Adversarial Swarm Defense with Decentralized Swarms

by

Jason X. Zhou

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Kristofer Pister, Chair
Professor Murat Arcak

Spring 2021

Abstract

Adversarial Swarm Defense with Decentralized Swarms

by

Jason X. Zhou

Master of Science in Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Kristofer Pister, Chair

The rapid proliferation of unmanned aerial vehicles (UAVs) in both commercial and consumer applications in recent years raises serious concerns of public security, as the versatility of UAVs allow the platform to be easily adapted for malicious activities by adversarial actors. While interdiction methods exist, they are either indiscriminate or are unable to disable a large swarm of drones. Recent work in wireless communications, microelectromechanical systems, fabrication, and multi-agent reinforcement learning make UAV-based counter-UAV systems increasingly feasible - that is, defense systems consisting of autonomous drone swarms interdicting intruder drone swarms. Such a system is desirable in that it can conceivably produce a highly versatile, adaptable, and targeted response while still retaining an aerial presence.

We progress towards such a system through deep reinforcement learning in two distinct domains, which can be broadly described as 1-vs-1 and N-vs-N autonomous adversarial drone defense. In the former scenario, we learn reasonable, emergent drone dogfighting policies using soft actor-critic through 1-vs-1 competitive self-play in a novel, high-fidelity drone combat and interdiction simulation environment, and demonstrate continual, successful learning throughout multiple generations. In the latter case, we formalize permutation and quantity invariances in learned state representation methods for downstream swarm control policies. By using neural network architectures that respect these invariances in the process of embedding messages received from, and observations made of members of homogeneous swarms (be it friendly or adversarial), we enable parameter-sharing proximal policy optimization to learn effective decentralized perimeter defense policies against adversarial drone swarms, while models using conventional state representation techniques fail to converge to any effective policies. Two such possible embedding architectures are presented: adversarial mean embeddings and adversarial attentional embeddings.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Unmanned aerial vehicles (UAVs), or more broadly drones, have seen a rapid and dramatic increase in usage throughout a wide variety of applications in recent years. The transformation of the UAV from a tightly-held, predominantly military-based technology to a consumer and commercial one, as well as its proliferation subsequently brings with it serious public safety concerns. In particular, quadcopters are especially appealing to malicious actors as they can be customized and outfitted with a variety of equipment to carry out insidious tasks all without risking any personnel in the process [1]; this is, in essence, the modern democratization of aerial warfare. Furthermore, non-malicious neutral actors such as drone enthusiasts are also liable to causing accidental, yet unacceptable incursions on sensitive airspace such as that over airports, governmental facilities, or critical public infrastructure. Furthermore, with recent advances in microelectromechanical systems (for example, in the form of miniature autonomous rockets and pico air vehicles [2], [3]), fabrication techniques, and decentralized control algorithms, truly large-scale, organized groups of autonomous microrobotic drones are becoming increasingly feasible. It becomes clear that the aforementioned risks and concerns are only amplified as the number of drones involved in an adversarial attack grow from one to potentially thousands, much like the scenario outlined by Russell et al. in [4]; we term this the *pervasive intelligent swarms* problem.

Counter-UAV systems (cUAV), which are designed to detect, intercept, or eliminate UAV threats, have thus seen growing demand as well. cUAV systems generally consist of two broad stages: detection or tracking, followed by some form of interdiction. The first stage is relatively mature, with a variety of modalities in use such as radar, radio-frequency (RF), electro-optical, infrared, acoustic or a combination of those aforementioned; the second stage, interdiction, is much less so [1]. Current methods include communication-based interdiction, such as jamming or spoofing RF or GPS signals, and physical interdiction. State-of-the-art drone removal technologies in this second category include the Battelle Drone Defender, NetGun X1, Skywall 100, Airspace Interceptor, or even highly trained eagles (adopted by the Dutch police and French Army) - all of which generally require highly skilled operators, and thus are easily overwhelmed when faced with a large swarm of incoming drones [5]. More heavy-handed and indiscriminate methods such as electromagnetic pulse devices (EMPs) may leave defenders without an aerial presence [1].

When considering the desired qualities of an effective cUAV system that is able to counter an incoming adversarial drone swarm - versatility, adaptability, resiliency, autonomy in threat analysis and target selection, etc. - one may come to the conclusion that those are precisely the qualities of the intelligent drone swarm itself, and the characteristics that make its misuse so dangerous. Conceivably, the intelligent drone swarm could serve as a viable candidate to countering its own malicious use - that is, defense systems consisting of high-agent-count autonomous drone swarms interdicting intruder drones. Such a system would be highly desirable in that it can conceivably produce a highly versatile and targeted response in a large variety of environments and situations, while still retaining an aerial presence. This serves as the motivation behind this thesis, in which we progress towards such a system through two settings, which can be broadly described as 1-on-1 and N-on-N adversarial drone defense.

In chapter 2, we focus on autonomously eliminating adversarial drones in the 1-on-1 setting; specifically, learning emergent drone dogfighting behaviours through 1-vs-1 competitive self-play and deep reinforcement learning. We present a novel high-fidelity drone combat and interdiction environment based on the *AirSim* simulator, an accompanying drone control and competitive self-play framework library, and successfully train autonomous agents using our framework, simulator, and soft-actor critic [6] in which by generation 5, drones are able to achieve $\approx 90\%$ winrate in a 1-on-1 engagement over an equal distribution of past generations while demonstrating reasonable emergent dogfighting behaviours after ≈ 50 hours of training. We additionally study the effect of generational hyperparameter scaling, and empirically show that it is necessary to achieve consistent and continual learning in a difficult task generation after generation.

In chapter 3, we examine N-on-N perimeter defense games, specifically focusing on enabling practical deep reinforcement learning in this area. We present a 2D multiagent environment simulating adversarial swarm defense of a sensitive region, present a formalization of deep embeddings, permutation and quantity invariance for swarm control policies, and outline potential architectures that preserve these key invariances that leverage useful and defining characteristics of homogeneous swarms, namely adversarial mean embeddings and adversarial attentional embeddings. We then benchmark these architectures on the environment through parameter-sharing, centralized-training decentralized-execution proximal policy optimization [7] against the conventional concatenation state representation method, and demonstrate superior performance as measured by mean episodic reward after 0.4 million training iterations, and successfully learn highly effective decentralized perimeter defense policies for this swarm-on-swarm confrontation scenario.

In the final chapter, we consider potential areas of future work in both settings that further build towards autonomous drone swarms as decentralized defensive systems against adversarial drone swarms.

Chapter 2

Countering Drones with Drones Via Competitive Self-Play

2.1 Overview

In this chapter, we study the mechanics of autonomous drone defense against an adversarial drone through 1-on-1 drone dogfighting as a key step towards fully autonomous swarm-on-swarm defense. We introduce a novel, high-fidelity drone combat and interdiction environment for deep reinforcement learning (RL), along with an accompanying competitive self-play framework and library. Using Soft Actor-Critic (SAC) [6], we then train up to five generations of drones in the task of drone interdiction, in which both drones aim to eliminate its opponent by aiming a ranged cone-of-effect over the other drone for a set number of continuous timesteps. With 2 million iterations per generation, we are able to produce autonomous agents that eliminate enemies sampled from an equal distribution of past generations with a 90% winrate. We additionally observe that the agents learn complex emergent aerial elimination and dogfighting tactics similar to those manually crafted. Lastly, during the training process, we employ an annealed dense reward curriculum and generational hyperparameter scaling in which the skill ceiling of the task is gradually increased, and show empirically that the former is essential for a difficult task with sparse rewards, and the latter necessary for continual, successful learning reflected in increasing winrates throughout multiple training generations.

2.2 Related Works

With regards to deep learning and c-UAV systems, much effort has been made in the sensing/tracking of intruder drones, including computer vision methods and multi-modal sensor fusion [8], [9], making this portion of c-UAV systems relatively robust in contrast to the interdiction stage. Even in the few research works on autonomous interdiction of drones via defensive drones such as [5], the details and process of drone takedowns are almost always abstracted away as an instant elimination upon the defender arriving in some range of the intruder, with the assumption that the intruder is a rather simple agent instead of also being a well-trained autonomous actor. In other words, the focus is almost always on the dynamic deployment of defender drones as opposed to high frequency control and aerial maneuvers involved in quadcopter dog-fighting between two skilled agents - this is a lacking area we hope to advance in this project. We additionally explore the specific dynamics and strategies involved in situations where *both* parties within a quadcopter confrontation are armed and capable of interdiction (the standard simplifying assumption is again that intruder drones are incapable of combat), to which we turn to deep reinforcement learning to gain insight on emergent behaviour.

