sum of each of that agent's five reward terms as per the following:

$$r_t^n = r_{1,t}^n + r_{2,t}^n + r_{3,t}^n + r_{4,t}^n + r_{5,t}^n \tag{5.1}$$

where r_t^n is the total reward given to agent n at timestep t and $r_{i,t}^n$ for $i = 1, 2, ..., 5$ are the five reward terms for agent n at timestep t. Note that the magnitude each of the reward function terms are rather small such that $r_{i,t}^n \in [-2.75, 1.35]$. While range of optimal reward magnitudes is somewhat debated by the community, it was found that larger reward function scaling resulted

in training instability as large magnitude rewards forces gradient and PPO clipping for each policy update. This same reward function is utilized in each subsequent tasks, with minor variations as necessary to preserve partial observability and to allow for unassigned targets.

Table 5.1: Task 1 Reward Function Terms

Term	Name	Purpose	Min	Max	Equation
$r^n_{1,t}$	Collision Reward	Punish collisions	-1	0	$\begin{cases} 0 & \text{no collision} \\ -1 & \text{collision} \end{cases}$
$r^n_{2,t}$	Neighbor Distance Reward	Avoid collision	-0.36	0	$-0.01(6 - \min(\text{NAD}, 6))^2$
$r^n_{3,t}$	Goal Distance Reward	Reward proximity to target	0	0.6	$0.01(10 - \min(\text{GD}, 10))^{1.7}$
$r^n_{4,t}$	Directional Velocity Reward	Encourage motion towards target	-0.75	0.75	$0.05 \left\|\text{vel}\right\|_2 (1 - \left\| \frac{\text{vel}}{\|\text{vel}\|} - \frac{\text{GD}}{\|\text{GD}\|} \right\|_2)$
$r^n_{5,t}$	Out of Bounds Reward	Punish going out of bounds	-1	0	$\begin{cases} 0 & \text{in bounds} \\ -1 & \text{out of bounds} \end{cases}$

NAD = Nearest Agent Distance [meters]
GD = Goal Distance [meters]
vel = Agent XYZ Velocity Vector [meters/second]

Significant efforts were spent iteratively refining and balancing each reward term to improve learnability. For example, the exponents on the second and third reward term have been balanced such that the reward gradient disincentives movement towards other agents even if that movement brings the agent closer to the target location, teaching agents to avoid near-collisions as well as collisions. Further, fourth reward term for directional velocity has been scaled such that it provides minimal positive or negative reward for the typical velocities exhibited during training. The result is that agents will only receive an impactfully strong positive or negative velocity reward term is at the start of training when policies are near random, and late into training when agents are learning to navigate towards targets more rapidly.

5.2.1.6 IPPO Loss

The chosen learning method for Task 1 (as well as all subsequent tasks) was Independent Proximal Policy Optimization (IPPO) [18], which has been shown to outperform Mutli-Agent Proximal Policy Optimization (MAPPO) [18] on the majority of tasks, especially those that require minimal agent interactions for optimal policies such as 3D Cooperative Navigation [41]. The trade-offs of IPPO vs MAPPO are discussed further in Section 2.2.4.

Specifically, agents are trained using torchrl's ClipPPOLoss module, using torchrl's Generalized Advantage Estimation estimator module to provide state value estimates in the loss calculation. The hyperparameters for both of these modules are discussed further in the following Section 5.2.2. Finally, the Adam Optimizer was employed to apply gradient descent/ascent to the policy and critic network weights based on the Clipped IPPO loss.

5.2.2 Hyperparameters

As with most reinforcement learning tasks, the implemented version of cooperative navigation has a long laundry list of hyperparameters that must be carefully tuned to allow system convergence towards stable and effective policies. Table 5.2.2 overviews the various hyperparameters used in each studied task. The remainder of this subsection discusses each of the different hyperparams and why they were chosen or how they were tuned for their use in Task 1. For the hyperparams that varied between tasks, those discrepencies are discussed in the sections dedicated to those tasks.

- **Steps per Batch** - Total number of steps collected per data collection cycle. Needs to be sufficiently large to provide an adequate sampling of how the policy interacts with the environment, though the number of steps per batch is directly proportional to how long it takes to collect those samples. By trial and error, it was found that 4096 provided consistent, low variance policy updates. With a single simulation running with the hyperparameters in Table 5.2 around 40 minutes to collect 4096 samples. For a typical 30 iteration collection-training cycle, this results in a 20 hour runtime.

- **Max Steps Per Trace** - Maximum number of timesteps before the environment will automatically be reset. To encourage early exploration, this was kept quite low at 32 steps/trace, which was found to be approximately double the timesteps needed for optimal policies to cross the simulation area. With the relative velocity setpointing used for movement actions, random actions are strongly biased towards increasing an agent's velocity. As a result, agents lose control quickly during initial training. By restricting how long each mission can be, the agents center their learning on exploring the immediate area surrounding their spawn location, which complements the imposed curriculum learning schemes in each task.

- **Number of Epochs** - Number of learning steps conducted on each batch of data collected

during collection phases. Chosen to be 32, with the rationale of that decision discussed with minibatch size.

- **Minibatch Size** - Size of the minibatch used during each epoch in each learning phase. Chosen to be 256, which includes enough timesteps to ensure reliable gradient estimations while also limiting how greedily the training phase consumes the collected data. When multiplied with the number of epochs (32), this gives the total number of timesteps that are processed per learning phase: $32 * 256 = 8192 = 2 * 4096$. This configuration results in each collected sample being used twice in the learning phase.

- **Learning Rate** - Coefficient for policy update magnitudes. Used by the Adam optimizer. Initial configuration used 1e-4, though with sufficient Steps per Batch updates were consistent enough to allow 3e-4 learning rate, hastening convergence.

- **Max Gradient Norm** - Clips gradients to have a maximum magnitude of 1.0. This is commonly done in machine learning works to prevent too large of a model update. It was configured during initial difficulties with model convergence and kept in after those difficulties were resolved.

- **PPO Clip Epsilon** - Maximum policy change ratio used in Clipped Proximal Policy Optimization Loss. The default value of 0.2 was employed, following [18].

- **GAE Gamma** - Discount factor used in Generalized Advantage Estimation. Larger values result in advantage estimates based more strongly on expected return from future states. Smaller values focus on the more immediate reward from an action. Typical values are [0.8, 0.9997]. As the reward function was designed such that the best action at a given state typically does not depend significantly on future actions (in general, simply moving towards a target location is sufficient), the somewhat low value of 0.95 was employed initially and then later reduced to 0.90 for Task 2 and Task 3. This lower value reduced agent look-ahead to allow for faster training, dramatically improving convergence properties as compared to using 0.95 or 0.99.

- **GAE Lambda** - Generalized Advantage Estimation bias-variance knob. Larger values tune the approximation towards higher variance, lower values towards higher bias. Typical values range in [0.9, 1]. This value was chosen as a middle-ground 0.95, which was experimentally

determined to be effective at Task 1. Larger values resulted in unsteady training, smaller values appeared to reduce training effectiveness.

- **PPO Entropy** - Coefficient for optional entropy term in PPO loss. Larger values encourage more exploration. Followed [18] and used 1e-3 for Task 1. Unused in Task 2 and 3 due to limitations of torchrl.

- **Share Policy Params** - Determines whether separate policy networks are trained for each agent or if one policy network is updated with every agent's experiences. This was set to True to enable more rapid and stable training, at the cost that heterogeneous team behaviors may not be easily learnable with homogeneous agent policies.

- **Policy Activation Func** - Activation function used at the output of each layer of the policy neural networks. The industry standard ReLU was used.

- **Policy Network Depth** - Number of hidden layers in the policy nerual networks. Sufficient output performance and training efficiency was found with four hidden layers.

- **Policy Network Width** - Number of neurons per hidden layer in the policy nerual networks. Sufficient output performance and training efficiency was found with 256 neurons per hidden layer.

- **Share Critic Params** - Determines whether separate critic networks are trained for each agent (each only able to access that local agent's observation history) or if one global critic network provides state-value estimates for each agent. This was set to True to enable more rapid and stable training. As each agent has the same observation structure and objective function, sharing critic parameters allows the critic to aggregate reward estimates from each agent, dramatically improving training time. Several other works including [18, 41] agree that decentralized critics provide better convergence properties than centralied ones for tasks with minimal inter-agent interactions, such as the 3D Cooperative Navigation task.

- **Critic Activation Func** - Activation function used at the output of each layer of the critic neural networks. The industry standard ReLU was used.

- **Critic Network Depth** - Number of hidden layers in the critic nerual networks. Sufficient output performance and training efficiency was found with four hidden layers.

- **Critic Network Width** - Number of neurons per hidden layer in the policy nerual networks. Sufficient output performance and training efficiency was found with 256 neurons per hidden layer.

- **Optimizer** - Optimizer that manages how neural network weights are updated for a given loss gradient. The commonly used Adam optimizer was employed.

- **Loss Function** - The function used to map policy network outputs, critic network outputs, and rewards into loss gradients to be used by the optimizer for updating policy weights. Following [18], ClippedPPOLoss was used with decentralized critic networks, also known as Independent Proximal Policy Optimization (IPPO). Per [18] and [78], this decentralized training architecture often outperforms centralized ones, especially on similar tasks where optimal policies require minimal inter-agent interaction.

- **Value Estimator** - The algorithm used to estimate the value of states. Used Generalized Advantage Estimation (GAE) [55] for its ability to tune the bias-variance tradeoff. See section 2.1.4 for more information on GAE hyperparameters.

- **Simulation Bounds [m]** - Side lengths for the rectangular prism shaped environment that simulated agents are restricted to, in simulation coordinates. The number of agents and the volume of the simulation must be balanced appropriately, as increasing the agent density drastically increases the task difficulty. Too large of a simulation will not allow agents to learn collision avoidance, too small will hinder learning. 40 m x 40 m x 40 m was found to be a good middle-point for 5 agents with uniformly distributed agents and targets.

- **Max Velocity** - Maximum velocity that agent velocity setpoints can be. Attempts to increase the setpoint past this value will clip the setpoint to the set maximum velocity. 20 m/s was found to be sufficiently high that it never actually clipped any movement actions, as the simulation space is only 40 m across, the velocity step size is 1 m/s/step, and steps only occur at roughly 2 Hz.

- **Velocity Step Size** - How large each change to an agent's relative velocity setpoint is per action. This value corresponds to how rapidly agents are able to accelerate. Larger values allow for more nimble, faster agents at the expense of movement precision. 1 m/s/step was

found to work well with the rate of actions, which (calculated from the Simulation Frame Rate and Frames per Step) is about 2 Hz.

