INSTITUTO TECNOLÓGICO DE AERONÁUTICA

Nicholas Scharan Cysne

DEEP REINFORCEMENT LEARNING FOR MULTI-AGENT SYSTEMS IN MULTICRITERIA MISSIONS

Final Paper
2022

Course of Electronics Engineering

CDU 681.3:16

Nicholas Scharan Cysne

DEEP REINFORCEMENT LEARNING FOR MULTI-AGENT SYSTEMS IN MULTICRITERIA MISSIONS

Advisor

Prof. Dr. Carlos Henrique Costa Ribeiro (ITA)

Co-advisor

Prof. Dr. Cinara Guellner Ghedini (ITA)

ELECTRONICS ENGINEERING

São José dos Campos
Instituto Tecnológico de Aeronáutica

2022

Cataloging-in Publication Data
Documentation and Information Division

Scharan Cysne, Nicholas
Deep Reinforcement Learning for Multi-Agent Systems in Multicriteria Missions / Nicholas Scharan Cysne.
São José dos Campos, 2022.
74f.

Final paper (Undergraduation study) – Course of Electronics Engineering– Instituto Tecnológico de Aeronáutica, 2022. Advisor: Prof. Dr. Carlos Henrique Costa Ribeiro. Co-advisor: Prof. Dr. Cinara Guellner Ghedini.

1. Multi-Agent Systems. 2. Decentralized Networks. 3. Deep Reinforcement Learning. 4. Multi-Agent Reinforcement Learning. 5. Multicriteria Missions. I. Instituto Tecnológico de Aeronáutica. II. Title.

BIBLIOGRAPHIC REFERENCE

SCHARAN CYSNE, Nicholas. **Deep Reinforcement Learning for Multi-Agent Systems in Multicriteria Missions**. 2022. 74f. Final paper (Undergraduation study) – Instituto Tecnológico de Aeronáutica, São José dos Campos.

CESSION OF RIGHTS

Nicholas Scharan Cysne
Rua H8B, Ap. 219
12.228-461 – São José dos Campos–SP

DEEP REINFORCEMENT LEARNING FOR MULTI-AGENT SYSTEMS IN MULTICRITERIA MISSIONS

This publication was accepted like Final Work of Undergraduation Study

Nicholas Scharan Cysne
Author

Carlos Henrique Costa Ribeiro (ITA)
Advisor

Cinara Guellner Ghedini (ITA)
Co-advisor

Prof. Dr. Marcelo da Silva Pinho
Course Coordinator of Electronics Engineering

São José dos Campos: November 17, 2022.

To all people curious enough to put the
pursuit of knowledge in the first place.

Acknowledgments

First of all, to my family, which supported me all the time prior to and during my time at ITA, without you I would never have reached this far. Throughout hard decisions and sacrifices, my achievement is now your achievement as well.

To my professors, which inspired me and helped me to achieve an unmatched professional level, all lessons given were of great importance and will be well used. To Prof. Carlos Ribeiro, and Dr. Cinara Ghedini, for providing me with the knowledge and guidance to the making of this work. A special thanks to Prof. Marcos Máximo, which acted as a mentor and guided me during my time at ITAndroids, and presented me with the high standards that I'll take to my professional and personal life.

To all my friends, in special Thiago Filipe de Medeiros and Thayná Pires Baldão, from which I learned so much and am so grateful. To Reynaldo Lima and Vinícius de Pádua, that supported and endorsed my evil genie. Also, Lara Campos Ibiapina, Matheus da Silva Martins, and Fernando de Moraes Rodrigues, which bore this path with me and made this period funnier and lighter even on the worst days.

To all the countless colleagues and friends that I've gained from this place that names couldn't fit this space.

"Cada criatura é um rascunho a ser retocado sem cessar."

— Guimarães Rosa

Resumo

Veículos Aéreos Não Tripulados (VANTs) podem atuar em diversos cenários em que o envio de equipes humanas é de grande risco ou dificuldade, como missões de exploração e reconhecimento em áreas perigosas ou de condições extremas ao ser humano, tanto em locais de desastres, quanto em operações militares. Essas missões comumente empregam múltiplos VANTs formando uma rede que permite comunicação e cooperação entre eles. Para o sucesso desse tipo de formação, é necessário a existência de uma topologia de rede robusta, ou seja, capaz de manter certas características como conectividade entre todos os agentes da rede e resiliência a falhas. Este trabalho trata de Missões de Multi-Critérios, com diferentes atividades a serem realizadas durante a missão, compondo um cenário complexo em que a cooperação de robôs é essencial para o sucesso da mesma. Tal cenário é um bom candidato a se aplicar técnicas de aprendizado de máquina devido a grande quantidade de configurações possíveis e falta de um modelo que possibilite uma única solução ótima. Este trabalho propõe aplicar técnicas de Aprendizado por Reforço Profundo para treinar um agente, atuando como um VANT, que coopera com replicas de si mesmo a fim de criar topologias de rede capazes de operar em Missões de Multi-Critérios. O agente foi treinado segundo 4 tarefas distintas para simular esse tipo de missão, sendo: Movimentação para Local Alvo, Manutenção de Conectividade, Cobertura de Área e Robustez contra Falhas. Nós aplicamos técnicas de *Deep Reinforcement Learning* e *Multi-Agent Reinforcement Learning*, como *Proximal Policy Optimiztion* (PPO) e *Centralized Training with Decentralized Execution* (CTDE), durante treinamento. Como resultado, nós atingimos ao final uma rede composta por 20 VANTs, todos atuando em cooperação para realizar as tarefas mencionadas. A rede apresentou bons resultados nas tarefas de Movimentação para Local Alvo, Manutenção de Conectividade, e Robustez contra Falhas, mas teve um baixo desempenho em Cobertura de Área. Esses resultados sugerem que o uso de MARL é capaz de atingir bom desempenho quando usado para treinar um conjunto de agentes para Missões de Multi-Critérios, mas ainda há espaço para aprimoramentos e exploração de outras técnicas.