With regards to deep reinforcement learning and self-play, Bansal et. al. [10] demonstrated emergent complexity via multi-agent competition. Although the environments created and used for this purpose were 3D, complex, and physically accurate, they are nevertheless learning examples such as sumo wrestling and kicking a goal. We follow many of the fundamental principles laid out in this paper for competitive self-play and learning, and examine whether they may potentially translate to real world scenarios by applying them to a high-fidelity drone simulator to learn policies that can easily and correspondingly be transferred directly to real drone controllers.

2.3 Simulation Environment & Drone Control

We employ *AirSim* [11], a high-fidelity Unreal Engine-based simulator for vehicles - including quadcopters with a footprint of 1m^2 - as our primary simulation environment for the drone dogfighting set-up. *AirSim* provides many built-in functionalities; however, the ways in which users interact with *AirSim* can be seen in the right half of Figure 2.1, which due to its complexity is not conducive for the typical machine learning workflow. *simdrones* is a Python library we have built on top of the *AirSim* package that aims to accelerate the *AirSim* research workflow, shown on the left of Figure 2.1 to wrap all of *AirSim* and its related components.

The *simdrones* library firstly provides an *AirSim* connector that interfaces with and manages *AirSim* instance lifecycles, including automatic generation of the *AirSim* settings file, as well as the choice of environment to deploy (i.e. a typical neighborhood, a flat plane with large block obstacles, etc.), removing the need for manual configuration. Additionally, *simdrones* provides a unified interface for drone sensing and control. This includes retrieving absolute position (only position relative to spawn is provided directly by *AirSim*), velocity and angle measurements, and snapshots from the drones' on-board cameras. In terms of movement control, *simdrones* provides low-level movement interface in angle-throttle movement (specifying roll, pitch, yaw, and throttle), or direct rotor pulse-width-modulation (PWM) control (specifying the PWM individually for each of the four rotors), as well as high-level movement controls such as directly setting rotation, altitude, and movement to a specified coordinate. Lastly, there are controls geared towards RL, such as random spawning

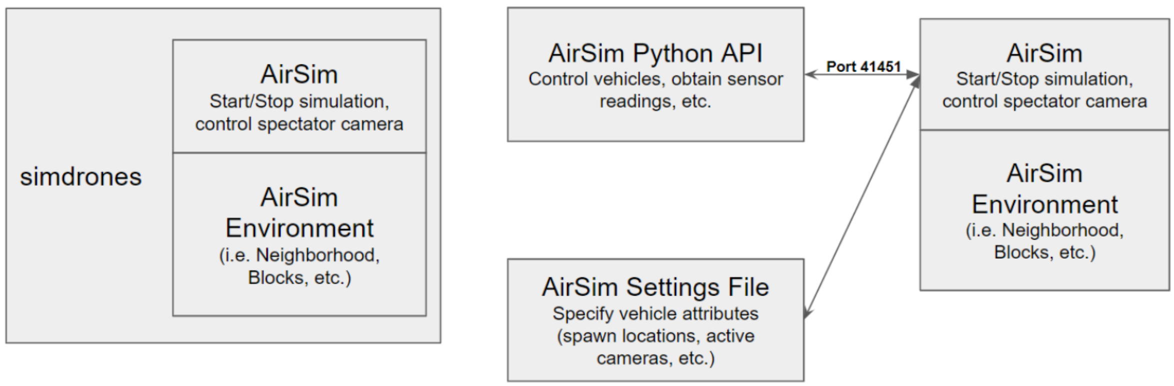

Figure 2.1: Architectural overview of *AirSim* with *simdrones* (left) and without (right).

of drones and resetting the environment to a predetermined configuration. Using *AirSim* and *simdrones*, users can rapidly and efficiently iterate and construct their own RL environments; we do so, and build a competitive self-play framework on which we can deploy learning algorithms for building autonomous UAV-based c-UAV systems.

2.4 Competitive Self-Play & Deep Reinforcement Learning Framework

For our 1-vs-1 competitive self-play training, we construct an OpenAI-Gym compliant [12] reinforcement learning environment upon which we can deploy any suitable learning algorithm of choice - in our particular case, Soft Actor-Critic (SAC) [6]. We walk through each component of the environment below.

Opponent Sampling

In every 1-vs-1 encounter, we designated one drone as the trainer, and the other the trainee. At training generation n, the trainer's model weights are those from generation g, where $g \sim \text{Uniform}(\delta n, n)$ for which $\delta \in [0, 1]$ is a hyperparameter determining how far back in history the trainer's weights will be sampled from, and the trainee is the agent (using weights from generation $n-1$, and when $n = 1$, the trainee is a random actor) learning from competing against the previous generations. The trainer is then *part of the environment.* At each generation, we produce k independent agents from k independent 1-vs-1 training sessions, all of which are available to be sampled as the trainer for a future generation. In our experiments, we chose $\delta = 0$ (always sample from all past generations), $k = 1$, and we

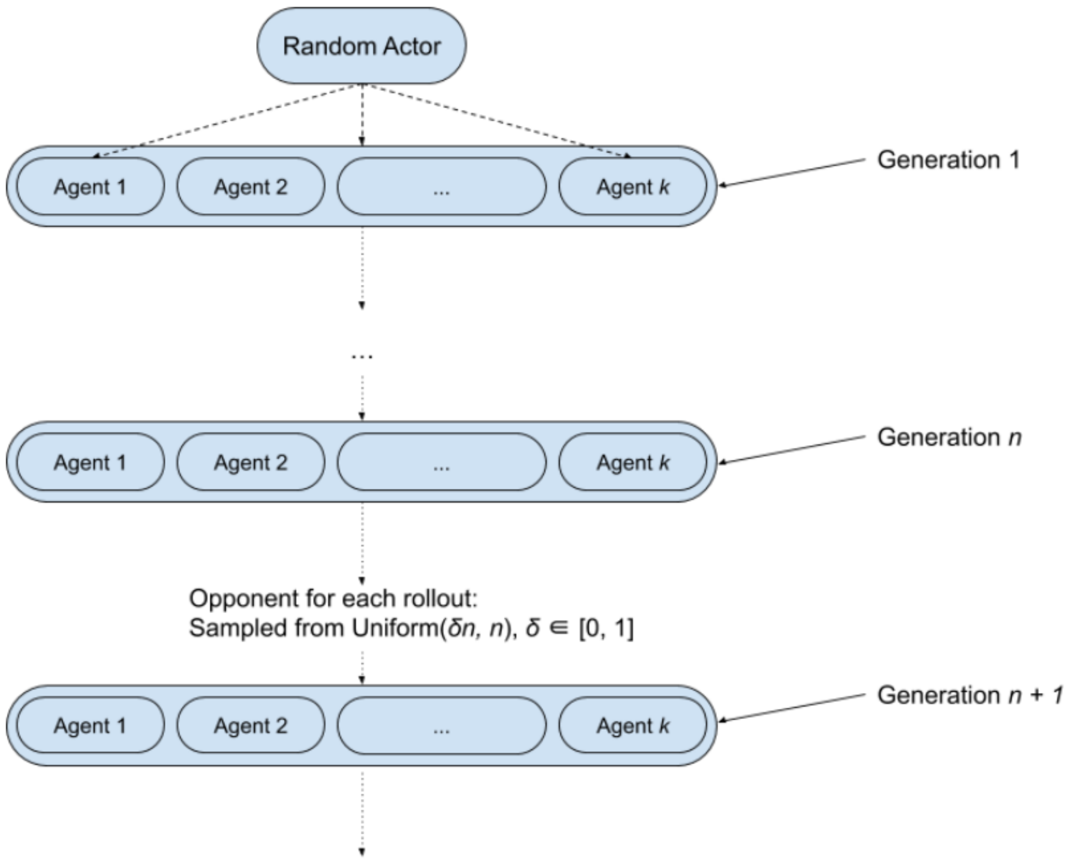

Figure 2.2: Competitive Self-Play Opponent Sampling Scheme.

proceeded up to $n = 5$ generations. For each training episode, we use 2 million iterations of SAC, with implementation provided by the Stable Baselines library [13].

Interdiction

Each of the two drones in an engagement are assigned some variable number of health-points (specific values for each generation can be found in Table 2.2); when a drone's health-points reach zero, that drone is interdicted. The interdiction process mimics existing net-shooters (and future missile-shooters) in that the attacking drone must keep the opponent within a cone-of-effect stemming from the front face of the drone. The cone-of-effect is approximately 15m long with a radius of 4m. For each clock tick that a drone is able to keep the opponent in its cone, the opponent will lose 1 health-point, but if the opponent escapes the cone, its healthpoints will be reset to full. That is, the cone-of-effect must be over the opponent for a contiguous number of timesteps. We set the clock frequency of the environment to be 50Hz.