- **Simulation Frame Rate** - How many physics engine updates per second of clock time. This value is locked at 120 Hz by Unreal Engine, though notably degrades if the application is not focused or if the computer running the application is unable to keep up with the computation rate. Can be modified in effect by altering the ratio of clock time to simulation time, though such changes were found to have other negative effects as discussed further in Appendix A. Such efforts were not pursued further than initial experimentation.

- **Frames per Step** - Number of simulation frames before a new action is given to agents. Chosen to be 50 due to initial belief that the simulation frame rate was 100 Hz, resulting in an action rate of 2 Hz. Once it was discovered that the simulation frame rate was actually 120 Hz, the Frames per Step was never corrected as agents seemed to be training effectively at 50 frames per action.

5.2.3 Optimizing Training Performance on Task 1

This section provides training outputs for different training configurations and curriculum learning schemes used to solve Task 1, including analysis of the learned behaviors. Further conclusions and implications of the included results are discussed in Section 6.

5.2.3.1 Uniformly Distributed Initialization

Initial attempts at training three agents towards Task 1 utilized uniformly distributed targets and agents within the simulation bounds. However, this configuration suffered from severe gradient instability. Agent policies updates suffered from high variance and failed to converged towards behaviors that satisfied the task within the allowed 100 episodes. Despite various efforts restructuring the reward function, allowing additional training, and significant hyperparameter tuning, the three agent system did not produce reliable policy updates towards desired behavior. Figure 5.3 shows the global reward averaged over each episode for one such 100 epoch training runs. As can be seen, high variance policy updates resulted in high training instability. Note also that the vertical reward axis is not normalized with respect to episode duration and thus shows five digit reward values.

Table 5.2: Hyperparameter List

Hyperparameter	**Task 1 Value**	**Task 2 Value**	**Task 3 Value**
Steps per Batch	4096		
Max Steps Per Trace	32		
Number of Epochs[a]	32		
Minibatch Size	256		
Learning Rate	3e-4		
Max Gradient Norm	1.0		
PPO Clip Epsilon	0.2		
GAE Gamma	0.95	0.90	
GAE Lambda	0.95		
PPO Entropy	1e-3	entropy disabled	
Share Policy Params	True		
Policy Activation Func	ReLU		
Policy Network Depth	4		
Policy Network Width	256		
Share Critic Params	True		
Critic Activation Func	ReLU		
Critic Network Depth	4		
Critic Network Width	256		
Optimizer	Adam		
Loss Function	ClipPPOLoss		
Value Estimator	GAE		
Simulation Bounds [m]	$40 \times 40 \times 40$		
Max Velocity	20 m/s		
Velocity Step Size	1 m/s		
Simulation Frame Rate	120 Hz		
Frames per Step	50		

[a] *Epochs* refers to the number of minibatch policy updates performed per collection-training episode.

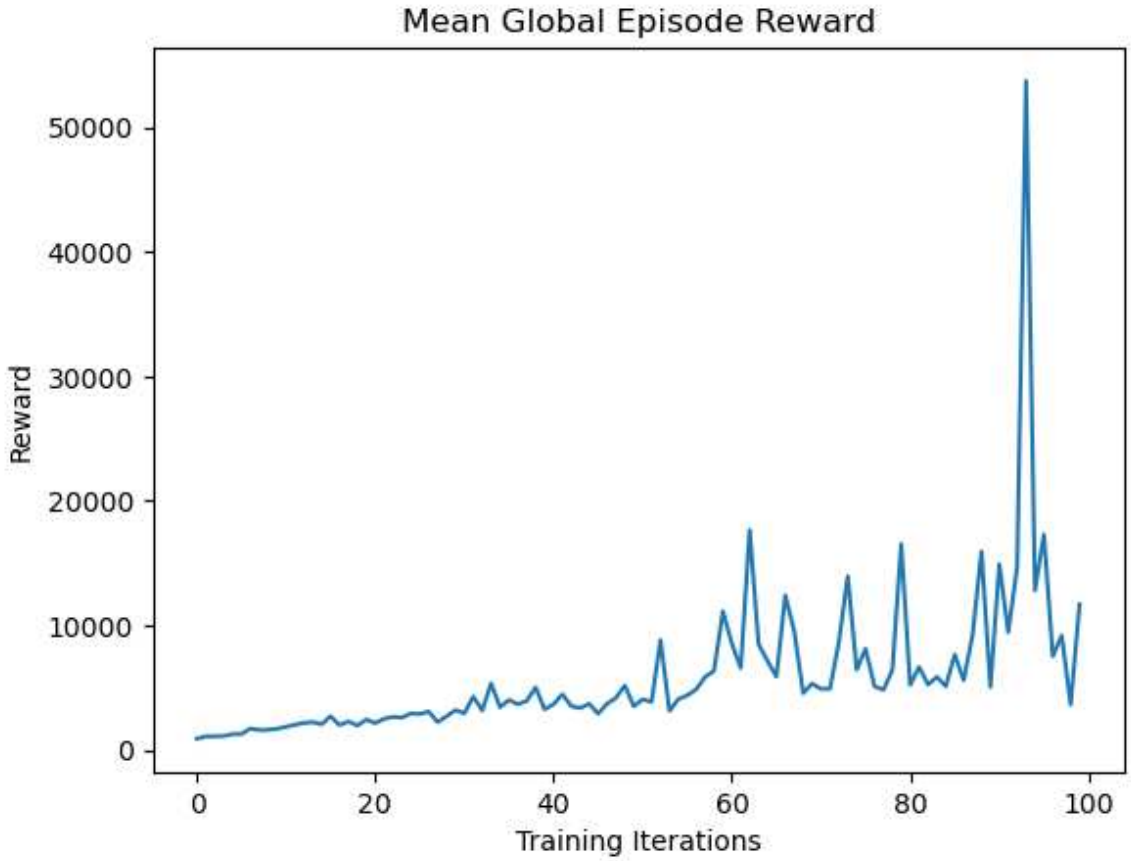


Figure 5.3: Global Return Average For Three Agent 3D Cooperative Navigation with Uniformly Distributed Initializations. Severe return gradient variance prevents effective training.

This is corrected in future plots. With the un-normalized reward, optimal policies should result in approximately 600,000 reward per Episode.

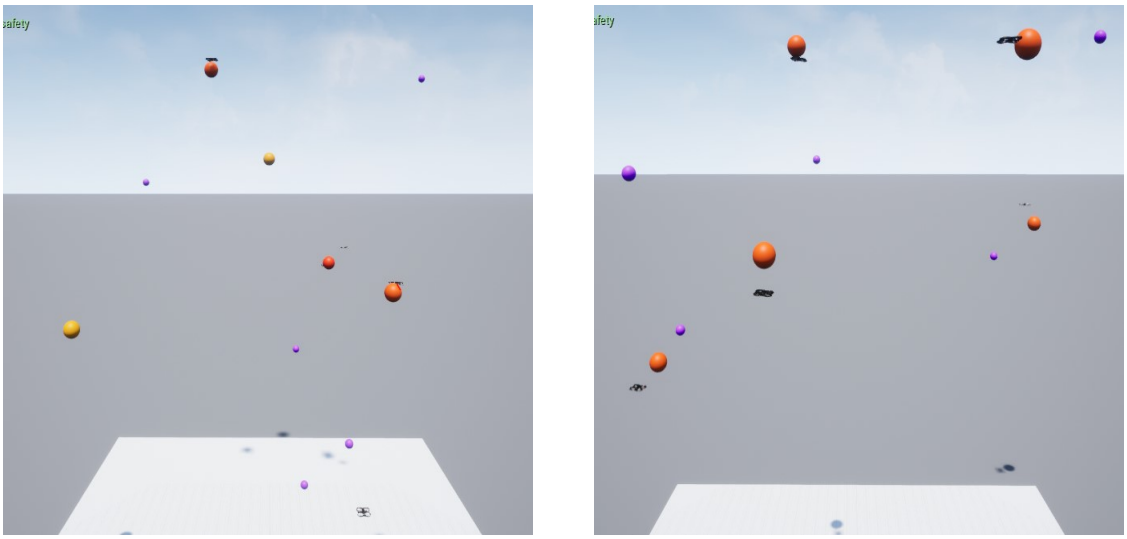
It was observed during this preliminary unstable training that the vast majority of time during data collection was spent 1) resetting the environment after a collision or agent going out of bounds and 2) with agents flying in seemingly unmotivated directions. The temporal overhead associated by resetting was lessened by overriding AirSim's default command to reset a simulation with custom code that employed agent teleportation rather than flight controller state resets. The details of this improvement are detailed in Appendix A. The second phenomenon appears to have been an artifact of the relative velocity setpointing system used to execute agent actions. As each action increases or decreases a velocity setpoint, fully random policies are significantly biased towards increasing the agent's speed. Further, with uniformly distributed targets and agents in a sufficiently

large space, linear paths in random directions have a low probability of ever moving close enough to a target location to obtain a large positive reward. The resulting negative feedback loop results in agents that are unable to sufficiently explore the space through random actions alone. This problem is especially severe in a fully three dimensional space. Similar works generally restrict the problem to two dimensions [80, 75, 51] or utilize a position-based action system [79, 37, 38, 39] that, while less realistic, is more resilient to this vicious cycle limiting exploration.

5.2.3.2 Enhancing Early Exploration with Curriculum Learning

When presented with the problem of an agent failing to learn in a task that is too difficult, there are two approaches to lessen the skill-gap: increasing the agent's capacity to match the task or decreasing the task's difficulty to match the agent. Generally, the optimal solution in terms of final policy performance is some combination of the two. The previous subsection discussed several methods for increasing agent capacity – tuning hyperparameters, optimizing how the agent effects or observes the environment, and improving system efficiency to allow for more training steps. This subsection instead focuses on improving performance by instead reducing task difficulty to match the agent's capabilities, then gradually increasing task difficulty back to the original, previously unsolvable level. This process is referred to in the reinforcement learning community as *Curriculum Learning* [20] or *Auto-Curriculum Learning* [22, 26] when the difficulty is automatically adjusted to match the agents' abilities.

For Task 1's 3D Cooperative Navigation, the primary difficulty of the task originates from the low probability of random actions receiving positive reward. This is caused by uniformly distributed targets and agents spread across a large three dimensional space. By varying the average distance between an agent and their target in the space, one can effectively vary the difficulty of the task. Further, it is expected that once an agent learns to navigate towards a specific target (represented by a relative XYZ position value in that agent's global observation), that agent will be able to learn from variations of Task 1 with any initial distance between agent and target. The vicious cycle of restricted early exploration becomes a virtuous cycle as the agent learns to make a first action towards that agent's target, as the reward density increases with proximity to their target. Thus, the first and only necessary curriculum learning step for Task 1 is to modify the target location initialization distribution to be at a random location within some sphere surrounding that agent's initialization. This range was chosen to be between 5.0 and 7.5 meters from the agent, falling well



(a) Uniformly Distributed Agents and Targets

(b) Curriculum Learning Distributed Agents and Targets

Figure 5.4: Comparison of Uniform Target Distribution vs Curriculum Learning Target Distribution. Purple balls show agent initialization points, yellow/orange balls show target locations. The curriculum learning distribution in (b) has targets initialized 5-10 meters from the agent initializations.

within the 10 meter range at which the agent begins receiving positive reward from target proximity, as defined in Table 5.1. Figure 5.4 depicts the system with a uniform target distribution compared against the 5.0 - 7.5 meter distribution.