Abstract

Unmanned Aerial Vehicles (UAVs) can act in a wide range of scenarios in which sending human personal is of great risk or difficulty, such as exploration and reconnaissance missions in dangerous areas or of extreme conditions for human beings, even in disaster sites or military operations. These missions commonly employ multiple UAVs, forming a decentralized network that allows communication and cooperation between them. For the success of this type of formation, it is necessary to have a robust network topology, that is, capable of maintaining certain characteristics such as connectivity between all agents in the network and resilience to failures. This work deals especially with Multicriteria Missions, characterized by having several distinct activities to be performed during the mission, composing complex scenarios in which robot cooperation is essential for mission success. Such scenarios are good candidates to apply machine learning techniques due to the large number of possible configurations and lack of a model that determines only one optimal solution. This work proposes to apply Deep Reinforcement Learning techniques to train an agent, acting as an UAV, that cooperates with replicas of itself to create a network topology able to perform Multicriteria Missions. The agent was trained following 4 distinct tasks to simulate such kind of mission, as follows: Move to Target Location, Connectivity Maintenance, Area Coverage, and Robustness to Failures. We applied Deep Reinforcement Learning and Multi-Agent Reinforcement Learning techniques, such as Proximal Policy Optimization (PPO) and Centralized Training with Decentralized Execution (CTDE), during training. As a result, we achieved at the end a network composed by 20 UAVs, all of them acting in cooperation to perform the mentioned tasks. The network presented great results in the tasks of Move to Target Location, Connectivity Maintenance, and Robustness to Failures, but underperformed in Area Coverage. These results suggest that the use of MARL is able to achieve good performance when training a set of agents for Multicriteria Missions, but there is still room for improvement and exploration of other techniques.

List of Figures

List of Tables

List of Abbreviations and Acronyms

UAV	Unmanned Aerial Vehicle
AI	Artficial Intelligence
RL	Reinforcement Learning
MARL	Multi-Agent Reinforcement Learning
ANN	Artificial Neural Network
NN	Neural Network
DNN	Deep Neural Network
MDP	Markov Decision Process
DQN	Deep Q-Learning Network
PPO	Proximal Policy Optimization
CTDE	Dentralized Training with Decentralized Execution
MASDPG	Multi-Agent Stochastic Deep Policy Gradient
AR	Accumulated Reward

List of Symbols

$\mathcal{G}$	Graph representing the network of drones
λ	Algebraic Connectivity of the Network
$\overline{\lambda}$	Mean Algebraic Connectivity
$\Theta(\mathcal{G})$	Robustness Level of the Network
$\overline{\Theta}(\mathcal{G})$	Mean Robustness Level
$\vec{v}_a$	Velocity of the drone determined by the NN
$\vec{v}_p$	Velocity of the drone determined by the potential field
$\vec{v}$	Resulting velocity of the drone
R	Observable Radius
α	Learning Rate
α'	Normalized Learning Rate
γ	Discount Factor

Contents

1 Introduction

In this chapter, motivation, contextualization, and objectives of this work are discussed, as well as a brief literature review and the intents of possible contributions of this project. Finally, a brief summary for each chapter is given.

1.1 Motivation

Robotics is a multidisciplinary field in the context of Electronics and Computer Engineering, with a broad range of applications. In the industry, robots are already present in most assembly lines and factories, as shown in Figure 1.1. The benefits of using robots in these processes includes among others: reducing risk to personnel, enhancing environmental and safety issues, and improvements in efficiency of production.

FIGURE 1.1 – Employees work alongside robotic arms, manufactured by Kuka, at Mercedes-Benz's factory in Bremen, Germany. Source: (MCGEE, 2017).

Robots can also be applied outside the industry scope. In Aeronautics, Unmanned Aerial Vehicles (UAVs), a.k.a Drones, already can perform missions of exploration, surveillance and reconnaissance, which can take place in dangerous or extreme condition environments, after disaster events, in inhospitable settings or for military purposes.

Despite robotics has greatly improved in the last decades (HENTOUT, 2019), most of the more impressive tasks performed by robots are still those attached to human control, such as the execution of minimal impact surgeries or military operations. The design of robots capable of proper autonomous decision-making and sub-sequential operation of tasks in real-world applications is still in its infancy. In this context, recent advancements made in Machine Learning, in special Deep Learning and Reinforcement Learning (RL), set the subject as a promising field to unlock even further capabilities in robotics.

To exemplify the recent developments in RL, in 2016, AlphaGo, an Artificial Intelligence (AI) program developed by Google's branch of research in Deep Learning, DeepMind, competed and defeated the then World Champion of the ancient game of Go, Lee Sedol (DEEPMIND, 2016). In 2018, AlphaZero, a reviewed version of the program, learned from scratch to master Go, Chess, and Shogi (DEEPMIND, 2018). Later in 2020, MuZero, another iteration of the program, made a significant step forward in the pursuit of general-purpose algorithms (DEEPMIND, 2020). MuZero was able to master Go, Chess, Shogi, and virtual games for Atari all from scratch, without needing to be told the rules of each game.