Observations and Actions

As we are focusing on the interdiction stage as opposed to the sensing and tracking stage of c-UAV systems, we initially assume that each actor can make the following perfect observations of themselves and their opponent every 0.02 seconds, corresponding to the clock frequency:

1. Self and Opponent Position triplets (x, y, z)
2. Self and Opponent Velocity triplets (v_x, v_y, v_z)
3. Self and Opponent Yaw (t)
4. Self and Opponent Health-points (h)

The observation vector is then of size 16. Next, we employ a high-level controller for the drone movements with a continuous action space; the actions consist of a simultaneous bounded velocity delta (± 0.2 m/s) and bounded yaw delta (± 5 degrees). Headless movement mode is chosen as the cone-of-effect stems from the front face of the drones, and so additional flexibility in the separation of yaw from heading direction is essential during this form of combat.

World Randomization

We instantiate the two opposing drones at random locations, one each within two distinct 20m^3 regions suspended 40m in the air, 25m apart, with random orientations to prevent over-fitting of policies to any particular starting positions. The trainer and trainee drones also alternate between the two spawning regions at the start of every episode. The simulation environment itself has no obstacles other than the flat ground plane. While randomization in the environment ensures that the policies learned are better able to adapt and generalize, it does hinder learning in the early stages leading to poorer beginner performance due to the added difficulty. Thus, we additionally introduce a curriculum to the environment to aid in early-stage learning.

Curriculum Training & Rewards

An episode terminates when one of the following 7 scenarios in Table 2.1 occur, for which events (5) - (7) are considered to be *confrontations*.

As with many RL tasks, the positive rewards from the end conditions are sparsely encountered when the desired end conditions are difficult to achieve; that is especially true in the case of interdicting an opponent drone without any form of guidance or curriculum. Thus, we provide the agents with an annealed dense reward *only for the first generation*, as well as generation hyperparameter tuning and the termination rewards as a curriculum to facilitate learning throughout all future generations. The first-generation dense reward is roughly based on the distance between the two drones. Specifically, we define it to be:

Table 2.1: Episode Termination Scenarios

No.	Description	Confrontation Classification	Victor
1	Trainer crashes	No	Trainee
2	Trainee crashes	No	Trainer
3	Time out	No	Trainer
4	Termination Distance	No	Trainer
5	Trainee interdicts trainer	Yes	Trainee
6	Trainer interdicts trainee	Yes	Trainer
7	Simultaneous interdictions	Yes	Trainer

$$r = -c\alpha\gamma^t ||s_{trainer} - s_{trainee}||^2 \tag{2.1}$$

For which s denotes the drones' position vectors, $c = 0.001$ is a constant scaling factor, $\gamma = 0.99$ is the reward discount factor, t is the current timestep of the episode, and α is an annealing factor. The annealing factor is applied to the dense reward throughout the 2 million iterations of the first generation starting at timestep $t' = 1500000$ for a total length of $l = 200000$, which produces:

$$\alpha = \begin{cases} 0 & \text{if training generation} > 1 \\ 1 & \text{if } t < t' \\ \max(0, 1 - \frac{(t-t')^2}{l}) & \text{otherwise} \end{cases} \tag{2.2}$$

For the generation hyperparameter tuning, we begin with a relatively low number of health-points for the drones so that *confrontation* scenarios occur more often. Additionally, we set a maximum distance between the two drones at which the episode will terminate in a loss for the trainee, corresponding to scenario (4) so as to ensure that occurrences of events in which the two drones are vastly far from each other and are unlikely to confront are minimized. For exploration purposes, this distance is initially set to be large; this requirement gradually becomes more stringent (distance decreases) as the agents are trained across generations. Additional hyperparameters that may be tuned include the length and radius of the cone of effect, the spawn distances between the drones, etc. The specific values used for the following experiments are shown in Table 2.2.

Otherwise, we present the agent with a large positive award for win conditions as specified above, and likewise a large negative award for losses. Following standard practice, we also apply a small time penalty for each timestep the agent remains alive.

2.5 Experimental Results

With the aforementioned set-up, we run SAC upon our competitive self-play environment at two million timesteps per generation, for a choice of 5 total generations, on a Nvidia GTX

Table 2.2: Generational Hyperparameter Scaling

Generation	Termination Distance (m)	Health-Points
1	100	3
2	70	3
3	60	4
4	60	5
5	60	7

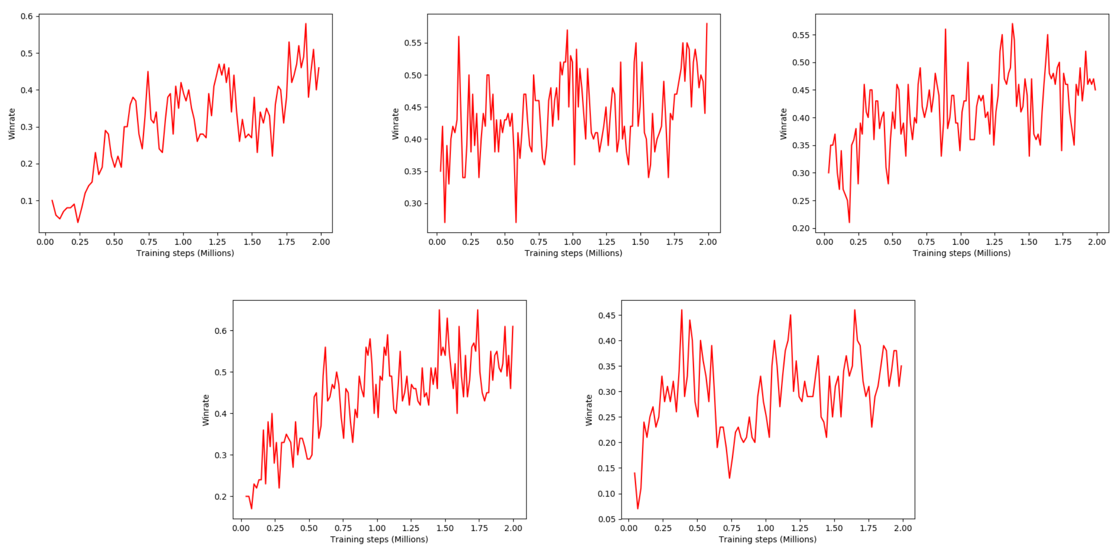

Figure 2.3: Overall winrates of generations 1-5 with uniform opponent sampling.

1080 GPU and Intel i9-9900k CPU for up to ≈ 50 - 60 hours of training time.

Winrates

Figure 2.3 and Figure 2.4 show respectively the graphs for overall winrates, and confrontation winrates of generations 1 through 5. We record the winrate every 100 episodes throughout the two million training steps per generation. We see a clear trend throughout each generation, in that the winrate steadily increases throughout the course of training and approaches 90%. Note that at the start of each new generation, the winrate is essentially reset as the trainer and trainee, if the immediate previous generation is sampled for the trainer, are identical and thus evenly matched.

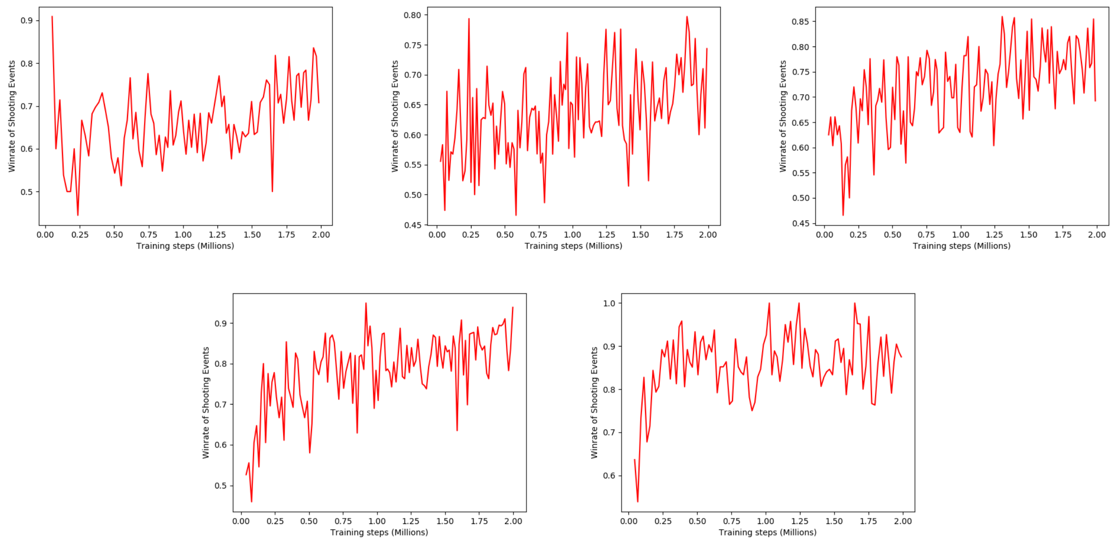

Figure 2.4: Confrontation winrates of generations 1-5 with uniform opponent sampling.