Under the easier curriculum learning task, the three Agent team rapidly learns an effective policy, resulting in the smooth learning curves shown in Figure 5.5. The top left plot shows the average global reward averaged across each episode, showing a steady, gradual increase as the agents trained. This smooth plot indicates low variance policy updates and agents that are well suited to thir given task. The second plot (top row) shows the episode and agent averaged collision reward for an episode (axis is negative). This curve rapidly plateaus to zero, indicating that agents learned to completely avoid collisions with other agents. The third and fourth plots (still top row) should be interpreted in tandem. The third shows the episode and agent averaged reward penalty for proximity to other agents, the fourth shows the positive reward for proximity to the designated target location. The target proximity reward plot shows a gradual increase, indicating that agents learned how to navigate towards their targets quickly and then hover near those targets for the remainder of the episode. The third plot shows that agents learned to get closer to other agents, which incurred larger

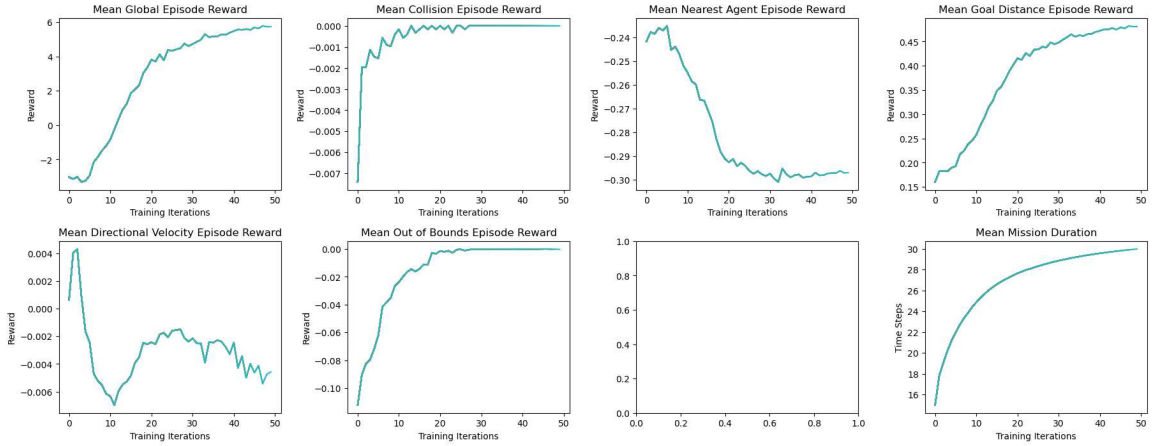


Figure 5.5: Task 1 Three Agent training curves with (5-7.5) meter target distance. Trained for 50 Episodes. The top left plot shows averaged global reward. The next five (left to right) show the five reward terms individually. The bottom right plot shows average mission duration.

negative rewards. As the benefit from proximity to a target location outweighed the negative from proximity to another agent, agents learned to prioritize getting close to their targets.

The second, third, and fourth plot together highlight the importance of a well balanced reward function. Agents typically learn the largest positive and negative reward values first, then the lesser reward terms. As seen in the Figure 5.5 plots, agents first learned to prioritize collision avoidance and to avoid going out of bounds (plot six, second from the left on the bottom row), then learned to next prioritize navigate to their target locations, and finally learned to do so while maximizing distance to other agents. Plot five (bottom row, far left) shows the reward term for velocity towards the target location. While this plot appears quite sharp and unstable, it should be noted that the scale of the plot is magnitudes smaller than other reward terms. As agents hover at target locations for the majority of well-trained missions, this velocity term really only impacts the initial timesteps of moving to a target location (and any overshooting necessary to stay at that target – see Section 5.2.1.1). The final eighth training plot (bottom right) shows the average mission duration vs training episodes. As the only ways for an episode to end early are for agents to collide or for agents to go out of bounds, this plot's smooth upward arc indicates a similarly smooth learning of simulation bounds and collision avoidance. Visual inspection of the team's learned performance confirmed the graphed findings, with agents effectively moving directly towards target locations and then hovering until the end of the episode.

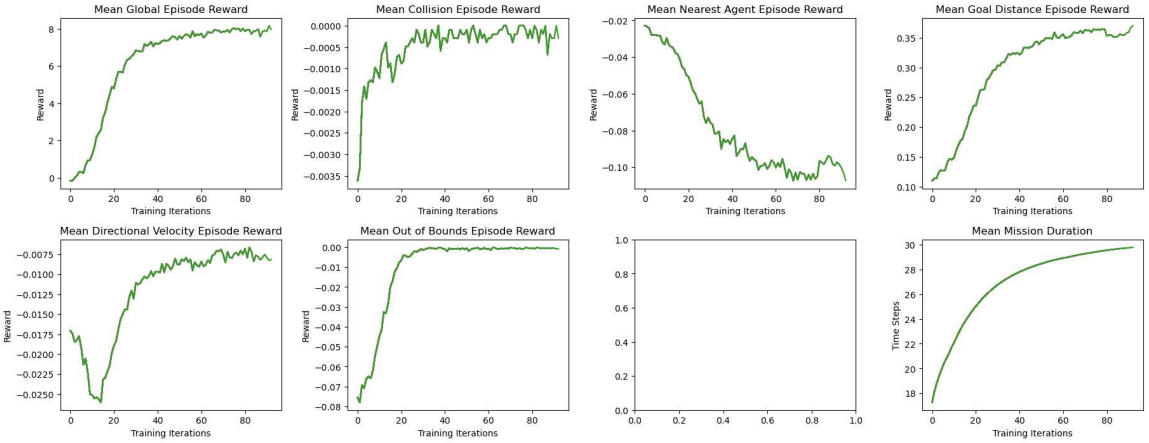


Figure 5.6: Task 1 Five Agent training curves with (5-7.5) meter target distance. Trained for 90 Episodes. The top left plot shows averaged global reward. The next five (left to right) show the five reward terms individually. The bottom right plot shows average mission duration.

5.2.3.3 Five Agent Curriculum Learning

The success of training Three agents on the curriculum learning task begged the question of how many agents were capable of learning this easier task at a time. Scalability is a large issue in Multi-Robot Systems, with system state space, multi-agent variance, and multi-observation variance each increasing exponentially with the number of agents [41]. Thus, two additional agents were added to the system and training was repeated, producing the training curves displayed in Figure 5.6.

The five agent team trained and performed quite similarly to the three agent team, with both sets of training curves exhibiting near identical shapes. However there is a noticeably larger variance present in the spikier training curves in the five agent case shown in Figure 5.6, indicating slightly less stable training. Another significant change is that the five agent team never fully learns to avoid collisions, as seen in the second plot showing the collision reward penalty term, which never quite plateaus to zero. In addition to the increased number of agents increasing the scale of the problem, they also increase the agent density, as the simulation space was not increased to accommodate the extra agents. As the five agent team was able to learn to navigate to their targets but not fully learn collision avoidance, it was believed that the task and learning methods were well suited to five agents. As training teams takes very significant real world time investment (approximately 40 minutes of clock time per episode), further scalability assessments were reserved for more interesting tasks, as discussed later in this thesis.

5.3 Task 2: Partially Observable 3D Cooperative Navigation

5.3.1 Expansions on Task 1

Task 1 served to validate that the developed curriculum learning scheme is capable of training fully three dimensional cooperative navigation teams in AirSim under idealistic conditions. However, the resulting teams' policies lack applicability to real world tasks that impose partial observability, sensor errors and delays, dynamic environments with obstacles, localization error, and other real-world challenges. For the eventual goal of developing a learnable way for a team of aerial agents to autonomously detect, observe, and potentially suppress incipient stage wildfires, the Task 1 teams are insufficient. As suppression and observation are predicated on capable detection, Task 2 focuses specifically on training a team of aerial agents capable of detecting a set of unknown targets in a partially observable setting.

Task 2 has two key differences from Task 1. Firstly, Task 2 imposes partial observability with a basic binary distance filter. The position and velocity of agents and targets within the set observation radius are fully visible while those outside the radius are not. While not entirely realistic, modeling partial observability in this manner forces agents to learn behaviors for exploration and encourages team-oriented behaviors. Task 3 delves deeper into the cooperation required for partial observability by implementing an inter-agent communication network. The second key difference is that the targets in Task 2 are no longer specific to any one agent. Any agent is able to receive a positive reward for being in close proximity to any reward location. This better mirrors real world observation tasks, where typically and agent is capable of monitoring each target with diminishing returns for additional agents monitoring the same target. Table 5.3 summarizes the key properties of the different 3D Cooperative Navigation Tasks variants that are studied in this thesis.

Table 5.3: 3D Cooperative Navigation Task Comparison

Property	**Task 1**	**Task 2**	**Task 3**
Individual Targets	True	False	False
Observation Distance [m]	∞	10	10
Num Communication Channels	0	0	5

Task 2 expands the set of necessary skills that a team needs to learn in order to satisfy the task. While Task 1 was solvable by agents capable of motivated flight towards a specific target while

avoiding collisions with other agents, Task 2 introduces the much more complicated task of target discovery. Without globally observable states, agents must now work together to identify targets and then coordinate which agent will fly to which target. Clearly, this task would be best suited by implementing some form of information sharing through communication between agents. Thus, Task 2 serves as a communication-less baseline to compare the against the results of Task 3, which implements a learnable communication scheme.

5.3.2 Learning Scheme

Task 2's environment and learning scheme are identical to Task 1, with modification made to enable partial observability and unassigned targets. The necessary modifications are discussed in the following subsections.

5.3.2.1 Observation Space

Task 2 modifies the base 3D Cooperative Navigation Observation Space in two key ways. Firstly, the positions and velocities of objects outside of the set observation distance of 10 meters are zero'd out. Objects within the radius of observation are considered fully visible, returning the same relative position and velocity information included for Task 1. The second change is to unassign targets such that any agent is rewarded for proximity to any target. To improve the learnability of the unassigned targets, any observable targets are sorted by distance from the agent such that the closest target to an agent appears in the same location in that agent's observation vector at each timestep. Applying these two changes, the new observation structure is as follows:

- X, Y, Z position of the agent in simulation coordinates [meters] (always visible)
- X, Y, Z velocity of the agent in simulation coordinates [meters/second] (always visible)
- For each other agent in the simulation (sorted by euclidean distance from the observing agent:
  - relative X, Y, Z position of the other agent in simulation coordinates if within 10 meters. Otherwise (0, 0, 0) [meters]
  - X, Y, Z velocity of the agent in simulation coordinates if within 10 meters. Otherwise (0, 0, 0) [meters/second]

- For each target location in the simulation (sorted by euclidean distance from the observing agent:
  - relative X, Y, Z position of the target in simulation coordinates if within 10 meters. Otherwise (0, 0, 0) [meters]

5.3.2.2 Reward Function

The reward function remains largely unchanged from Task 1, with the exception that $r^n_{3,t}$, the goal distance reward term, is modified to use the nearest target within the observation radius rather than the agent's assigned target. The remaining unchanged reward terms and their equations are listed in Table 5.1.