All these algorithms lean on Deep Reinforcement Learning techniques, which combines the flexibility of Reinforcement Learning to model sequential decision-making problems and the power of Deep Neural Networks architectures.

In addition to this, the subject of Multi-Agent Reinforcement Learning (MARL) had great development in the last decade, especially in cooperative dynamics. Although most of the research made in MARL so far have been on Competitive AI, such as applications in Game Theory, Cooperative AI for Multi-Agent systems is a field of great promise since it can be found in several scenarios ranging from self-driving cars, and global commerce to daily tasks and pandemic preparedness (DEEPMIND, 2020). Our own evolution as human species depended a lot in our ability to cooperate with one another (BOYD; RICHARDSON, 2009), and the evolution of Cooperative AI should not be different. With an increasing number of AI agents becoming part of our lives, to learn to cooperate with each other and with our own species have become of huge importance in its development.

1.2 Contextualization

Multi-agent systems are defined as one of the main challenges in robotics for the next decade (YANG *et al.*, 2018), and most of what is attributed to its eventual success is the existence of a resilient network topology for these agents. Yang *et al.* define resilience as a "property about systems that can bend without breaking. Resilient systems are self-aware and self-regulating, and can recover from large-scale disruptions". In other words, a resilient network topology has all its members connected, and provides robustness to failures, meaning that even if a node fails, the network will not break into multiple components, and will continue to be able to perform its task.

Several researchers have studied the impact that both inherent dynamics and disturbances from targeted attacks have on the network efficiency and connectivity. This assessment has been made through simulations of node disconnections based on two main criteria: at random and at choosing the better positioned nodes, to mimic, respectively, the network dynamics (failures) and targeted attacks. It is well known that decentralized networks are well adapted to their own dynamics, but highly vulnerable to losses of prominent elements, either due to internal or external forces. Indeed, prominent nodes play a crucial rule to maintain the network working properly (GHEDINI; RIBEIRO, 2011).

This work deals especially with Multicriteria Missions, characterized by having a decentralized network to perform several distinct tasks, sometimes with opposing objectives, composing complex scenarios in which the existence of a resilient network, as well as robot cooperation, are essential for mission success.

An example of such kind of missions are Coverage Missions, in which a network of agents (in our case, a set of drones) cooperates between themselves to explore a region and avoid near obstacles (radars or enemy drones), all while maintaining connectivity with each other and communicating the discovery of possible points of interest. The success of such coverage missions in most cases depends on: area coverage, connectivity maintenance, and robustness to failures.

While area coverage is more closely related to the mission purpose itself, as spreading the nodes of the network to increase the search space, connectivity and robustness mechanisms are more related to safe-guarding the conditions of the network throughout the mission. Independently of the purpose of the mission, the agents should always arrange themselves in a resilient topology that enables them to perform their purpose even when under attacks. Given the extreme conditions in which our agents present themselves, it is acceptable to consider that disturbances from targeted attacks will occur, testing the robustness of the network.

Multicriteria missions compose complicated scenarios that are good candidates to apply machine learning techniques, such as Deep Reinforcement Learning, due to the large number of possible configurations and lack of a model that determines a optimal solution. In this context, we can consider each drone in the network as an agent, being trained for the the purposes of the mission it performs. Also, the network presents a good example of a multi-agent system, where we can apply Multi-Agent Reinforcement Learning techniques to train our agents.

1.3 Objective

This work aims to use state-of-the-art Deep Reinforcement Learning techniques to train an agent that is capable of cooperation with replicas of itself to create a resilient network topology able to perform multicriteria missions. In our scenario, each agent takes the role of a drone in the network, and we set tasks defined for coverage missions, to simulate a multicriteria mission. Given this context, we break the problem as follows:

1. Train an agent that moves from start to target location;
2. Train a network of 20 agents that executes 1 while performing:
    a. Connectivity Maintenance;
    b. Area Coverage;
    c. Robustness to Failures;

The final trained agent performance will be evaluated in terms of Connectivity and Robustness in scenarios with and without the presence of attacks and failures.

1.4 Literature Review

In the context of network topologies for multicriteria missions, we have had different approaches to the problem, from non-AI based algorithms to fully-AI based algorithms.

Ghedini *et al.* (2018) addresses the issue by proposing a combination of control laws that takes into account each objective of the network: connectivity maintenance, collision avoidance, robustness to failure, and area coverage. The resulting model demonstrated that it is possible to determine a set of weights for each control law that achieves a resilient network topology. The topology is able to enhance its coverage performance while avoiding collision and maintaining global network connectivity and robustness to failures of elements.

On another approach, Minelli *et al.* (2018) proposed an online optimization of weights for the control laws devised by Ghedini *et al.* (2018). This outperforms the scenario of static gains in some cases, showing that there is still improvements to be made in the problem.