Generational Hyperparameter Scaling Effects

Note that without the generational hyperparameter scaling, no meaningful trend in the winrates are observed; that is, the agent in that case will be unable to learn the dogfighting task. We see in Figure 2.5 that with the dense curriculum provided to generation 1 (without which we observe no trends even for generation 1, while below we do observe such an upwards trend in winrate at least over generation 1), the winrate completely degenerates starting generation 2. This highlights the need for hyperparameter scaling, in order to gradually increase the skill ceiling of the task and increase its difficulty (in our case, by applying more stringent distance requirements and harder to eliminate enemies). Intuitively, if the task is too simple, it may be that by the end of generation 1, an optimal strategy is found and we observe only very basic, degenerate behaviour due to the simplicity of the task. Maintaining the enemy drone within the cone-of-effect for a few fractions of a second is almost never a guaranteed interdiction in real life with current net-shooter drones, and thus increasing the health-points of the drones enforces the requirement that drones must"lock-on" consistently for longer periods of time to achieve an elimination.

Emergent Behaviours

After five training generations, we performed 1-vs-1 dogfighting between two agents both sampled from generation 5. In these competitions, we observed emergent skills, such as drones attempting to circle to the back of the opposing drone - demonstrating a rather complex learned behaviour matching the cone-of-effect positioning and interdiction mechanics

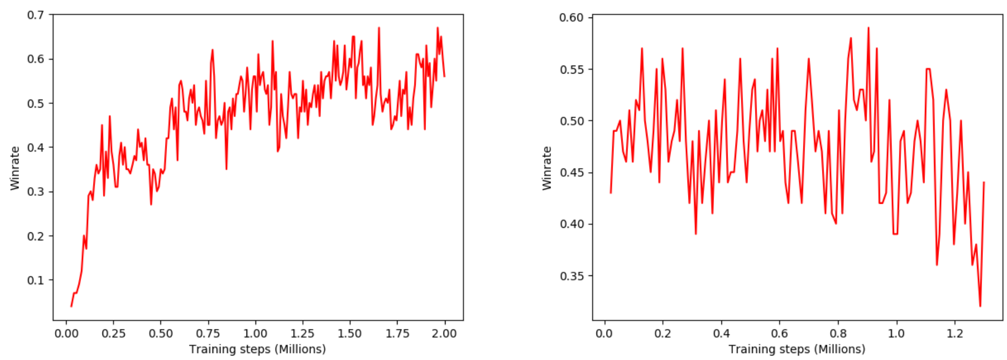

Figure 2.5: Overall winrates of generations 1-2 without generational hyperparameter scaling.

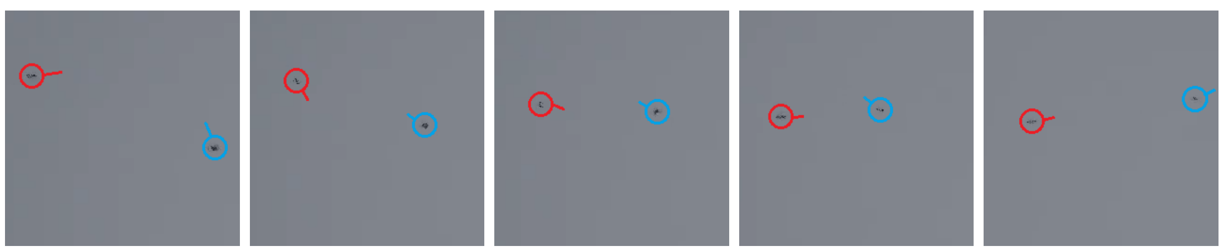

Figure 2.6: *AirSim* Screen captures of a 4 second sequence (displayed left to right chronologically), of emergent behaviors observed in a dogfighting sequence between two generation 5 agents. The drones are circled, and the corresponding lines indicate the heading of the drone.

(as the back of the enemy drone is the safest position, requiring the enemy to make the largest change in its yaw to attack), as well as drones performing evasive maneuvers starting at the 15 meters range from the opposing drone, which is precisely the cone-of-effect's range. When the two agents are sampled each from generation 5 and an earlier generation, we observe that the agent from generation 5, in confrontation scenarios, is almost always the one to maneuver more "decisively", such as *consistently and smoothly* moving or turning in a chosen direction, while drones from earlier generations may pivot between directions in a more stochastic manner. A specific dogfighting sequence is highlighted in Figures 2.6, which occurs between two generation 5 agents. We observe that the agent on the left swoops down aggressively into the agent on the right side of the figure; the opposing agent responds by backing away from the swooping agent correspondingly, maintaining a separation distance greater than the reach of the cone-of-effect.

Chapter 3

Deep Embeddings for Decentralized Swarm Systems

3.1 Overview

We now take a macroscopic approach and examine the control and dynamics of decentralized swarms in multi-agent perimeter-defense games, specifically in the context of enabling practical deep reinforcement learning in this area. As the number of agents within a swarm grows, so does the system's resiliency and capabilities; however, the number of interactions occurring also scales as agents explore the environment, cooperate with allies, and compete against adversaries.

In this chapter, we formalize the concepts of permutation and quantity invariant functions, and show their importance to learning for the swarm-on-swarm task of perimeter defense. Throughout experiments in this section, we take a centralized-training, decentralized-execution approach in which all agents share their experiences during the learning phase, but are evaluated in a decentralized manner in which all agents make their decisions locally, with no central coordinator - as is the case for an ideal, truly decentralized autonomous swarm. We discuss and present three representation techniques: conventional concatenation, adversarial mean embedding, and adversarial attentional embedding, the latter two of which attempt to faithfully capture the state of the swarm for downstream decision-making through the preservation of the aforementioned invariances. By respecting these invariances, these deep embeddings ensure that messages from or observations made of homogeneous agents are exchangeable, while being able to accept a variable number of such observations or communications as agents commonly enter or are eliminated from the environment. These techniques also address the issues of increasing dimensionality and scalability that particularly plague learning methods on autonomous swarm systems. In the case of 10-on-10 perimeter defense, we show that under the same algorithm, parameter sharing proximal policy optimization (PPO) [7] and the same number of timesteps (0.4 million), the policy utilizing conventional state representation is unable to learn successfully, while adversarial mean and adversarial attentional embeddings enable PPO to learn vastly more effective swarm defense strategies.

Real-world concerns such as communication constraints further change these dynamics and the observations of each agent makes of its surroundings; we include a case study in communication models and decentralized formation control at the end of this chapter.

3.2 Necessary Invariances for Swarm Control Policies

Conventional State Representation for Adversarial Swarm Defense

We first build towards and formalize the notion of permutation invariance for homogeneous swarm policies in the context of deep reinforcement learning and adversarial swarm-on-swarm perimeter defense. Assume that there are two distinct swarms consisting of n_d defenders and n_a attackers. For any particular one of the n_d agents within the defenders, it may receive a set M of up to $n_d - 1$ messages, denoted by m_i, from its teammates within the same swarm. Simultaneously it may make a set O of up to n_a observations of the attackers, denoted as o_i. Barring the defensive agent's own observations of itself, the total set of observations available to it is then:

$$S = O \cup M = \{o_1, o_2, ...o_{n_a}, m_1, m_2, ..., m_{n_d-1}\} \tag{3.1}$$

Conventionally, these elements of S are concatenated to produce a fixed-sized state vector. Denote this vector as S^c, which would generally be used downstream for deep RL. Note that if the size of each observation made of an attacker (the dimension of o_i) is K, and the size of each message received from a fellow defender is L (the dimension of m_i), then the size of the conventional state vector will be $Kn_a + L(n_d - 1)$. This may be acceptable for low agent count scenarios, but scalability becomes a serious concern when the number of agents grow large, as conventional concatenation based state representation does not mitigate or reduce the total input size of observations.