5.3.3 Hyperparameters

Only two hyperparameters were changed between Task 1 and Task 2. Those changes are detailed below:

- **GAE Gamma** - Discount factor used in Generalized Advantage Estimation. Larger values result in advantage estimates based more strongly on expected return from future states. Smaller values focus on the more immediate reward from an action. This value was reduced from 0.95 to 0.90 for Task 2 and Task 3. This reduced agent look-ahead to allow for faster training, dramatically improving convergence properties on the more difficult tasks as compared to using 0.95 or 0.99.
- **PPO Entropy** - Coefficient for optional entropy term in PPO loss. Larger values encourage more exploration. Task 1 followed [18] and used 1e-3. Due to limitations of torchrl, the PPO entropy loss term was unused in Task 2 and Task 3. This limitation is discussed further in Appendix A.

5.3.4 Curriculum Learning Results

As this task is effectively a more difficult variant of Task 1, a similar though more involved curriculum learning method was implemented. While Task 1 is solvable with the capability to avoid collisions and the ability to fly towards a target, Task 2 adds the additional requirement of being

able to discover targets if there are not any within observation distance. While controlling the distance that target locations are able initialized with respect to agents provides a means to control the difficulty of flying towards targets, it does not provide suitable control over the difficulty in discovering new targets.

To provide this second axis of control, the initialization scheme for target locations is modified to allow multiple targets to spawn near some agents while others may have no nearby target. The percentage rate at which a target will be initialized at the "wrong" agent is referred to as the *Wrong Agent Rate*. The number of target locations is still kept equal to the number of agents, so if an agent spawns without a nearby target, it must explore the space until it finds one that may or may not be already occupied by another agent. Both agents are able to simultaneously receive a positive reward for being close to the target, though they will also receive a not insignificant punishment for being close to each other. This may encourage agents to search around for a target that is unoccupied rather than attempt to share with a teammate. Wrong Agent Rate also teaches agents about prioritizing targets. An agent may observe two similarly distanced target locations nearby, and would need to judge which one is optimal to fly towards. This is further complicated if there are any other agents within observation range. With this new knob available to increase a task's difficulty in target discovery, a fresh five agent team was trained using the three curriculum learning steps outlined in Table 5.4.

Table 5.4: Task 2 Curriculum Learning Steps

Curriculum Step Number	**Target Initialization Distance (min, max) [m]**	**Wrong Agent Rate**	**Number of Episodes**
#1	(1, 3)	0%	30
#2	(5, 10)	5%	30
#3	(5, 10)	20%	30

As Wrong Agent Rate refers to the percentage chance of each individual target spawning at a different agent than it would have normally, it is also valuable to consider the chance that every agent will be initialized near a target as such an initialization is identical to the initialization used in Task 1 curriculum learning. Table 5.5 lists this percentage for different numbers of agents, calculated using the geometric distribution in the following Equation 5.2

$$\% \text{ Chance} = (1 - \text{WAR})^N * 100 \tag{5.2}$$

where % Chance refers to the chance of every agent being initialized near a target, which is equivalent to the Task 1 curriculum learning initialization scheme, WAR is the wrong agent rate, and N is the number of agents.

Table 5.5: Wrong Agent Rate vs Chance of Task 1 Initialization

	Chance of Task 1 Initialization		
Wrong Agent Rate	**3 Agents**	**5 Agents**	**7 Agents**
0%	100%	100%	100%
5%	85.73%	77.37%	69.83%
10%	72.9%	59.04%	47.82%
15%	61.41%	44.37%	32.05%
20%	51.2%	32.76%	20.97%
40%	21.6%	7.77%	2.79%
60%	6.4%	1.02%	0.16%
80%	0.8%	0.03%	0%
100%	0%	0%	0%

Figure 5.7 shows the resulting training curves from the three-step curriculum learning process outlines in Table 5.4. For the first 30 training episodes under Curriculum Learning Step #1, the agents trained with 0% WAR and incredibly close target initializations. This task focused on teaching the team how to stay within the simulation bounds and how to hover at the closest nearby target location. Unsurprisingly, the sections of training figures for this step are very similar to those observed for Task 1. As the team transitioned to Curriculum Learning Step #2, with further targets and 5% WAR, there is a clear drop in average global reward. This drop is not completely a result of agents performing worse on the more challenging task, but also a result of missions requiring more timesteps spent navigating to target locations and less timesteps hovering at those locations. The most optimal performance on Curriculum Learning Step #2 would receive less reward than near optimal performance on Curriculum Learning Step #1. With the increase in WAR, it is seen that agents begin to learn exploration behaviors, as evidenced by the gradual rise in goal distance reward (top right plot) during episodes 30-60. It is also seen that increase in collisions as Curriculum Learning Step #2 continues, which is caused by agents attempting to share reward locations too closely, resulting in collisions.

Without any ability to communicate with one another, and with short mission durations, agents do not learn how to disperse such that each agent has their own individual target. As a result, when agents are spawned without a nearby target, they will roam and then hover at the first target

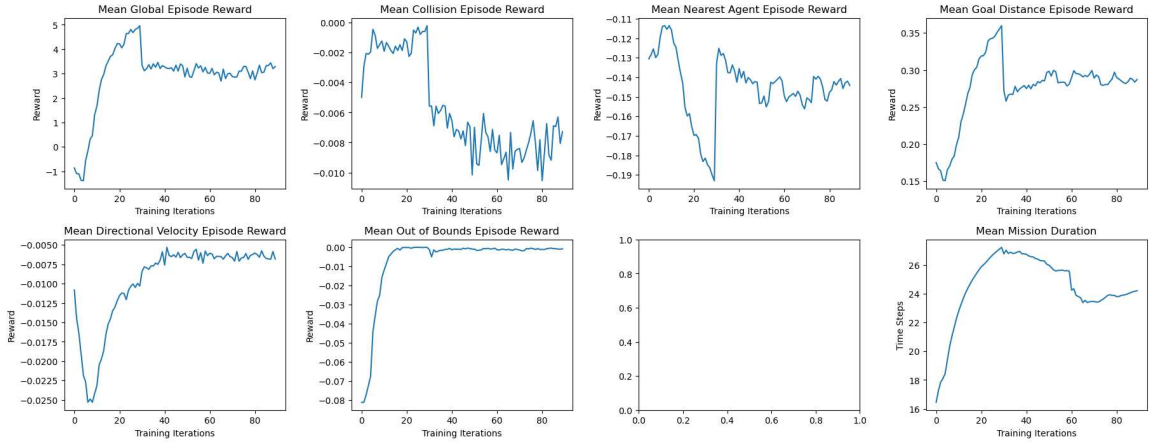


Figure 5.7: Task 2 Five Agent Team Curriculum learning training curves. Trained for 90 total Episodes,with changing curriculum every 30. The top left plot shows averaged global reward. The next five (left to right) show the five reward terms individually. The bottom right plot shows average mission duration.

they find, which is most often already occupied. With strong reward gradients encouraging both agents to attempt to get as close to the target as possible, collisions occur often. As the learning shifts to Curriculum Learning Step #3, which increases the WAR to 20%, we see that this problem worsens slightly, with the average mission duration (bottom left plot) taking a sharp decline at Episode 60. With the current communication-less configuration, the team appears unable to learn effective group exploration behaviors.

To provide a difficult evaluation of the behaviors learned during the curriculum learning steps, the trained team was then briefly fine-tuned on the task with uniformly distributed target and agent initializations. Further, the maximum mission duration was increased to 256 timesteps to provide ample time for navigation. The resulting training plot is shown in Figure 5.8. As seen in the bottom right plot that shows average mission duration, the teams are generally collide before the mission reaches the full 256 timesteps. This appears to be a result of the suboptimal target sharing behaviors that were learned previously. With uniform agent and target distributions, it is quite likely for multiple agents to find themselves attempting to navigate towards the same target location. This typically results in a collision within a few timesteps as each agent attempts to hover as close to the target as possible. After visually inspecting several training runs, this theory was confirmed.

The visual inspection revealed another sub-optimal behavior. When agents find themselves

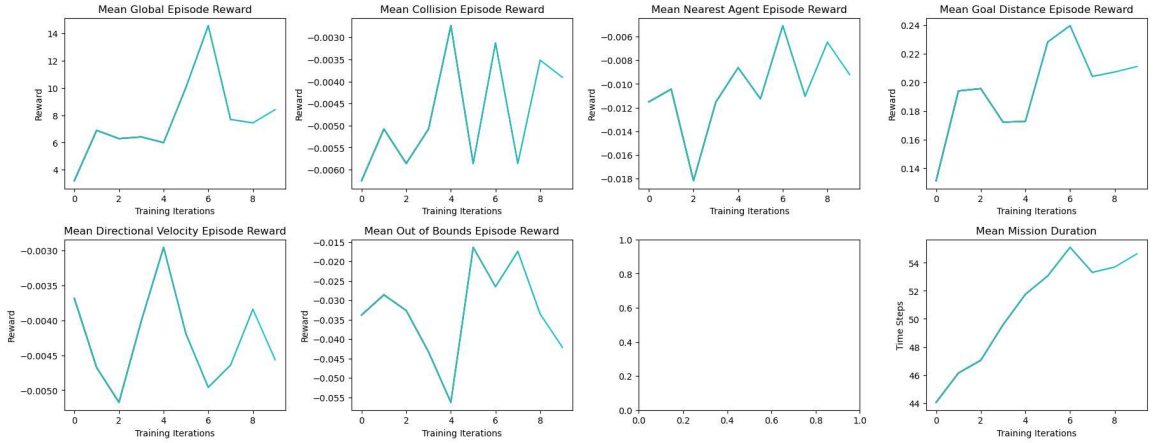


Figure 5.8: Task 2 Team Performance fine-tuned on Uniformly Distributed Targets and Agents. The maximum mission duration was increased from 32 to 256 to provide ample time for navigation. Reward terms are broken out into individual plots.

without a nearby target, they typically hover in place with slow lateral movement, generally towards the center of the simulation. The expected more optimized behavior would be for them to search around the environment until they find an unoccupied target. It is believed that this behavior is likely caused by the current method of zeroing out observation vector fields for objects that are outside the radius of observability. With the outlined curriculum learning steps, it is quite likely that the policy networks are trained to focus almost entirely on navigating in the direction of the nearest target location. With that nearest target location being zeroed out, the associated network sections are also zeroed out and unable to contribute meaningfully towards actions. Further training on curriculum learning tasks may train away from this suboptimal hovering policy, though without any capacity for memory, agents may not be able to ever achieve fully optimal target discovery behavior.