In the field of Deep Reinforcement Learning, we had several interesting results in the last years, such as AlphaGo (DEEPMIND, 2016) and OpenAI Five (OPENAI, 2018), computer algorithms that achieved human level performance on games as Go and Dota 2, respectively. These advancements were enabled by the developments in model-free RL algorithms, for example Q-Learning, Deep Q-Networks, and Policy Gradient methods, in special Proximal Policy Optimization (PPO). These kind of algorithms do not need to learn a complete model of the environment in which they are acting, but instead just map inputs (states, actions) to expected returns.

Multi-Agent Reinforcement Learning in sequential environments has been a promising field for a while now (LEIBO *et al.*, 2017), in which OpenAI Five can be shown as an example. This algorithm presents five RL agents in a mixed Cooperative and Competitive environment, where they must cooperate to achieve the ultimate goal of winning a game of Dota 2 against a human-only team. Other recent researches in Cooperative AI include research in communication, commitment, and trust between agents (DEEPMIND, 2020).

The application of Deep Reinforcement Learning to network topologies of drones have few but interesting results (GUPTA; NALLANTHIGHAL, 2021). It is shown that a set of drones can be trained to act in a few simple tasks with relative ease.

1.5 Contributions

With this work we propose a Deep Reinforcement Learning framework for training a decentralized network of drones that are able of cooperation between themselves to create a resilient network topology. Our main contribution is to promote a simple Multi-Agent Reinforcement Learning algorithm to expand the field of Cooperative AI, and to train a set of agents, constrained by limited communication range, capable of obstacle avoidance, connectivity maintenance, and robustness to failures in multicriteria missions.

1.6 Outline

The remainder of the work can be described as follows:

- Chapter 2 - Covers Network Theory necessary for problem modeling.
- Chapter 3 - Covers Deep Learning and Reinforcement Learning theory.
- Chapter 4 - Outlines problem modeling and core tools used.
- Chapter 5 - Presents the training results and analysis.
- Chapter 6 - Shares conclusions and suggestions of future works.

2 Network Theory

In this chapter, we cover the theoretical background needed in Network Theory and present the principal evaluation metrics used in this work.

2.1 Networks

Networks are defined as “a concrete pattern of relationships among entities in a social space” (OWEN-SMITH, 2017). Network Theory has a broad range of applications, from communications to social networks apps, or basically any kind of dataset where relationships can be determined between its patterns. For example, Figure 2.1 presents a network that represents the relationships of inventors and inventions in Silicon Valley from 1986 to 1990.

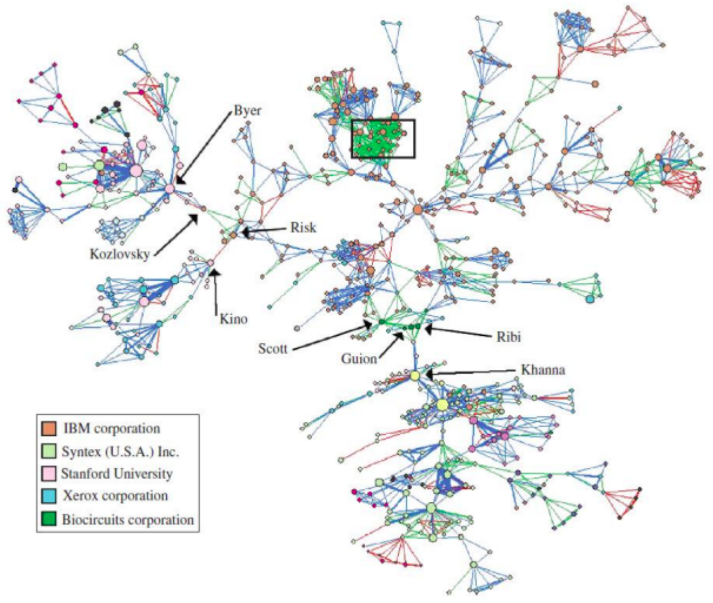


FIGURE 2.1 – Inventors of Silicon Valley’s largest component in 1986 - 1990 by Assignee and Importance of Inventions. Source: (FLEMING *et al.*, 2007).

In the following sections, we go through a few important concepts in network theory for our work.

2.1.1 Algebraic Connectivity

For all following concepts, consider an undirected graph $\mathcal{G} = (V, E)$, and let $\mathcal{L} \in \mathbb{R}^{N \times N}$ be the Laplacian Matrix of graph $\mathcal{G}$, $\mathcal{L}$ is defined as follows:

$$L_{ij} = \begin{cases} \deg(v_i) & \text{if } i = j, \\ -1 & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_j, \\ 0 & \text{ otherwise.} \end{cases} \tag{2.1}$$

Calculating the eigenvalues λ_i, $i = 1, ..., N$, of the Laplacian Matrix, we can extract the following properties (GODSIL; ROYLE, 2001):

- For all eigenvalues λ_i, $\lambda_i \in \mathbb{R}$;
- The eigenvalues can be sorted in a way that $0 = \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_N$.

Let λ be equal to λ_2, the second smallest eigenvalue (counting multiple eigenvalues separately) from the Laplacian Matrix of $\mathcal{G}$, then $\lambda > 0$ if, and only if, $\mathcal{G}$ is a connected graph. The value of λ can serve as indicator of how well connected the overall graph is. In this context, λ is defined as the Algebraic Connectivity of the graph $\mathcal{G}$.