Note that as the swarms are homogeneous, the act of concatenation necessarily enforces an undesired ordering to the elements within M, and again to the elements within O; one key advantage of homogeneous swarms is that any agent is exchangeable with another, and consequently no agent is truly irreplaceable or unique. Therefore, concatenation essentially creates additional information about either the messages received by an agent of its homogeneous teammates, or observations made by an agent of its homogeneous enemies, that is not necessarily present or reflected in reality. Secondly, another primary advantage of swarm systems, in contrast to single-agent actors, is their resiliency to agent loss. In particular, this makes swarm systems particular well-suited to more advanced and challenging tasks oft-found in hazardous environments, such as search-and-rescue and exploration. Therefore, n_d and n_a are not necessarily constant throughout a trajectory; conventional architectures used for deep RL (generally fully-connected networks) are unable to handle varying input sizes, which would then be especially problematic for swarm multi-agent RL.

To address these three issues: scalability, undesired ordering, and variable input size, we can look to permutation and quantity invariant functions. We define quantity invariant functions as functions that can take in a variable number of fixed-sized inputs, and produce a fixed-sized output. Note that consequently, issues of scalability are necessarily resolved by quantity invariant functions due to invariant output size.

Permutation Invariant Embeddings

Next, we formalize permutation invariant functions; let f be permutation invariant, which for the purposes of deep RL we can readily assume to be a deep neural network parameterized by θ, and Y be some set of information obtained from a homogeneous swarm (for example, Y may be either O, or it may be M). Then, it must satisfy the following property:

$$f_\theta(Y_m) = f_\theta(Y_n) \;\; \forall Y_m, Y_n \in Y! \tag{3.2}$$

That is, the value of f_θ is the same under any permutation of such a set of elements Y. This resolves the undesired ordering issue as all possible orderings produce the same output.

Depending upon the deep RL algorithm employed, one may simultaneously require multiple such permutation invariant functions (i.e. value functions, policy functions, Q-networks, etc.). To adapt and use standard RL algorithms for swarm systems - instead of designing permutation invariant architectures for each network - one may compose a known permutation invariant function over each set of homogeneous observations to produce a single state embedding encapsulating the overall observations made/collected by one agent of the environment, that serves as the input to standard value/policy/Q-network architectures.

For example, in the case of proximal policy optimization (PPO) [7], a value function network V parameterized by ϕ, and policy network π parameterized by θ are required. Generally, we obtain the value estimate v of a state vector S^c, and the probability p of taking a particular action a in s as:

$$V_\phi(S^c) = v \tag{3.3}$$

$$\pi_\theta(a|S^c) = p \tag{3.4}$$

However, we know S^c to be the concatenation of the two sets O and M. With a known permutation invariant state embedding function ω_O for the elements of O and ω_M for the elements of M, we can instead obtain v, p through the following method:

$$S^O = \omega_O(O), \;\; S^M = \omega_M(M) \tag{3.5}$$

Let S^{OM} denote the concatenation of S^O and S^M (variations to concatenation are discussed in the following subsection). Next:

$$V_\phi(S^{OM}) = v \tag{3.6}$$

$$\pi_\theta(a|S^{OM}) = p \tag{3.7}$$

We see then that due to the permutation invariant nature of ω_O and ω_M, v, p obtained through this method are invariant to permutations with O and permutations within M.

(Note that we cannot permute across O and M as the information across both sets are not exchangeable with one another, while the elements within each set are, by nature of originating in the defending vs. attacking swarms).

Additionally, ω_O and ω_M may themselves also be deep networks parameterized by γ_O and γ_M, in which case γ_O, γ_M may be updated simply by through automatic differentiation, by backpropagating the gradients of ϕ and θ when those sets of parameters are optimized by the deep RL algorithm.

As we replace the state vector S^c with a learned embedding S^{OM} for use as the new state vector in deep RL algorithms, we call such transformations $S \to S^{OM}$ *deep embeddings*.

Architectures for Deep Embeddings

To construct a neural network that is permutation invariant, a common paradigm is to first transform individual inputs into another representation through a (learned) embedding function, and then to apply a commutative function (i.e. summation, max-pool, max, min, mean, etc.). By composing many such commutative functions, we can ultimately produce such quantity and permutation invariant ω functions. For example, ω_O and ω_M could be constructed by adapting [14] for the adversarial case in which two swarms are present, which achieve permutation and quantity invariance through the mean operator. We term this swarm adversarial mean embeddings:

$$S^O = \omega_O(O) = \frac{1}{n_a} \sum_{i=1}^{n_a} v_{\gamma_O} \cdot o_i \tag{3.8}$$

$$S^M = \omega_M(M) = \frac{1}{n_d - 1} \sum_{i=1}^{n_d - 1} v_{\gamma_M} \cdot m_i \tag{3.9}$$

In which $v_{\gamma_O}, v_{\gamma_M}$ are learned vectors parametrized by γ_O, γ_M.

Additionally, one can envision an attention-based mechanism with which to achieve the necessary invariances. For example, one may transform the messages m_i received from teammates into queries k_i through a learned transformation f_{query}:

$$q_i = f_{query}(m_i) \tag{3.10}$$

We may similarly produce a set of keys for observations of enemies o_j through a learned transformation f_{key}, and a set of value embeddings similarly:

$$k_j = f_{key}(o_j) \tag{3.11}$$

$$v_j = f_{value}(o_j) \tag{3.12}$$

We can then produce the attention weight coefficients as:

$$\alpha_{ij} = \frac{exp(q_i \cdot k_j)}{\sum_{j'} exp(q_i \cdot k_{j'})} \tag{3.13}$$

For each message m_i, we can then produce an attentional encoding e_i over the agent's enemy observation set O as:

$$e_i = \sum_j \alpha_{ij} v_j \tag{3.14}$$

We can then apply a permutation and quantity invariant transformation over the set of embeddings e_i. We term this architecture swarm adversarial attentional embeddings.

Lastly, in the previous subsection we produced S^{OM} as the concatenation of S^O and S^M; one may also generalize this subsequent transformation on S^O and S^M as some function H. We now have:

$$S^{OM} = H(S^O, S^M) \tag{3.15}$$

However, note that H need not have any special properties, since permutation and quantity invariances to O, M have already been achieved through S^O, S^M, and since the elements of O and M originate from distinct swarms we may now impose an ordering or other constraints on S^O and S^M. Therefore, H itself may be absorbed into conventional deep RL architectures directly and thus the focus need only be on producing state embeddings for homogeneous swarms; namely, the method of creating S^O and S^M.

3.3 Related Works

Robust, practical, and deployable multi-agent reinforcement learning (MARL) remains an active area of research and brings with it unique challenges in addition to those of single-agent RL; applying MARL to large scale swarm systems commonly amplify the effects of these challenges by nature of having a higher agent count at play. These challenges include the non-stationarity of the multi-agent environment, scalability as the number of actors grows large, state representation for multi-agent environments, communication models amongst agents, and training and executing policies under different coordination paradigms (centralized vs. decentralized). Generalized MARL algorithms have been developed by adapting pre-existing single-agent RL algorithms, namely Q-learning and policy gradient methods, and addressing their shortcomings; specifically, that Q-learning-based methods are affected by non-stationarity, and policy gradient-based methods by high variance as the number of actors grows large. The most prominent of these MARL algorithms include Counterfactual Multi-Agent Policy Gradients (COMA) [15], Multiagent Bidirectionally-Coordinated Nets (BICNET) [16], and Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments (MADDPG) [17]. However, in this chapter, we specifically focus on state embedding networks as a preprocessor to standard single agent RL methods, since the centralized training paradigm allows us to simply accumulate the observations of all agents during learning.

Many learning representation or embedding methods that are applicable for this concern come from the fields of computer vision (CV), or natural language processing (NLP) for example; many methods have also been proposed to specifically address this issue within the context of RL, in both state representation and communication models. One such method is Deep Sets [18] and Deep M-Embeddings [19], which use and explore mean, max, and min as the commutative functions. Practical implementations using invariance in MARL include close-proximity drone flight [20]. Additionally, PointNet [21] for classification of 3D point clouds share many of the key elements of Deep Sets regarding invariance, but also introduces a mechanism for local and global information aggregation (in which a concatenation of the two are paired for each local feature) that has potential to be transferred to RL. Recurrent-based methods, including the Set Transformer [22], an attention-based neural network component, and SWARM Mappings [23], a modified long short term memory (LSTM) cell, consist another broad class of methods. Mean field methods, including Mean Field MARL [24], in which the Q-value function for a particular agent is defactorized into one dependent only upon its own action and the mean field over the action of its neighbors (a phantom mean agent representing all other agents) have also been proposed. In this chapter, we focus on architectures most closely related to deep sets/deep mean embeddings, as the simplicity in the architecture lends well to large swarms consisting of low-compute agents.