5.4 Task 3: Partially Observable 3D Cooperative Navigation With Networked Agents

5.4.1 Expansions on Task 2

Task 3 includes the same partial observability and unassigned targets as from Task 2, with an added learnable inter-agent communication network. The various specifics of this implementation

are discussed in the following subsections.

5.4.1.1 Action Space

The action space of Task 3 was expanded to include multiple communication actions. In addition to the discreet movement action, agents provide several floating point values that are communicated to other agents on the next time step as a part of their observations. When determining how many values each agent should be able to communicate to other agents each timestep, there emerges a balancing act between ensuring that agents have sufficient capacity to communicate necessary information while also maintaining a sufficiently small action and observation space to enable effective training. A large number of communication channels would require a similarly large agent policy and critic network to adequately commute actions and value from the increased observation lengths. To this end, the number of communication channels was initially set to be equal to the number of agents. Thus, the revised action space is as follows (new items are in **bold**):

- Movement Action – Discreet value within [1, 7] corresponding to a change in the agent's relative velocity setpoint.
- **Communication Actions** – For each agent in the simulation, append a continuous real-valued number that is communicated to every other at the start of the following timestep.

5.4.1.2 Observation Space

The Task 3 observation space is expanded to accommodate the added communication actions, resulting in the following observation space structure (new items are in **bold**):

- X, Y, Z position of the agent in simulation coordinates [meters] (always visible)
- X, Y, Z velocity of the agent in simulation coordinates [meters/second] (always visible)
- For each other agent in the simulation (sorted by euclidean distance from the observing agent:
  - relative X, Y, Z position of the other agent in simulation coordinates if within 10 meters. Otherwise (0, 0, 0) [meters]
  - X, Y, Z velocity of the agent in simulation coordinates if within 10 meters. Otherwise (0, 0, 0) [meters/second]

  - **Communication vector from that agent with length equal to the number of agents**

- For each target location in the simulation (sorted by euclidean distance from the observing agent:
  - relative X, Y, Z position of the target in simulation coordinates if within 10 meters. Otherwise (0, 0, 0) [meters]

5.4.1.3 Actors

Implementing multiple action types in torchrl version 0.3.0 is not officially supported, requiring several workarounds to implement with computable loss gradients. These different implementation-specific workarounds are discussed further in Appendix A. One key result of these limitations is that each communication action must be modeled by some distribution through a stochastic actor. As such, each communicated value is sampled from a Normal distribution where the average and variance of the distribution are each additional outputs from the policy network. That is, for every value that the agent communicates, it requires two separate neural network outputs. The effect of this distribution sampling is that there will be some inherent stochasticity to communicated values, especially so during early training. It is expected that after sufficient learning, agents will learn to minimize the output Gaussian variance such that the network may communicate with higher precision. Future system evaluations may fix the communication variance parameter to some set value, modeling a communications channel with some Gaussian error present in the communicated values.

5.4.1.4 Loss Function

To make the communication values learnable, a second Clipped PPO Loss module is added to the learning loop. For a given batch of collected trajectories, the clipped IPPO loss is calculated for movement actions and then a second clipped IPPO loss term is calculated for communication actions. Neural network weights are updated with a step from the Adam optimizer based on the sum of both action and communication loss gradients.

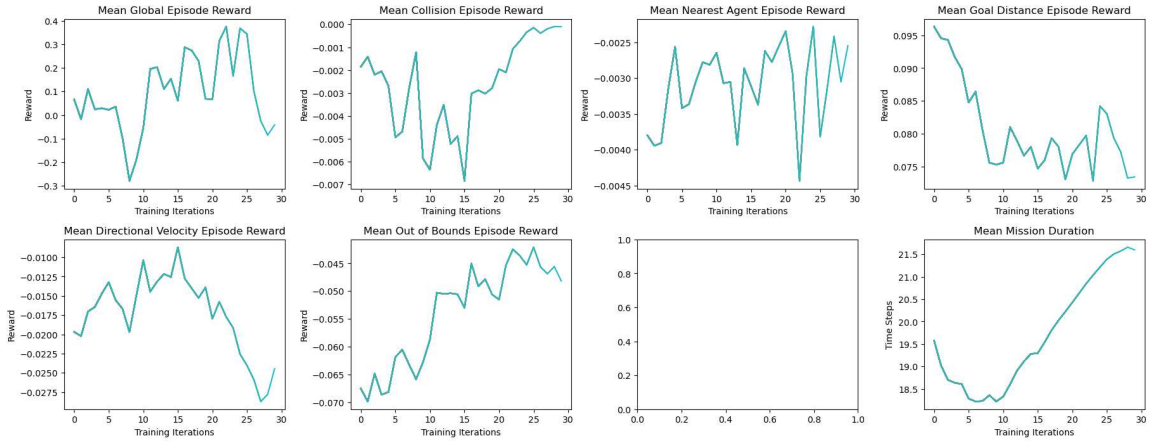


Figure 5.9: Task 3 Initial Training Failure. Trained task 2 team with enabled communication network was trained on Task 2 Curriculum Learning Step #3. Plots show near-random actions.

5.4.2 Training Performance

Initially, the trained team from Task 2 simply had its communication network enabled. However, as the communication network constitutes a major portion of agent policy networks, these efforts were met with failure. The resulting training curve when training the Task 2 team on Task 2 Curriculum Learning Step # 3 is plotted in Figure 5.9. As can be seen in said figure, enabling the untrained communication network resulted in catastrophic forgetting and near-random actions. The team was unable to relearn any desired behaviors on Curriculum Learning Step #3, which highlights the importance of first training on easier task to enable learning on more difficult ones.

Without a pre-trained team with communication capabilities, a new team was trained using the curriculum learning steps from Task 2 (see Table 5.4), with similar curriculum transitions every 30 epochs. The resulting training plots can be seen in Figure 5.10.

Reviewing Figure 5.10, there is a clear performance increase when providing teams with interagent communications. Most notably, the team exhibits improved collision avoidance behaviors as evidenced by plot 2's (top, second from right) reduced collision penalty and plot 3's (top, third from right) penalty for proximity to other agents. This indicates that the team is generally staying further away from one another, resulting in less collisions. This is further confirmed by visual inspection of team performance, indicating that agents shared objectives less often and, when they did share objectives, tended to occupy opposing sides of the target.

The Task 3's higher average reward (plot 1, top left) results from improved collision avoid-

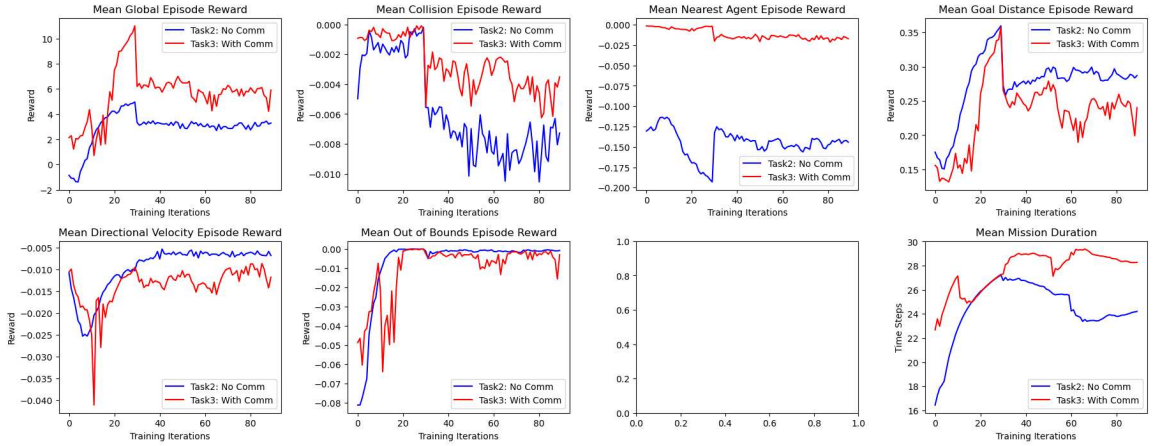


Figure 5.10: Task 3 Training Performance Vs. Task 2 Training Performance, using the curriculum learning steps outlined in Table 5.4. The Task 2 team (blue) does not have interagent communication while the Task 3 team (red) does.

ance rather than improved target proximity. In fact, the team appears to get slightly worse at occupying targets, as indicated by plot 4 (top right), which shows the reward term for proximity to target. This is expected, as agents that avoid sharing a target will need to travel further to get to an unoccupied one. This trend is backed up by the increased average mission duration in plot 8 (bottom right), which shows the Task 3 team episodes last notably longer. This is a direct result of the improved collision avoidance (both through reduced target sharing and improved pathing) that interagent communication provides.

It should be noted that adding interagent communications is not without downsides. The Task 3 training curves are significantly more jagged and variable than the Task 2 curves, which indicates less stable training. This indicates the importance of appropriately tuning the hyperparameter for how many floating point values each agent is able to communicate with other agents at each timestep. Too many values increases observation and action space dimensionality, hindering training. Too few values limits agent communications and prevents the learning of necessary coordination behaviors. Further work may investigate training performance as the number of communicated values changes, expecting to find some optimal value that balances policy update variance with task performance.

Chapter 6

Discussion and Conclusions

6.1 Summary and Discussion of Simulated Results

6.1.1 Task 1

6.1.1.1 Summary

Task one requires a team of agents to navigate to individual targets with global state observability. Initially, teams were unable to effectively learn at the task due to insufficient early exploration of the environment in a negative feedback loop. A curriculum learning scheme was created, parameterizing the environment to control the difficulty of finding targets for agents. By first training on a simpler subtask where agents are initialized in close proximity to their targets, agents learned effective policies for navigating towards close targets and then hovering at those targets for the remainder of the episode.

After first training on the easier curriculum learning task, teams showed impressive performance on the original task with uniform agent and target distributions, indicating that the easier curriculum learning task taught sufficient behaviors to enable learning on the more difficult original task. Notably, training on the curriculum learning task resulted in consistent policy improvements with low variance in state-value estimates. Teams were able to learn complete collision avoidance, with the later half of training iterations showing zero collisions. Essentially, a drone that is first taught how to hover is better equipped to then learn collision avoidance than a randomly initialized agent.