2.1.2 Betweenness Centrality

The Betweenness Centrality (BC) of a node v is a measure of influence this node has in the graph $\mathcal{G}$ regarding the flow of information through nodes (GODSIL; ROYLE, 2001). This metric is usually used to locate bridge nodes, i.e essential nodes for the flow on information passes from one part of a graph to another. The betweenness centrality of node, $g(v)$, can be calculated as

$$g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}, \tag{2.2}$$

where σ_{st} is the total number of shortest paths from node s to node t, and $\sigma_{st}(v)$ is the number of those paths that pass through v.

Given that $g(v)$ scales with the number of nodes in the graph, we can define a normalized betweenness centrality, $g_{\text{normal}}(v) \in [0, 1]$, given by

$$g_{\text{normal}}(v) = \frac{g(v) - \min(g)}{\max(g) - \min(g)}, \tag{2.3}$$

where $\min(g)$ and $\max(g)$ are the minimal and maximum values of BC in the graph, respectively.

Figure 2.2 shows a graphical representation of the betweenness centrality of nodes in an undirected graph as example, it is shown that central nodes have higher flow of information passing through (blue nodes), while nodes in the border have little to no information passing (red nodes).

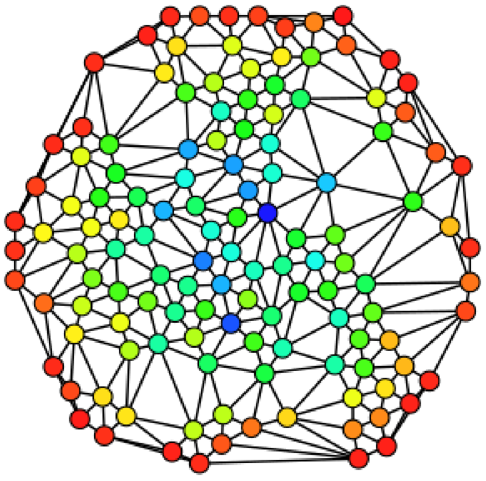


FIGURE 2.2 – An undirected graph colored based on the betweenness centrality of each vertex from least (red) to greatest (blue).

2.1.3 Robustness Level

In network topologies, attacks directed to nodes with high ranking of BC are likely to harm the network connectivity, in the worst case breaking the network into smaller components.

We may define *robustness* in network theory as the ability to withstand failures and perturbations. It is an important attribute that helps to evaluate the resilience of networks to attacks or malfunction (GODSIL; ROYLE, 2001).

Consider a graph $\mathcal{G}$ with N nodes, and let $[v_1, ..., v_N]$ be the list of nodes ordered by descending value of Betweenness Centrality. Let $\phi < N$ be the minimum index such as removing nodes $[v_1, ..., v_\phi]$ leads to disconnecting the graph. We define the Robustness Level of a graph as

$$\Theta(\mathcal{G}) = \frac{\phi}{N}, \tag{2.4}$$

being the number of nodes needed to be removed before the graph disconnects, over the total number of nodes in the graph.

3 Machine Learning

In this chapter, we cover the theoretical background needed in Deep Learning and Reinforcement Learning to understand this work.

3.1 Artificial Neural Networks

The artificial neuron model, shown in Figure 3.1, is a function approximator based on how actual biological neurons work. Artificial neurons are the basic building blocks of artificial neural networks (ANN). The artificial neuron receives weighted inputs and a bias and outputs a linear combination of these factors, followed by a non-linear activation function.

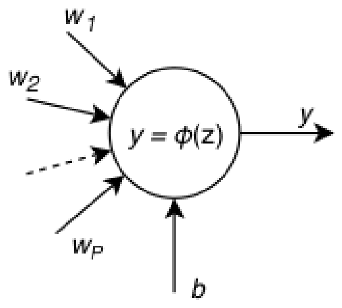


FIGURE 3.1 – Representation of the artificial neuron model as an approximator of a mathematical function. Source: (GOODFELLOW *et al.*, 2016).

Its mathematical representation is given by,

$$z = \left(\sum_{i=1}^{P} w_i x_i \right) + b \Longrightarrow y = \phi(z), \tag{3.1}$$

where each x_i is an input, w_i the equivalent weight of input i, z the internal state of the neuron, $\phi(.)$ a non-linear activation function, and y the output of the neuron.

A summarized version of z, defined previously in (3.1), also called vectorized form of z, is given by the equation

$$z = \mathbf{w}^{\mathbf{T}}\mathbf{x} + b. \tag{3.2}$$

This version of the equation allows us to work with stacks of neurons and maintain the compact notation. These stacks of neurons can be organized in what is called a *layer*. A collection of one or more layers, in which the output of one is the input of another, constitutes a Neural Network, as shown in Figure 3.2.

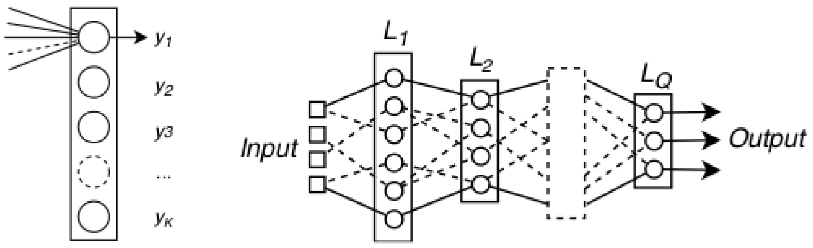


FIGURE 3.2 – Example of layer with stacked Artificial Neurons (left). Example of multi-layer Neural Network (right). Source: (GOODFELLOW *et al.*, 2016).