3.4 Experimental Set-up

We employ an OpenAI-Gym compliant 2D particle environment to implement our perimeter defense experiments, in which a set of homogeneous defensive agents learn through parameter-sharing proximal policy optimization with deep embeddings, strategies with which to defend a sensitive region from an incoming homogeneous adversarial swarm. The adversarial swarm employs the simple strategy of heading at max speed directly towards the sensitive perimeter. In this scenario, we further assume that attacker drone interdictions occur as an instant elimination upon any defender coming within a certain radius of the attacker; the specific mechanisms of 1-on-1 interdiction are examined in detail instead in the previous chapter. We also assume that both the attacking and defending agents are identical in capability (except that attacking agents cannot eliminate defending agents), the most important of which are speed and agent count. The defending and attacking sides each have a corresponding spawn center, placed 20m apart from each other; depending on the number of agents in the configuration, agents of each side randomly spawn around their spawn center with the approximate density of agents adaptively held constant. A sample spawn configuration can be seen in Figure 3.1.

Defenders are individually assigned sparse rewards; the rewards scheme is simply that for each interdiction, the responsible defender obtains a reward of $+5$. At each timestep, all defenders incur a reward of -0.005. The defenders are victorious when all attackers have been interdicted (and will receive a reward of $+100$), while the defenders lose when any one attacker enters the sensitive region (and will incur a reward of -100). The action space for the defenders are a length 2 vector denoting the displacement along the x-axis and y-axis (with maximum speed constraint enforced).

We test the adversarial mean embedding, and adversarial attentional embedding methods outlined in the previous section regarding architectures, against the conventional concate-

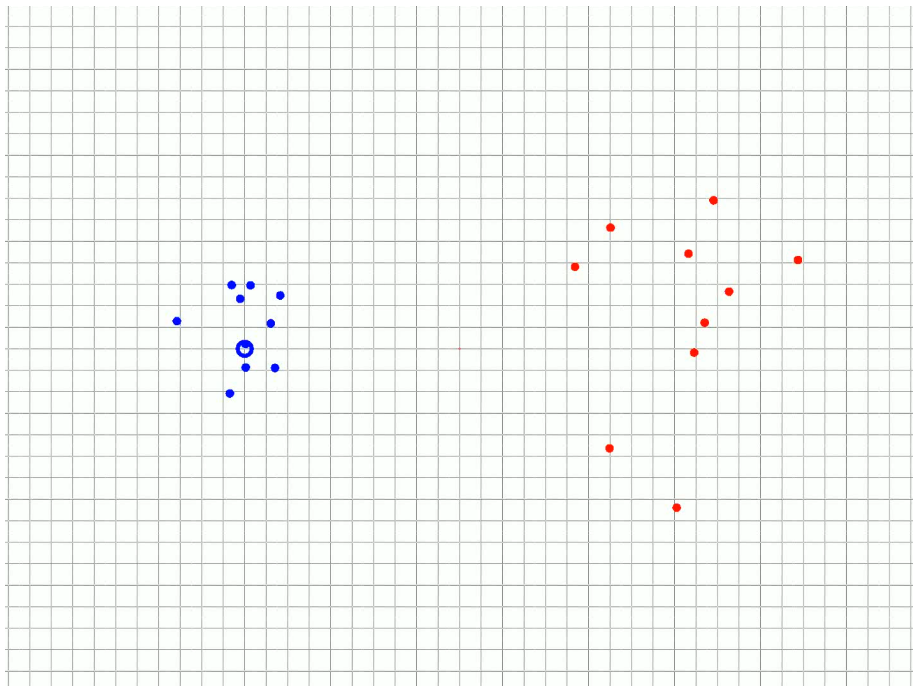

Figure 3.1: Sample spawn configuration with 10 defender agents in blue, 10 attacker agents in red, and the sensitive region rendered as the blue ring.

nation technique in confrontations between two swarms each consisting of 10 agents each. Each defensive agent receives the 2D locations of its teammates as m_i, the 2D locations of enemies and an additional Boolean flag indicating whether the enemy is dead or alive, as o_i; thus, in the vanilla setup, the state vector is of size $2 \cdot 9 + 3 \cdot 10 = 48$. Note that while a size 48 vector is not particularly large for a RL state space, in a 100-vs-100 scenario (a fairly reasonable size for a robotic swarm), the vector will grow to be of size 498.

3.5 Parameter-Sharing Proximal Policy Optimization with Deep Embeddings

For our experiments, we follow the centralized training, decentralized execution paradigm with PPO, using the PPO implementation and multi-agent environment frameworks from Ray [25] & RLlib [26], and custom deep embedding networks. All agents share parameters θ, ϕ, γ for the value, policy, and embedding networks respectively. During training, the experiences of each agent in the swarm are collected and the advantage estimates are made over their cumulative observations. The pseudo-code for this algorithm (modification upon the PPO pseudo-code outlined in [27]), used in the following experiments is outlined in

Algorithm 1. Note that the overall embedding architecture is denoted as ω, and the observation set S at time t denoted as s_t.

Algorithm 1 PS-PPO-DSE

```
Initialize environment with n homogenous swarm agents.
Initialize three sets of parameters: θ_0 for policy function, ϕ_0 for value function, γ_0 for embedding function.
for k = 0, 1, 2, ... do
    Initialize trajectories set D_k for current iteration.
    for l = 0, 1, 2, ... do
        Collect set of n trajectories {τ_1, τ_2, ...τ_n} by running policy π(θ_k) for all agents.
        Add trajectories set {τ_i} to D_k.
    end for
    Compute rewards-to-go R̂_t, and advantage estimates Â_t based on current value function V_{ϕ_k}.
    Update the policy and the embedding by maximizing the following standard PPO-Clip objective via gradient ascent:
```

$$\omega_{k+1}, \theta_{k+1} = \text{argmax}_{(\theta,\omega)} \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|(\omega(s_t))}{\pi_{\theta_k}(a_t|(\omega_k(s_t))} A^{\pi_{\theta_k}}((\omega_k(s_t), a_t), g(\epsilon, A^{\pi_{\theta_k}}((\omega_k(s_t), a_t))\right)$$

For which the function g is defined as:

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0 \end{cases}$$

```
    Fit value function and the embedding by regression on mean-squared error via gradient descent:
```

$$\omega_{k+1}, \phi_{k+1} = \text{argmin} \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} (V_\phi(\omega_k(s_t)) - \hat{R}_t)^2$$

```
end for
```

3.6 Experimental Results

Learning curves of the vanilla concatenation embedding, adversarial mean embedding, and adversarial attentional embedding showing the episode reward averaged over every 4000 iterations of the modified PPO algorithm in **Algorithm 1** are show in Figure 3.2; up to 0.4 million training iterations are employed. We see that concatenation fails to learn on the task

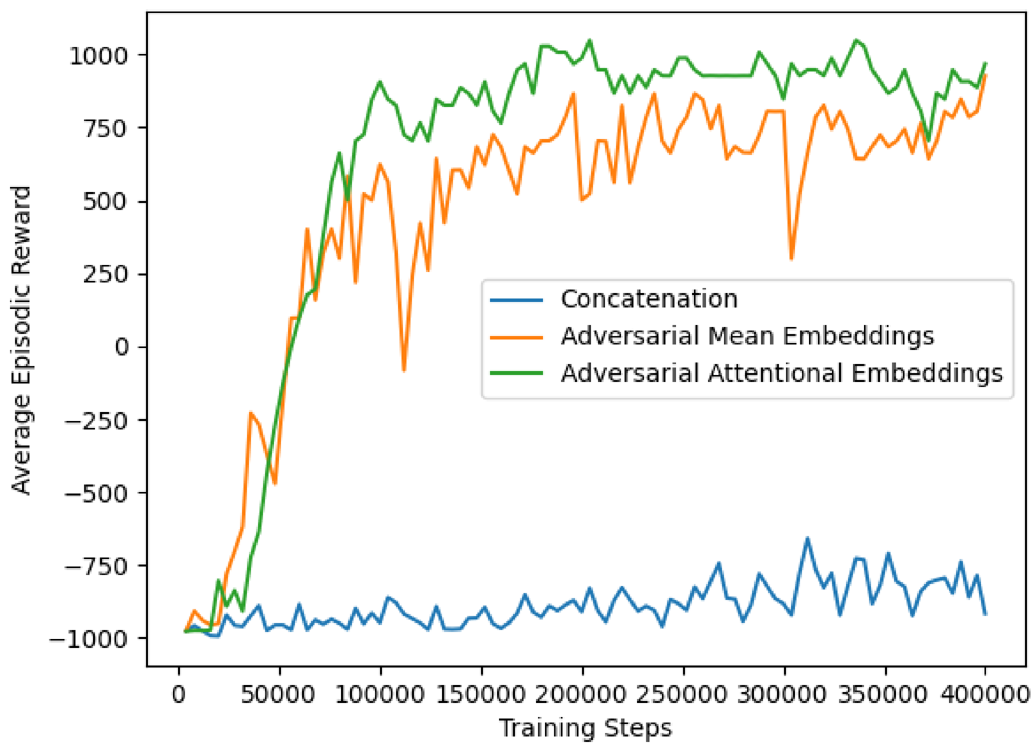

Figure 3.2: Average episodic reward over latest 4000 training iterations of concatenation, adversarial mean embeddings, and adversarial attentional embeddings with parameter-sharing proximal policy optimization in 10-on-10 perimeter defense games, up to 0.4 million training iterations total.