6.1.1.2 Implications

With individually assigned objective locations and global observability, Task 1 aligns with the real world task of arranging a team of UAVs into desired locations with automated collision avoidance. This could be achieved using the learned Task 1 policies. A sufficiently configured centralized coordinator could manage communication with agents to provide accurate positions and velocities, emulating global state observability. By sending each agent a series of waypoints, the central coordinator could arrange agents into any desired configuration. As the agents were trained with uniform agent initializations, the policies should generalize effectively to any 3D pattern or orientation.

Specific to wildfire management, a team of UAVs might detect a fire and need to move into specific positions to fully observe burn progression or for organized suppression efforts, like a sequential deployment of chemical suppressant onto a certain section of fire. A central coordinator (or some leader-follower drone scheme) would allow the precise positioning in an arbitrary fully three dimensional space. While the learned policies indicated full collision avoidance in the simulated results, there was not any testing conducted to evaluate collision avoidance on challenging hand-picked navigation operations. Future work may include such evaluations.

6.1.2 Task 2

6.1.2.1 Summary

Task two focuses on 3D Cooperative Navigation with partial observability and unassigned target locations. Agents must learn the same navigation and collision avoidance behaviors as in Task 1, with the addition of learning to first discover targets. To provide an additional knob of control on the difficulty of target discovery, a new parameter was added to environment initialization: Wrong Agent Rate. With this, targets were sometimes initialized with a different agent than they would have been using Task 1 initialization. That is, some agents may be spawned in close proximity to two or three targets while others may be spawned without a target within observation radius. As such, a higher Wrong Agent Rate would force agents to learn behaviors for target discovery.

Using the two curriculum learning parameters, a three step curriculum learning plan was created, aiming to sequentially train agents in navigating towards the nearest target, collision avoidance, and target discovery. Notably, agents had no means of communicating with each other. As

such, agents without an observed target would wander generally towards the center of the simulation until a target was spotted, after which they would quickly fly towards that target. More often than not, this resulted in the suboptimal behavior of target sharing in which two agents aim to observe the same target. Both agents receive a negative reward for proximity to each other, which should train agents to prioritize finding unique targets.

6.1.2.2 Implications

The curriculum learning trained team was evaluated on Task 2 with uniformly distributed agents and targets, validating the curriculum learning methodology. The team displayed two key suboptimal behaviors: target sharing and slow searching for new targets. However, the team was able to continue learning and performing on the uniformly distributed task where a randomly initialized team would fail. Thus, the curriculum learning methods proved effective at training agents towards fully three dimensional cooperative navigation with partial observability. With continued training on the uniformly distributed task or with additional curriculum learning steps, it is expected that agents would be able to improve policies away from the previously discussed suboptimalities. However, without systems for agent memory or interagent communication, optimal team-based target discovery is likely out of reach.

6.1.3 Task 3

Task 3 is identical as Task 2, with an added learnable inter-agent communication network.

6.1.4 Summary

Initial attempts to enable the dormant communication network of the trained Task 2 team resulted in near-random action selection, as if policy networks were freshly created. This, in retrospect, is because they effectively were. The combined movement and communication policy network was quickly overwhelmed by such a large section suddenly being enabled, immediately devolving into chaos. The initial failed training efforts served as a potent reminder of the need for curriculum learning to enable training on 3D Cooperative Navigation.

Subsequent efforts at retraining the network from scratch proved quite successful, with the Task 3 team's communication network allowing them to surpass the Task 2 team in all studied met-

rics, as was expected. Most notably, the interagent communication network reduced target sharing, with agents displaying an inclination towards unoccupied targets when feasible. It is expected that continued training with longer mission durations would eventually fully dissuade target sharing. The learned reduction in target sharing resulted in a corresponding decrease in collisions between agents, further highlighting the value of interagent communication.

6.1.5 Implications

Interagent communication is an essential part of real-world cooperative UAV teams. Effective communication allows UAVs to share critical information such as the location of hotspots, environmental changes, and strategic maneuvers, which is crucial for coordinated wildfire management operations. In a dynamic and rapidly evolving wildfire scenario, the ability of UAVs to communicate in real-time ensures that the team can adapt quickly to changing conditions, distribute tasks efficiently, and optimize their collective efforts for fire detection, assessment, and suppression. The success of such a basic learnable interagent communication scheme in this work highlighted both the potential for improved target discovery and collision avoidance behaviors as well as the training instability inherent with increasing task complexity. Further work may investigate this tradeoff by training various teams with different communication capacities, though such robust analysis might be best tested once simulator computational efficiency has been improved (see Appendix A.3).

Task 3 represents the most realistic 3D Cooperative Navigation task variant studied in this thesis, including both partial observability and interagent communication. Coupled with the high-fidelity AirSim simulation environment, the successful end-to-end training of a UAV team in Task 3 represents a promising avenue for future fully-integrated autonomous wildfire response solutions.

6.2 Implications for Wildfire Management

In wildfire management, there is a notable gap between simulated works and real-world wildfire management operations. This thesis worked towards bridging that gap by employing high-fidelity simulations in AirSim. In exchange for a significant reduction in computational efficiency, the developed systems are robust and reliable, increasing the likelihood of successful real-world deployment. The simulation platform developed is modular, with capacity for customized wildfire environments and camera integration for further increased realism. This adaptability allows re-

searchers and practitioners to tailor the simulation to specific scenarios, enhancing its utility and applicability to various wildfire management challenges.

Furthermore, this work addressed a largely unnoticed gap in related works regarding dimensionality. By developing methods to successfully train teams of autonomous agents in fully three-dimensional Cooperative Navigation, MARL methods become more applicable to wildfire management tasks. Fully utilizing verticality allows UAV agents to enhance their capabilities for detection, assessment, and even suppression. This added dimension significantly improves the agents' operational effectiveness, enabling them to navigate complex terrain, gain better vantage points for monitoring, and deploy suppression mechanisms more efficiently.

The implications of these advancements are profound for wildfire management. Autonomous UAVs trained through these methods can potentially transform how wildfires are detected and managed, providing rapid, real-time data and intervention capabilities. This can lead to faster response times, more accurate assessments of wildfire spread and intensity, and more effective deployment of suppression resources, ultimately reducing the impact and damage caused by wildfires.

6.3 Limitations and Future Research

6.3.1 Vectorized Environments

By far the most significant limitation of the developed simulation environment is a lack of computational efficiency. AirSim's linked rendering and physics engine prevented all efforts at vectoring environments such that multiple simulated teams could train at the same time. Further, the provided sim-time controls were found to be highly damaging to simulation fidelity. Thus, the simulations conducted in this work are locked to a 1:1 ratio of sim-time to clock-time.

As reinforcement learning is known for requiring large amounts of data, commonly requiring millions or tens of millions of timesteps for well-trained agents, this puts the simulated environment at a severe disadvantage. Optimizations to AirSim's built-in simulation reset method as well as a variety of enhancements to early environment exploration were able to make up in part for the platform's computational inefficiency. Appendix A.2 goes further into specific improvements. The employed curriculum learning methods allowed teams to learn complex tasks with access to hundreds of thousands of timesteps, where traditional learning methods would require millions or fail entirely. For the three-step curriculum learning method used for Task 2, a total of 90 episodes were employed,

each with 4096 timesteps, totaling 368,640 simulated timesteps - a total real-world training time of approximately 60 hours.

A clear direction for improvement of the platform and work would be to further improve computational efficiency. Significant efforts have been made towards vectorizing the simulator (see Appendix A.3), though were ultimately met in failure. The ability to simultaneously train multiple teams within the simulation would result in dramatic training speedups, without any compromises to environment fidelity. Such speedups would enable more robust analyses and ablation studies of the developed methods in realistic time frames.

6.3.2 Imagery-Based Detection

One of the core simplifications of the 3D Cooperative Navigation environments was the use of radius-based observations (at least for tasks with partial observability). Effectively, if objects were within some set observation distance of the observing agent, the properties of those objects are assumed to be fully visible to that agent. Clearly, this assumption is not realistic. Real world autonomous UAVs will likely depend on a variety of sensor fusions to detection their surroundings, predominantly using distance sensors, LiDAR cameras, visual spectrum cameras, and infrared cameras. One of the core reasons why AirSim was selected is for it's superb sensor and camera integrations. With minor modification, the platform can be reconfigured to instead use onboard gimbal cameras as input rather than the radius-based observation method.

Given the author's past works collecting and processing real-world wildfire imagery with UAVs [23, 15, 11], the environment was designed with the plan to eventually integrate collected imagery from the FLAME 2 or FLAME 3 datasets into simulations, effectively training simulated teams to detect and monitor real-world wildfires. Unfortunately, this scheme was never realized as it was found that camera-based inputs were computationally prohibitive (both in terms of training difficulty and computation overhead) for the given deadline. Future works would continue these efforts, bringing the simulated methods closer to real-world wildfire management.

6.3.3 Simulation

One of the main limitations is the reliance on simulated environments, which, despite being high-fidelity, may not capture all the complexities and unpredictabilities of real-world scenarios and

flight dynamics. Future research should focus on bridging this gap by conducting field tests and real-world evaluations of the developed systems. Integrating Hardware-in-the-Loop (HITL) simulations within AirSim could be an effective intermediate step. HITL allows for the incorporation of real-world hardware into the training processes, which can help in identifying and addressing discrepancies between simulated and real-world performance. By using actual UAVs flight controllers, software stacks, and other relevant hardware components in conjunction with the simulation environment, researchers can gain insights into practical challenges such as sensor noise, hardware limitations, and real-time processing constraints. This approach can simplify the transition from simulation to real-world deployment by providing a more accurate and practical testing ground for the autonomous systems.

6.4 Final Thoughts

In conclusion, this research represents a significant step forward in the development of autonomous systems for wildfire management. By addressing key challenges in three-dimensional Cooperative Navigation and leveraging the power of multi-agent reinforcement learning and curriculum learning, this study provides valuable insights and tools for future advancements. The successful training of autonomous agents in complex, three-dimensional tasks not only enhances our understanding of MARL but also opens new possibilities for improving wildfire response strategies.

As climate change continues to increase the frequency and intensity of wildfires, the need for innovative and effective solutions has never been more urgent. This research contributes to this pressing need, offering a promising pathway towards more efficient and effective wildfire management practices. By bridging the gap between theoretical simulations and real-world applications, and by providing a robust and adaptable simulation platform, this thesis lays the groundwork for future research and development in the field. The integration of autonomous systems into wildfire management holds the potential to significantly improve response times, operational efficiency, and overall effectiveness in combating one of the most challenging natural disasters facing our world today.