A neural network can have a certain number of layers l, where each layer is fully connected to the immediate previous layer. Each layer can also have a different number of neurons and different activation function $\phi(.)$.

There are three different types of layers: Input, Output, and Hidden layers, the first two are self-explanatory, while the third one is defined as a layer that is simply in between of two other layers. In Figure 3.2, the neural network on the right contains an input layer, followed by several hidden layers L_1, L_2, ... , L_Q, followed by an output layer.

3.2 Deep Learning

Most of the problems that Machine Learning models are being applied to are complex and abstract problems, such as self-driving cars, virtual assistants, and speech recognition. These kind of problems need a certain abstraction from the machine to understand for example what a car is, or what the human voice is. These concepts cannot be hard-coded on the model since they are extremely complex tasks on their own. To tackle these problems we lean on Deep Learning, a paradigm that allows computers to learn from experience and understand the world in terms of a hierarchy of concepts, with each concept defined through its relation to simpler concepts.

Deep Learning enables machines to create layers of abstraction on the data, learning complicated concepts by building on top of simpler ones. This hierarchy is built without the need of human intervention and explanation of the world, the machine can learn by itself all that it needs from the world to perform its given task.

3.2.1 Deep Neural Networks

We can build a Deep Neural Network (DNN) by stacking multiple layers of neurons on top of each other, as shown in Figure 3.3. In the literature, a neural network with 2 or more hidden layers can already be considered a deep neural network (GOODFELLOW *et al.*, 2016).

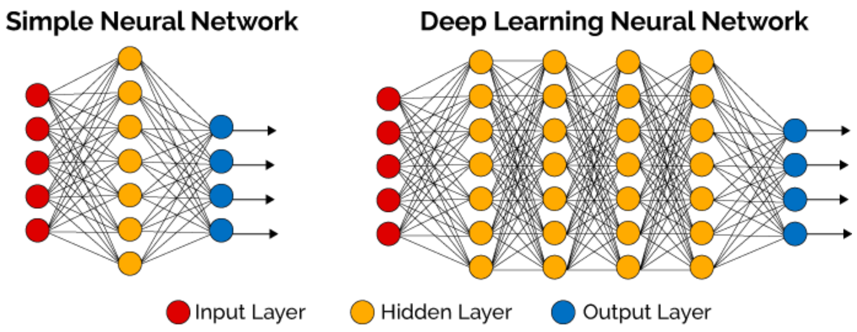


FIGURE 3.3 – Comparison of a Shallow Neural Network (left) to a Deep Neural Network (right).

On DNNs, similar to simple neural networks, the main objective is that the model learns a function ψ that solves a specific problem. For that, we create a model $y = f(x, \theta)$, where θ is a vector of parameters of the model, that can approximate $\psi(x)$ with the help of an optimization algorithm. The optimization algorithm updates the values of θ at each training iteration so that the model can learn the best representation of ψ.

The reason why Deep Neural Networks work so much better than their shallow versions is that increasing the number of layers also increases the power of abstraction of the network, giving it the capacity to tackle a broad range of problems outside the scope of simple regression, classification, or clustering.

3.3 Reinforcement Learning

Reinforcement Learning is a broad class of learning algorithms that differs from Supervised and Unsupervised learnings alike. RL can be defined as a structure where *agents* learn to perform certain *actions* on an *environment* in order to maximize its expected return (SUTTON; BARTO, 2018).

3.3.1 Key Definitions

As mentioned, RL defines agents and environments. Agents are those who are trying to learn about the environment in which are present while interacting with it. Environments represents the problem to be solved.

Figure 3.4 presents a common framework to understand the dynamics of RL. An agent at state S_t chooses to perform an action A_t upon the environment at time t, transitioning to a next state S_{t+1} and receiving a reward R_{t+1}.

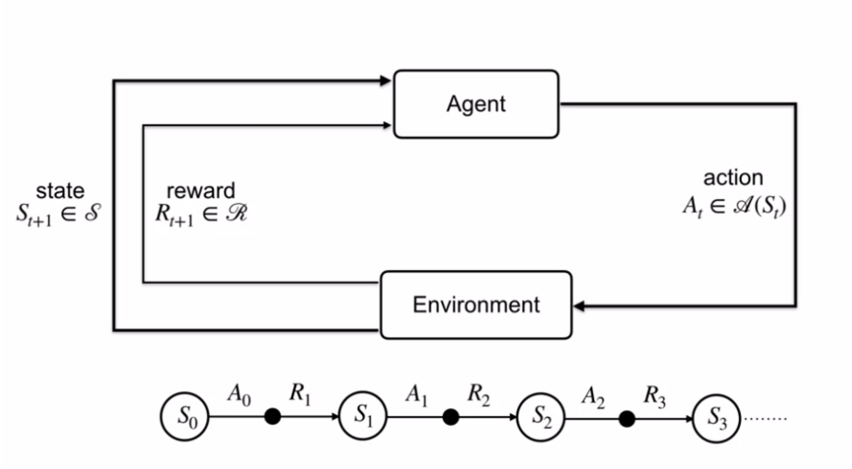

Agent
Environment
state
$S_{t+1} \in \mathcal{S}$
reward
$R_{t+1} \in \mathcal{R}$
action
$A_t \in \mathcal{A}(S_t)$
S_0 A_0 R_1 S_1 A_1 R_2 S_2 A_2 R_3 S_3 ....