entirely; with the mean episodic rewards perturbating between -1000 and -750, this implies that the policy is failing to interdict any intruders. Meanwhile, by approximately training step 60000, both adversarial mean embedding and adversarial attentional embedding are able to achieve positive rewards, which indicates partially successfully interdictions. By the end of training, we see that adversarial mean embedding consistently achieves around +750 reward, while adversarial attentional embedding is the better performer at around +950 reward. A reward of that magnitude indicates that the defensive swarm is able to eliminate all attackers the vast majority of the time; this is verified in model evaluations as well. Such a stark contrast in the benchmark and performance between the conventional technique of concatenation, and swarm embedding methods in the task of swarm-on-swarm defense highlights the importance of respecting permutation and quantity invariance in perhaps other tasks involving (multiple) homogeneous swarm systems.

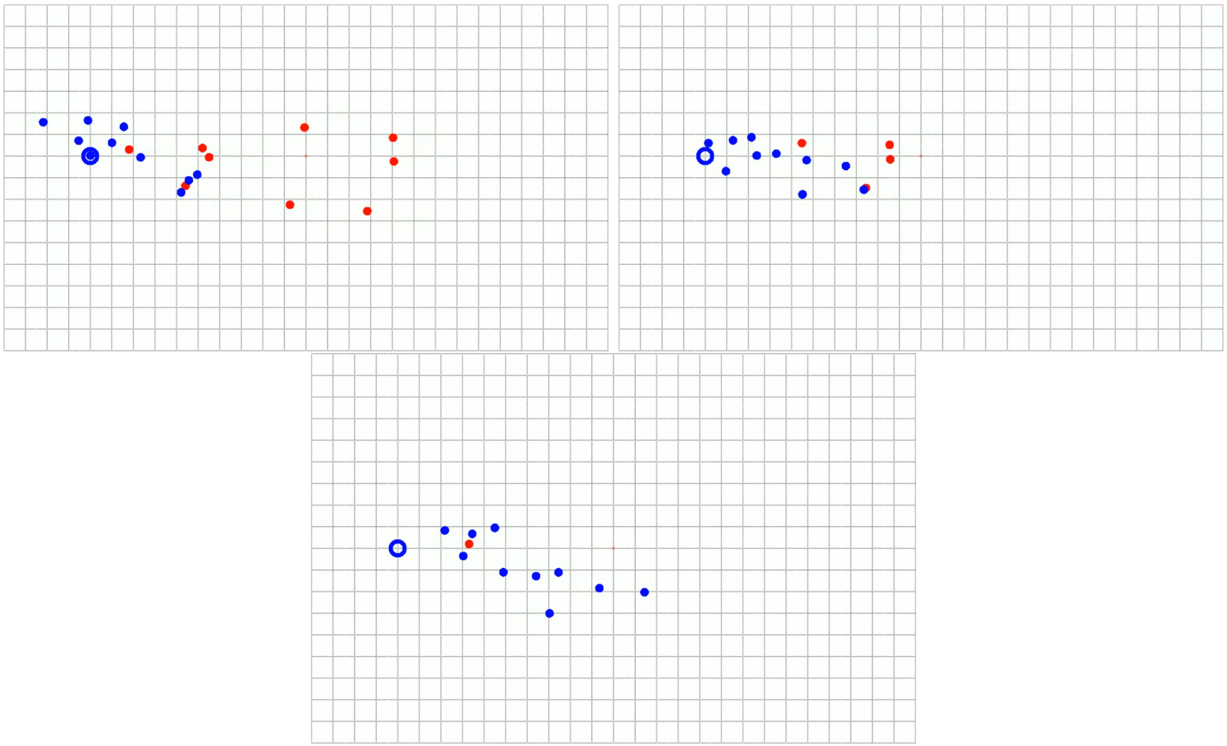

Figure 3.3: Screen captures of a sample progression of an adversarial swarm defense sequence between 10 attackers and 10 defenders, in which the defenders successfully eliminate all attackers, using swarm adversarial attentional embeddings.

In Figure 3.3, we examine a sample sequence of a successful swarm defense in the 10-on-10 scenario, in which the defenders are using adversarial attentional embeddings. The defenders at first assume a fairly spread out formation around the perimeter region, similar to the spawn configurations, and hover in that formation until the first intruders arrive. Then, the defending swarm proceeds to disperse along the horizontal axis, and in the process of dispersing, interdict nearby drones. This corresponds well to intuition, in that as the attackers approach from the right side of the environment, dispersing horizontally allows the swarm to have redundancies in its defense; if a defender misses an interdiction, its teammates to the left serve as backup. In contrast, dispersing vertically means that if a defender were to fail to perform an interdiction, then there is little chance for teammates to interdict the intruder (as both the defenders and attackers move at the same speed, it becomes impossible to interdict an intruder once it has passed the defender closest to the sensitive region).

3.7 Case Study: Realistic Communications & Swarm Formation Control

The aforementioned 2D particle environment is built with-in a novel simulator, also from Pister Group, called BotNet, which emulates realistic communication models and enables researchers to study the effects of realistic communication on high agent count swarms. We conduct an additional case study to examine the effects of different communication models on decentralized swarm control through formation control tasks [28]. The goal is to deploy agents that seek to organize themselves into a single line, irrespective of any particular ordering of agents or orientation of the line itself. The decentralized line formation algorithm follows from [29]; at each timestep, every agent locally applies least-squares to its own position and that of all neighbors it is connected to (the connectivity is then dictated by BotNet through the particular communication model configuration and position of the agents). This produces a local approximation of the optimal global line of convergence for each agent, which it uses to update its next control input (move directly towards their individual lines-of-best-fit with a maximum velocity). An example of the task is shown in 3.4.

We benchmark the effects of communication modeling on this decentralized swarm task through two metrics: convergence time and residual error (Figure 3.5) over 100 trials of each configuration of communication model and agent count. Convergence time is defined as the number of timesteps taken until all agents stop moving (i.e., until all agents individually believe the line formation task has been achieved). Note that this captures only the local belief state of the agents, and not the true global quality or convergence of the agent formation into a line. Residual error, calculated directly from applying a standard line-fitting least square procedure over the positions of all agents, shows the relative accuracy of a given formation. Omitting details on the 6 different communication models employed, it becomes clear from the benchmark results that communication has a large impact on the efficacy of decentralized swarm systems (i.e. better communication results in less residual error and faster convergence time), and thus the incorporation of communication into simulation and swarm modeling is an essential consideration, especially in high-agent-count systems.

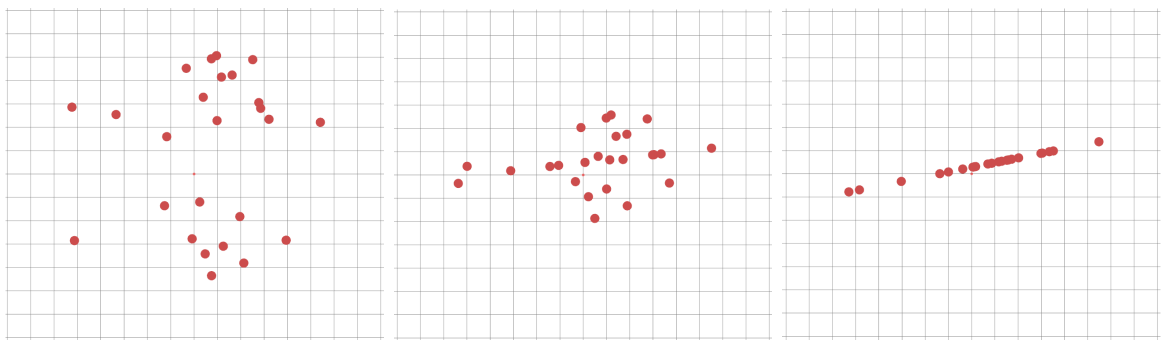

Figure 3.4: Screen captures of swarm during line formation task under full communication at timesteps 0, 25, 50 respectively.