Appendices

Appendix A Lessons Learned and Software Workarounds

A.1 Further Comments on Simulator Choice

The AirSim simulator [58] provides several attractive features for training UAV teams on realistic wildfire response tasks, many of which were unutilized or underutilized in this work. Several of the more interesting features are listed:

- **Realism** – AirSim is built on Unreal Engine, a state of the art simulation and rendering engine known for its ability to model real-world environments with high fidelity. A wealth of marketplace assets allows level designers to drop in 3D scans of real world forestry and other ultra-realistic assets directly into an Unreal Engine stage. When considering the optimal simulation and rendering engine for creating convincing forest fire recreations, Unreal Engine is a very strong candidate. This does however come at the cost of computational efficiency, resulting in some non-ideal properties for multi-agent reinforcement learning, as is discussed in further sections of this Appendix.

- **Camera Integrations** – AirSim boasts flexible image collection and manipulation APIs, which allow the simulator to be used to collect imagery data and to train reinforcement learning agents on real-time camera feeds. This works especially well when coupled with Unreal Engine's capacity for hyper-realism. Further, external real-world imagery can be integrated into training workflows to allow RL agents to train on real world wildfire data, including multi-spectral imagery like that in FLAME 1 [61] or FLAME 2 [23]. Such an adequately configured simulator would further bridge the gap between real-world wildfire tasks and theoretical research works.

- **Real-Time Hardware-In-The-Loop (HITL)** – A key gap of many theoretical UAV works is the modeling of conventional UAV flight controllers, which themselves are a source of error and variance. AirSim allows for HITL with popular hardware platforms such as Pixhawk, common firmwares such as PX4, and standard messaging protocols like MavLink. This is not to say that AirSim is unique in providing real-time HITL. The Gazebo simulator [30] has several packages including Hector [35] and RotorS[21] that can be modified to provide similar functionality.

- **Wind and Weather APIs** – AirSim includes programmatic control over some basic weather/environmental aspects including simulated time of day, wind, rain, snow, dust, and fog. Such variable condi-

tions would allow the training of more generalizable, robust RL agents that are able to operate in uncertain environments. Further, if the provided weather controls are unsuitable for an application, additional weather effect plugins can be installed from the Unreal Marketplace and dropped into Unreal stages to provide more flexibility.

A.2 Resetting AirSim

One of the basic requirements for any reinforcement learning environment is the ability to reset and reinitialize the environment state and any agent states. Such a routine is expected to be called thousands of times over the course of a typical training regime, and is thus desired to be computationally effective and minimally invasive. As is discussed further in the following Appendix A.3, simulation resetting is also desired to be localized to a specific agent, team, or environment within a simulation as to enable parallel training. AirSim's provided API call for resetting a simulation (both environment and agents) is workable, though generally inefficient at this task, especially in a multi-agent setting.

AirSim's reset API is applied to the whole simulation including all agents within. Effectively, it restarts a new instance of the simulation, resetting each agent position to preset fixed points as well as resetting each agent's simulated flight controllers. Thus, for a task that requires any form of random initialization, there must be some added customized code to have each agent arm, takeoff, and then navigate to the desired starting locations. This is quite costly, adding a significant temporal overhead to collect what may be very brief trajectories.

To enable the trainings presented in this thesis, a more efficient reset functionality was created. AirSim provides the ability to teleport agents to arbitrary coordinates in simulation, though notably they do not provide the ability to reset flight controllers. Without a means to reset agent flight controllers, the typical behavior when teleporting a UAV agent is that the agent will crash or ignore further commands due to flight controller failures. To avoid these limitations, the created reset command first commands agents into a slow vertical ascent (2 meters/second) for approximately a second before the teleport is enacted to ensure that agents have a relatively consistent flight controller state before teleportation. Immediately after teleportation, the agents are commanded to hover, allowing a second or two for internal flight controller states to settle. This process has been found to allow arbitrary individualized agent reinitialization without invoking AirSim's provided reset command, with generally lower overhead. It is noted that AirSim's reset command is called

every 1000 team resets to hard reset flight controller states. It is suspected that repeated soft resets may accumulate error over time, potentially resulting in inconsistent agent behaviors.

A.3 AirSim in Parallel

The largest limitation of employing AirSim for reinforcement learning tasks is certainly the lack of parallelization. Certain workaround allow the running of multiple simulation processes at a time on a single computer, though such efforts are typically met with computation slowdown rather than improvement, as the multiple processes compete for resources. Throughout the creation of this thesis, significant efforts were dedicated towards creating a means of training multiple teams of UAVs simultaneously within the same simulation. These efforts were mostly unsuccessful, as there was no ready solution for resetting individual teams without impacting the training of other teams. The reason for this can be found by investigating the procedure used to step an environment within the simulation:

1. Pause the simulation (if not already paused)
2. Send each agent their individual actions
3. Unpause the simulation and continue for 50 frames
4. Pause the simulation
5. Calculate observations, rewards, and collisions for each agent.

While designing some centralized synchronous stepper to simultaneously step multiple environments within a single simulation is feasible, the author was unable to devise a reasonable means of resetting a single environment (see the procedure outlined in the above Appendix A.2) that would not interfere with the stepping of other environments. It is a requirement of many stacked environment wrappers (torchrl, PettingZoo, SuperSuit, etc.) that stacked environments are able to be reset individually. Thus, the effort to allow for multiple teams to train within AirSim simultaneously was eventually abandoned.

A.4 Simulation Time vs Clock Time

AirSim has a linked physics and render engine through Unreal, which can cause difficulty when attempting to improve computation efficiency. The physics engine runs at 120 Hz, with a

default 1:1 ratio of simulated time to clock time. When the application is not focused or if the operating machine is under significant computational load, the physics engine refresh rate will drop, resulting in slower simulation time than real-time. AirSim provides some limited controls to increase the simulation speed above real-time. That is, rather than 120 simulation update ticks being equal to one real simulated second, 90 or 60 simulation updates might be made per simulated second, resulting in a 1.5 or 2.0 ratio between simulated time and clock time. While such speed increases allow for faster training, they come at the cost of fidelity. Faster simulations are less accurate, and it was found that the difference between simulation timings was sufficient for teams trained at 2:1 sim-time to clock-time were unable to demonstrate equivalent performance on 1:1 sim-time to clock-time. As a result, all simulations in this thesis were conduced using the default 1:1 sim-time to clock-time ratio. The remainder of this section is used to provide some basic temporal measurements and calculations of the created 3D Cooperative Navigation AirSim environments.

For the training configurations used in this thesis, it was typical for an episode of 4096 training steps to take approximately 41 minutes on an unloaded system[1]. Given a physics engine target of 120 Hz and 50 frames per simulation step, a 4096 training step episode should ideally take 1706.67 seconds or 28.44 minutes. Thus, the created environment has approximately a 30% temporal overhead for environment resetting, model weight updates, and intermittent computation slowdown from competing processes. For the three stage curriculum learning process outlined for Task 2, 90 episodes each at 40 minutes results in approximately 60 hours total training time.

A.5 From PettingZoo to TorchRL

The most popular python package for single-agent reinforcement learning (SARL) is OpenAI's Gymnasium [14], which establishes a framework for defining and interacting with single-agent RL environments. For multi-agent reinforcement learning (MARL), the Farama Foundation provides PettingZoo [66], a python package designed as a multi-agent extension of Gymnasium. PettingZoo introduces two MARL environment concepts: "Parallel Environments" and "Agent Environment Cycle (AEC) Environments". Parallel environments apply to tasks where every agent acts and observes simultaneously. AEC environments describe turn based games where agents act and observe sequentially. PettingZoo also provides several wrappers to provide compatibility with popular SARL Python libraries, such as Stable-baselines3 [52] and Supersuit[67]. Stable-baselines3 provides

[1]CPU: i7-10700K, GPU: RTX 3070ti, RAM: 32 GB, OS: Windows 10

ready-access benchmarks, algorithms, and evaluations for SARL tasks, and was initially intended to be used with this project. Similarly, Supersuit provides wrappers for environment parallelization/stacking as well as some imagery manipulations for training RL agents with camera inputs and was intended for use in this thesis.

Initial efforts at training agents towards 3D Cooperative Navigation in AirSim employed a custom made PettingZoo Parallel API environment. While the environment's base functionalities all worked as desired, AirSim requires the use of a MultirotorClient object to send and receive information from the simulator. This object depends on future objects and python's asynchronous communication library to manage the connection between python code and the Unreal Engine simulation. Notably, this MultirotorClient object is incompatible with the commonly used python object packing library Pickle and/or Cloudpickle. It was discovered that many of PettingZoo's and Supersuit's environment wrappers have hidden dependencies on either the Pickle or Cloudpickle libraries, preventing the usage of Supersuit and Stable-baselines3 with the created PettingZoo environment. Despite various attempted workarounds such as RLLib [36] and other online "solutions", resolving these incompatibilities proved unworkable.

After several weeks of debugging, TorchRL [13] was discovered as a near drop-in solution. TorchRL is an open-source reinforcement learning python library developed by Meta in 2022 that is built directly on pytorch [49], one of the most popular machine learning python frameworks. TorchRL employs a "dataset pillar" approach that provides excellent modularity and top-notch compatibility with other RL libraries. Further, this approach considers both multi-agent tasks as well as vectorized environments to both be simply another dimension to the data tensors. For a programmer well-versed in tensor operations, the framework can be used for powerful operations with relatively minimal learning overhead. Their provided wrapper for PettingZoo environments allowed for the previously created 3D Cooperative Navigation environment to be used with TorchRL modules with minimal required modification. Within a day or two of setup, the environment was configured to train a team of agents using TorchRL and pytorch.

Bibliography

[1] AmpliCam — Wildfire Detection Systems. https://www.amplicam.com/.

[2] Geostationary Operational Environmental Satellites (GOES)R Series. https://www.goes-r.gov/multimedia/dataAndImageryVideosGoes-17.html.

[3] Prescribed fire — US Forest Service. https://www.fs.usda.gov/managing-land/prescribed-fire.

[4] Wildfire Detection Systems - Forest Fire Smoke Automatic Detection - SMOkED. https://smokedsystem.com/.

[5] Wildfires and acres — National Interagency Fire Center. https://www.nifc.gov/fire-information/statistics/wildfires.

[6] ALERT Wildfire. https://www.alertwildfire.org/, 2018.

[7] F. Afghah, M. Zaeri-Amirani, A. Razi, J. Chakareski, and E. Bentley. A coalition formation approach to coordinated task allocation in heterogeneous (UAV) networks. In *IEEE American Control Conference (ACC'18)*, June 2018.

[8] Fatemeh Afghah. Autonomous unmanned aerial vehicle systems in wildfire detection and management-challenges and opportunities. In Erik Blasch, Frederica Darema, and Alex Aved, editors, *Dynamic Data Driven Applications Systems*, pages 386–394, Cham, 2024. Springer Nature Switzerland.

[9] Fatemeh Afghah, Abolfazl Razi, Jacob Chakareski, and Jonathan Ashdown. Wildfire monitoring in remote areas using autonomous unmanned aerial vehicles. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 835–840, 2019.