FIGURE 3.4 – Reinforcement Learning framework of agent – environment interaction. Source: (SUTTON; BARTO, 2018).

The sets of all possible states, actions, and rewards are given by $\mathcal{S}, \mathcal{A}$, and $\mathcal{R}$, respectively. In an finite Markov Decision Process (MDP), these sets are also finite.

The decision to take a certain action given a current state is determined by a *policy* π. A policy π is a function $\pi : \mathcal{S} \to \mathcal{A}$, from the state space to the action space, which can be deterministic or stochastic.

Deterministic policies are defined as

$$\pi(s) = a, \tag{3.3}$$

where for each state there is only one possible action.

Stochastic policies return a probability distribution over the action space given a current state, as given by

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]. \tag{3.4}$$

This way, an agent at time t and state s_t has a probability p_i of performing the action a_i. Also, from now on, due to the scope of the work, we are going to treat all policies as stochastic policies.

For each possible action at a given state a reward r_{t+1} can be calculated, that represents how good or bad that decision was. At the end of all iterations, a total return G_t of the policy π can be calculated as

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}. \tag{3.5}$$

The return G_t evaluates how good was this policy for the agent throughout the entire sequence of actions, also called an episode. In an optimal scenario, an agent at time t and state s_t would always take an action a_t which maximizes the return G_t received at the end of the episode. The constant γ is a discount factor, $\gamma \in [0, 1]$, that transforms G_t in a weighted sum in time. If γ is closer to 1, all future rewards are taken into account when performing an action, and if γ is closer to 0, the agent tends to make decisions based on more immediate rewards (SUTTON; BARTO, 2018).

To evaluate a given policy at state s_t, we can use the *state-value function* V_π, that maps a state to a expected *value*. Since a policy return a probability distribution over all possible actions, we calculate the value of a state s with a policy π as the expected return through all possible actions, as

$$V_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]. \tag{3.6}$$

We can also calculate the expected value for each action separately in a state with policy π using the *action-value function* Q_π. The notation is similar and it is given by

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]. \tag{3.7}$$

For the environment, the only outcome of an agent performing an action is the state transition, and this is independent of the agent policy. To calculate the state transition, we can use the *State Transition Matrix* $\mathcal{P}$, which is defined as

$$\mathcal{P}_{a,s,s'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]. \tag{3.8}$$

3.3.2 Markov Decision Processes

Markov Decision Processes (MDPs) are a formalization of sequential decision-making, where actions influence not just immediate rewards, but also subsequent states and rewards. With MDPs we can model problems with delayed rewards and the trade-off between immediate and delayed reward (SUTTON; BARTO, 2018).

A MDP is defined by a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$, in which $\mathcal{S}$ is the set of all possible states, $\mathcal{A}$ is the set of all possible actions, $\mathcal{P}$ is the state transition matrix, $\mathcal{R}$ is the reward function, and γ is the discount factor.

The modeling of a RL problem as we have seen follows the dynamics of a MDP.

It is important to frame that for a RL problem be feasible, its MDP must satisfy the Markov Property, in other words, the problem must be a stochastic process in which the transition from the current state to the next state is independent of past information about the process. The Markov Property is defined by

$$\mathbb{P}[S_{t+1} = s'_t | S_t = s_t] = \mathbb{P}[S_{t+1} = s'_t | S_t = s_t, S_{t-1} = s_{t-1}, ..., S_1 = s_1]. \tag{3.9}$$

3.3.3 Policy Gradient Methods

So far we have used a table to map states to actions, for policy gradient methods we substitute this table with a function approximator, in our case a neural network with parameters θ, that receives as input the current state and outputs the action.

Policy gradient methods are a type of RL algorithms that optimizes the policy's parameters θ with respect to G_t by gradient descent or ascent. These kind of methods are especially efficient in environments with continuous states and actions. Using gradient methods in continuous states and actions also guarantees convergence at least to a local optimum (SUTTON; BARTO, 2018).

In mathematical notation, the cost function $J(\theta)$, given by

$$J(\theta) = \mathbb{E}_{\pi_\theta}[G_t], \tag{3.10}$$

is maximized with gradient ascent to achieve the optimal policy,

$$\theta_{k+1} = \theta_k + \alpha_\theta \nabla_\theta J(\theta), \tag{3.11}$$

where α_θ is the learning rate for the parameters θ.

Although policy gradient methods are capable of solving the Reinforcement Learning problem, they have two main issues. Firstly, policy gradients tend to take a long time to converge and have high variance. Additionally, they usually converge only to a local optimum (SUTTON; BARTO, 2018). Secondly, it is troublesome to obtain a good estimator of the policy gradient $\nabla_\theta J(\theta)$.

One way to approximate this gradient, without having to model every detail of the system, is showed below. We start re-writing the gradient of the cost function $J(\theta)$,

$$\nabla_\theta J(\theta) = \int \nabla_\theta \pi_\theta(\tau) G_t(\tau) d\tau. \tag{3.12}$$

Now, we can use the identity showed in (3.13) to rearrange (3.12) in the next step,

$$\nabla_\theta f(\theta) = \frac{f(\theta)\nabla_\theta f(\theta)}{f(\theta)} = f(\theta)\nabla_\theta \log(f(\theta)), \tag{3.13}$$

$$\nabla_\theta J(\theta) = \int \nabla_\theta \pi_\theta(\tau) G_t(\tau) d\tau = \int \pi_\theta(\tau)\nabla_\theta \log(\pi_\theta(\tau)) G_t(\tau) d\tau, \tag{3.14}$$

where (3.14) can be simplified as

$$\nabla_\theta J(\theta) = \mathbb{E}_{\theta\pi}[\nabla_\theta \log(\pi_\theta) G_t], \tag{3.15}$$

which allows a sampling technique for estimating the gradient. This result is also called Policy Gradient Theorem (SUTTON; BARTO, 2018).