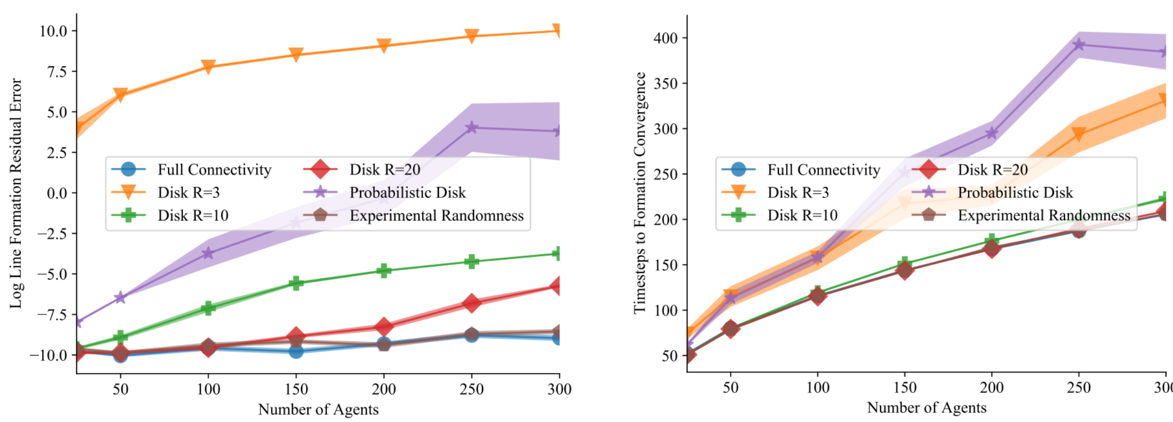

Figure 3.5: Logarithmic least squares residual error (left) and timesteps to line formation convergence (right) of line formation task under varying number of agents and communication models.

Chapter 4

Conclusions & Future Work

In this thesis, we have partitioned the problem of how best to counter pervasive intelligent drone swarms using intelligent drone swarms into two areas: 1-vs-1 autonomous interdiction, and N-vs-N decentralized swarm control; we also took a first step at examining realistic communication models in this context. Through 1-vs-1 competitive self-play, we observed emergent, complex maneuvering behaviours demonstrating reasoning about drone capabilities, and continually improving drone dogfighting skills generation after generation. In future work in this domain, it would be interesting to compare the performance of the learned strategies against human drone pilots or manually designed, classical control methods (such as linear quadratic regulators); an additional venture would be to transfer the learned policies to two physical drones (which *AirSim* is able to do directly), and demonstrate the dogfighting drone trajectories with hardware. In N-vs-N perimeter defense, we showed that homogeneous swarm reinforcement learning problems can benefit greatly from carefully crafted state embeddings that leverage the advantageous properties of swarms by respecting permutation and quantity invariances. In a problem setting where conventional parameter-sharing proximal policy optimization fails to find an effective policy entirely, we demonstrated that passing the state observations through adversarial mean embeddings or adversarial attentional embeddings architecture allows the algorithm to produce effective, decentralized swarm defense policies. An important next step in the future would be to push the agent count up to perhaps thousands of agents, and then examine the relative performance of various embedding techniques (and their performance in contrast to concatenation). Much like the case with 1-vs-1, demonstrating and evaluating these policies on hardware would also prove to be a valuable future contribution by bridging the gap between mere particle agents in simulation, and physical drones in reality.

Bibliography

[1] Jian Wang, Yongxin Liu, and Houbing Song. "Counter-Unmanned Aircraft System (s)(C-UAS): State of the Art, Challenges and Future Trends". In: *arXiv preprint arXiv:2008.12461* (2020).

[2] Ahad M Rauf et al. "Towards Aerodynamic Control of Miniature Rockets with MEMS Control Surfaces". In: *2020 IEEE 33rd International Conference on Micro Electro Mechanical Systems (MEMS)*. IEEE. 2020, pp. 523–526.

[3] Daniel S Drew, Brian Kilberg, and Kristofer SJ Pister. "Future mesh-networked pico air vehicles". In: *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2017, pp. 1075–1082.

[4] Stuart Russell et al. "Why You Should Fear "Slaughterbots"—A Response". In: *IEEE Spectrum* (2018).

[5] Philippe Martin Wyder et al. "Autonomous drone hunter operating by deep learning and all-onboard computations in GPS-denied environments". In: *PloS one* 14.11 (2019), e0225092.

[6] Tuomas Haarnoja et al. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *CoRR* abs/1801.01290 (2018). arXiv: 1801.01290. URL: http://arxiv.org/abs/1801.01290.

[7] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[8] Stamatios Samaras et al. "Deep learning on multi sensor data for counter UAV applications—a systematic review". In: *Sensors* 19.22 (2019), p. 4837.

[9] Eren Unlu et al. "Deep learning-based strategies for the detection and tracking of drones using several cameras". In: *IPSJ Transactions on Computer Vision and Applications* 11.1 (2019), p. 7.

[10] Trapit Bansal et al. "Emergent Complexity via Multi-Agent Competition". In: *CoRR* abs/1710.03748 (2017). arXiv: 1710.03748. URL: http://arxiv.org/abs/1710.03748.

[11] Shital Shah et al. "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles". In: *CoRR* abs/1705.05065 (2017). arXiv: 1705.05065. URL: http://arxiv.org/abs/1705.05065.

[12] Greg Brockman et al. "OpenAI Gym". In: *CoRR* abs/1606.01540 (2016). arXiv: `1606.01540`. URL: `http://arxiv.org/abs/1606.01540`.

[13] Ashley Hill et al. *Stable Baselines*. `https://github.com/hill-a/stable-baselines`. 2018.

[14] Maximilian Hüttenrauch, Adrian Sosic, and Gerhard Neumann. "Deep Reinforcement Learning for Swarm Systems". In: *CoRR* abs/1807.06613 (2018). arXiv: `1807.06613`. URL: `http://arxiv.org/abs/1807.06613`.

[15] Jakob Foerster et al. "Counterfactual multi-agent policy gradients". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[16] Peng Peng et al. "Multiagent bidirectionally-coordinated nets: Emergence of human-level coordination in learning to play starcraft combat games". In: *arXiv preprint arXiv:1703.10069* (2017).

[17] Ryan Lowe et al. "Multi-agent actor-critic for mixed cooperative-competitive environments". In: *arXiv preprint arXiv:1706.02275* (2017).

[18] Manzil Zaheer et al. "Deep sets". In: *arXiv preprint arXiv:1703.06114* (2017).

[19] Gregor HW Gebhardt, Maximilian Hüttenrauch, and Gerhard Neumann. "Using M-Embeddings to Learn Control Strategies for Robot Swarms". In: *Swarm Intelligence* (2019).

[20] Guanya Shi et al. "Neural-swarm: Decentralized close-proximity multirotor control using learned interactions". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 3241–3247.

[21] Charles R Qi et al. "Pointnet: Deep learning on point sets for 3d classification and segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 652–660.

[22] Juho Lee et al. "Set transformer: A framework for attention-based permutation-invariant neural networks". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 3744–3753.

[23] Roland Vollgraf. "Learning Set-equivariant Functions with SWARM Mappings". In: *arXiv preprint arXiv:1906.09400* (2019).

[24] Yaodong Yang et al. "Mean field multi-agent reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 5571–5580.

[25] Philipp Moritz et al. "Ray: A Distributed Framework for Emerging AI Applications". In: *CoRR* abs/1712.05889 (2017). arXiv: `1712.05889`. URL: `http://arxiv.org/abs/1712.05889`.

[26] Eric Liang et al. "Ray RLLib: A Composable and Scalable Reinforcement Learning Library". In: *CoRR* abs/1712.09381 (2017). arXiv: `1712.09381`. URL: `http://arxiv.org/abs/1712.09381`.

[27] Joshua Achiam. “Spinning Up in Deep Reinforcement Learning”. In: *GitHub repository* (2018).

[28] Kwang-Kyo Oh, Myoung-Chul Park, and Hyo-Sung Ahn. “A survey of multi-agent formation control”. In: *Automatica* 53 (2015), pp. 424–440.

[29] Xiaoping Yun, Gokhan Alptekin, and Okay Albayrak. “Line and circle formation of distributed physical mobile robots”. In: *Journal of Robotic Systems* 14.2 (1997), pp. 63–76.