[10] V G Ambrosia, S Wegener, D V Zajkowski, S Buechel, F Enemoto, E Hinkley, B Lobitz, and S Schoenung. The Ikhana UAS western states fire imaging missions: from concept to reality (2006–2010). *Geocarto Int. J.*, 26:85–101, 2011.

[11] Julia Boone, Bryce Hopkins, and Fatemeh Afghah. Attention-guided synthetic data augmentation for drone-based wildfire detection. In *IEEE INFOCOM 2023-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6. IEEE, 2023.

[12] Sayed Pedram Haeri Boroujeni, Abolfazl Razi, Sahand Khoshdel, Fatemeh Afghah, Janice L. Coen, Leo O'Neill, Peter Fule, Adam Watts, Nick-Marios T. Kokolakis, and Kyriakos G. Vamvoudakis. A comprehensive survey of research towards ai-enabled unmanned aerial systems in pre-, active-, and post-wildfire management. *Information Fusion*, 108:102369, 2024.

[13] Albert Bou, Matteo Bettini, Sebastian Dittert, Vikash Kumar, Shagun Sodhani, Xiaomeng Yang, Gianni De Fabritiis, and Vincent Moens. Torchrl: A data-driven decision-making library for pytorch. *arXiv preprint arXiv:2306.00577*, 2023.

[14] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[15] Xiwen Chen, Bryce Hopkins, Hao Wang, Leo O'Neill, Fatemeh Afghah, Abolfazl Razi, Peter Fulé, Janice Coen, Eric Rowell, and Adam Watts. Wildland fire detection and monitoring using a drone-collected rgb/ir image dataset. *IEEE Access*, 10:121301–121317, 2022.

[16] United States Joint Economic Committee. Climate-exacerbated wildfires cost the U.S. between $394 to 893$ billion each year in economic costs and damages, 10 2023.

[17] DataCluster Labs. Fire and smoke dataset, 2021.

[18] Christian Schroeder De Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip HS Torr, Mingfei Sun, and Shimon Whiteson. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*, 2020.

[19] Dincer, Baris. Wildfire detection image data, 2021.

[20] Jeffrey L Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1):71–99, 1993.

[21] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. *Robot Operating System (ROS): The Complete Reference (Volume 1)*, chapter RotorS—A Modular Gazebo MAV Simulator Framework, pages 595–625. Springer International Publishing, Cham, 2016.

[22] Alex Graves, Marc G Bellemare, Jacob Menick, Remi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. In *international conference on machine learning*, pages 1311–1320. Pmlr, 2017.

[23] Bryce Hopkins, Leo O'Neill, Fatemeh Afghah, Abolfazl Razi, Eric Rowell, Adam Watts, Peter Fule, and Janice Coen. Flame 2: Fire detection and modeling: Aerial multi-spectral image dataset, 2022.

[24] Qiyuan Huang, Abolfazl Razi, Fatemeh Afghah, and Peter Fule. Wildfire spread modeling with aerial image processing. In *2020 IEEE 21st International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, pages 335–340, 2020.

[25] Shafkat Islam, Qiyuan Huang, Fatemeh Afghah, Peter Fule, and Abolfazl Razi. Fire frontline monitoring by enabling uav-based virtual reality with adaptive imaging rate. In *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, pages 368–372, 2019.

[26] Minqi Sebastian Jiang. *Learning Curricula in Open-Ended Worlds*. PhD thesis, UCL (University College London), 2023.

[27] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 267–274, 2002.

[28] Ali Khan and Bilal Hassan. Dataset for forest fire detection, August 2020.

[29] Sahand Khoshdel, Qi Luo, and Fatemeh Afghah. Pyrotrack: Belief-based deep reinforcement learning path planning for aerial wildfire monitoring in partially observable environments, 2024.

[30] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, 2004.

[31] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999.

[32] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[33] Christos Kyrkou and Theocharis Theocharides. Emergencynet: Efficient aerial image classification for drone-based emergency monitoring using atrous convolutional feature fusion. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 13:1687–1699, 2020.

[34] Kyrkou, Christos and Theocharides, Theocharis. Deep-learning-based aerial image classification for emergency response applications using unmanned aerial vehicles, 2019.

[35] Junheng Li and Quan Nguyen. Force-and-moment-based model predictive control for achieving highly dynamic locomotion on bipedal robots. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 1024–1030. IEEE, 2021.

[36] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. RLlib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2018.

[37] Suhyeon Lim, Heejung Yu, and Howon Lee. Optimal tethered-uav deployment in a2g communication networks: Multi-agent q-learning approach. *IEEE Internet of Things Journal*, 9(19):18539–18549, 2022.

[38] Xiao Liu, Yuanwei Liu, and Yue Chen. Reinforcement learning in multiple-uav networks: Deployment and movement design. *IEEE Transactions on Vehicular Technology*, 68(8):8036–8049, 2019.

[39] Xiao Liu, Yuanwei Liu, Yue Chen, and Lajos Hanzo. Trajectory design and power control for multi-uav assisted wireless networks: A machine learning approach. *IEEE Transactions on Vehicular Technology*, 68(8):7957–7969, 2019.

[40] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

[41] Xueguang Lyu, Yuchen Xiao, Brett Daley, and Christopher Amato. Contrasting centralized and decentralized critics in multi-agent reinforcement learning. *arXiv preprint arXiv:2102.04402*, 2021.

[42] Luis Merino, Fernando Caballero, Jose Ramiro Martinez de Dios, Iván Maza, and Aníbal Ollero. Automatic forest fire monitoring and measurement using unmanned aerial vehicles. 2010.

[43] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[44] Earth Science Data Systems Nasa. MODIS — EarthData. `https://www.earthdata.nasa.gov/sensors/modis`.

[45] Earth Science Data Systems Nasa. VIIRS — EarthData. `https://www.earthdata.nasa.gov/sensors/viirs`.

[46] Moses Olafenwa. FireNET, August 2019.

[47] Ritu Pande. Fire Detection from CCTV, 2019.

[48] Jacopo Panerati, Hehui Zheng, SiQi Zhou, James Xu, Amanda Prorok, and Angela P. Schoellig. Learning to fly—a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7512–7519, 2021.

[49] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[50] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[51] Han Qie, Dianxi Shi, Tianlong Shen, Xinhai Xu, Yuan Li, and Liujing Wang. Joint optimization of multi-uav target assignment and path planning based on multi-agent reinforcement learning. *IEEE Access*, 7:146264–146272, 2019.

[52] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[53] RAIN. Autonomous aircraft for wildfire response. /urlhttps://www.rain.aero/.

[54] Pol Rosello and Mykel J Kochenderfer. Multi-agent reinforcement learning for multi-object tracking. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1397–1404, 2018.

[55] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[56] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[57] Dhruvil Shah. Fire detection, July 2020.

[58] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles, 2017.

[59] Alireza Shamsoshoara and Fatemeh Afghah. *Airborne Fire Detection and Modeling Using Unmanned Aerial Vehicles Imagery: Datasets and Approaches*, pages 525–550. Springer International Publishing, Cham, 2023.

[60] Alireza Shamsoshoara, Fatemeh Afghah, Abolfazl Razi, Liming Zheng, Peter Z Fulé, and Erik Blasch. Aerial imagery pile burn detection using deep learning: The flame dataset. *Computer Networks*, 193:108001, 2021.

[61] Alireza Shamsoshoara, Fatemeh Afghah, Abolfazl Razi, Liming Zheng, Peter Fulé, and Erik Blasch. The flame dataset: Aerial imagery pile burn detection using drones (uavs), 2020.

[62] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. Pmlr, 2014.

[63] Yunlong Song, Selim Naji, Elia Kaufmann, Antonio Loquercio, and Davide Scaramuzza. Flightmare: A flexible quadrotor simulator. In *Conference on Robot Learning*, pages 1147–1157. PMLR, 2021.

[64] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[65] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[66] J. K. Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis Santos, Rodrigo Perez, Caroline Horsch, Clemens Dieffendahl, Niall L. Williams, Yashas Lokesh, and Praveen Ravi. Pettingzoo: Gym for multi-agent reinforcement learning, 2021.

[67] J. K. Terry, Benjamin Black, and Ananth Hari. Supersuit: Simple microwrappers for reinforcement learning environments, 2020.

[68] UAVOS. UAVOS deploys UVH 170 unmanned helicopter for wildfire support. /urlhttps://www.uavos.com/uavos-deploys-uvh-170-unmanned-helicopter-for-wildfire-support/, 4 2024.

[69] Alberto Viseras, Michael Meissner, and Juan Marchal. Wildfire front monitoring with multiple uavs using deep q-learning. *IEEE Access*, 2021.

[70] Tian Wang, Ruoxi Qin, Yang Chen, Hichem Snoussi, and Chang Choi. A reinforcement learning approach for uav target searching and tracking. *Multimedia Tools and Applications*, 78:4347–4364, 2019.

[71] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[72] Adam C. Watts, Vincent G. Ambrosia, and Everett A. Hinkley. Unmanned aircraft systems in remote sensing and scientific research: Classification and considerations of use. *Remote Sensing*, 4(6):1671–1692, 2012.

[73] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

[74] H. Wu, , H. Li, A. Shamsoshoara, A. Razi, and F. Afghah. Transfer learning for wildfire identification in uav imagery. In *54th Annual Conference on Information Sciences and Systems (CISS)*, 2020.

[75] Zhaoyue Xia, Jun Du, Jingjing Wang, Chunxiao Jiang, Yong Ren, Gang Li, and Zhu Han. Multi-agent reinforcement learning aided intelligent uav swarm for target tracking. *IEEE Transactions on Vehicular Technology*, 71(1):931–945, 2022.

[76] XPrize. XPRIZE Wildfire Competition. /urlhttps://www.xprize.org/prizes/wildfire, 4 2024.

[77] Hongyi Zhang, Zhiqiang Qi, Jingya Li, Anders Aronsson, Jan Bosch, and Helena Holmström Olsson. 5g network on wings: A deep reinforcement learning approach to the uav-based integrated access and backhaul. *arXiv preprint arXiv:2202.02006*, 2022.

[78] Kaiqing Zhang, Zhuoran Yang, Han Liu, Tong Zhang, and Tamer Başar. Fully decentralized multi-agent reinforcement learning with networked agents, 2018.

[79] Wenqi Zhang, Qiang Wang, Xiao Liu, Yuanwei Liu, and Yue Chen. Three-dimension trajectory design for multi-uav wireless network with deep reinforcement learning. *IEEE Transactions on Vehicular Technology*, 70(1):600–612, 2021.

[80] Wenhong ZHOU, Jie LI, Zhihong LIU, and Lincheng SHEN. Improving multi-target cooperative tracking guidance for uav swarms using multi-agent reinforcement learning. *Chinese Journal of Aeronautics*, 35(7):100–112, 2022.