3.3.4 Actor-Critic Method

One way to improve the policy learning methods is to apply both concepts of value-based and policy-based methods at the same model, this kind of approach is called Actor-Critic Method.

In this method there are two models operating at the same time, as defined:

- *Actor*: model that interacts with the environment and suggests actions given the current state with parameters θ_a;
- *Critic*: model that receives as input the state and the action taken by the Actor and calculates the Q-Value for that scenario with parameters θ_v.

For the Critic to be able to evaluate the Q-value of the Actor's actions, we define the Advantage Function $A_\pi(s, a)$, given by

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s). \tag{3.16}$$

The value of $V_\pi(s)$ takes into consideration all possible actions for that state, so it computes the average rewards of actions, while $Q_\pi(s, a)$ computes the value for a specific action, so the difference calculated by the advantage function acts as an estimate of how good or bad an action is to a given state. $V_\pi(s)$ acts on the equation as a baseline parameter.

The output of the function can be used by the actor to updates its own parameters. If A_π is positive, the gradient ascent algorithm will update the actor's parameters towards that direction, and if it is negative the parameters will follow in the opposite direction.

To optimize the time spent calculating the values of action-value and state-value functions, we can use the Temporal Difference approximation

$$A_\pi(s,a) \approx r + \gamma V_\pi(s') - V_\pi(s). \tag{3.17}$$

Having two models to solve the RL problem greatly increases the performance of the system in comparison to each model solving it separately. The Actor-Critic algorithm is presented in Algorithm 1.

Algorithm 1 Actor-Critic Model

```
Initialize step counter: $t \leftarrow 0$
Get actor and critic parameters: $\theta_a$ and $\theta_v$
Get initial state: $s_t$
while $t < T_{max}$ do
    // Actor-Critic Interaction
    Sample action $a_t$ from $\pi(a_t|s_t;\theta_a)$
    Sample reward $r_{t+1}$ from $R(s,a)$
    Sample state $s_{t+1}$ from $P(s'|s,a)$
    // Model Updates
    Compute TD approximation: $A_\pi(s,a) \approx r + \gamma V_\pi(s') - V_\pi(s)$
    Update Actor Parameters: $\theta_a \leftarrow \theta_a + \alpha_a A_\pi(s,a)\nabla_a \log(\pi_\theta(a|s))$
    Update Critic Parameters: $\theta_v \leftarrow \theta_v + \alpha_v A_\pi(s,a)\nabla_v V_v(s)$
    $t \leftarrow t+1$
end while
```

3.3.5 Proximal Policy Optimization

To refine even more our model, we can use Proximal Policy Optimization (PPO) for updates on the parameters θ. PPO is a state-of-the-art reinforcement learning algorithm that is becoming popular due to being much simpler to implement and tune than other algorithms (SCHULMAN *et al.*, 2017).

In general, policy gradient algorithms are extremely sensitive to the learning rate. A learning rate too small and the learning becomes too slow, while a higher learning rate introduces noise and variance to the process, making learning unstable.

There have been attempts to solve this issue with algorithms like Trust Region Policy Optimization (TRPO), see Schulman *et al.* (2015) for details, and Sample Efficient Actor-Critic with Experience Replay (ACER), see Wang *et al.* (2016) for details. But each of these algorithms had their own annoyances.

PPO derives from TRPO, but with a much simpler approach. PPO is an on-line policy algorithm that computes at each iteration a clipped update of parameters θ that minimizes the cost function L. This clipping is to ensure that the deviation from the previous policy is relatively small avoiding too much variance.

The clipped cost function, also called clipped surrogate objective function (SCHULMAN *et al.*, 2017), is given by

$$L^{CLIP}(\theta) = \mathbb{E}_{\theta,\pi}[\min(r_t(\theta')A_w, \text{clip}(r_t(\theta')A_w, 1-\epsilon, 1+\epsilon))], \tag{3.18}$$

where $r_t(\theta')$ is defined by

$$r_t(\theta') = \frac{\pi_\theta(a|s)}{\pi_{\theta'}(a|s)}, \tag{3.19}$$

with $\pi_\theta(a|s)$ and $\pi_{\theta'}(a|s)$ being the new updated policy and the previous policy, respectively. And $[1-\epsilon, 1+\epsilon]$ is the bounded update interval for policy π_θ, with ϵ being 0.1 or 0.2 usually (SCHULMAN *et al.*, 2017). We can see its implementation with Actor-Critic in Algorithm 2.

Algorithm 2 Proximal Policy Optimization with Clipped Objective Function

```
Initialize step counter: $t \leftarrow 0$
Get initial parameters: $\theta_0$
**while** $t < T_{max}$ **do**
    // Actor-Critic Interaction
    Collect set of trajectories $\mathcal{D}_k$ on policy $\pi(a|s, \theta_k)$
    Compute TD approximation: $A_{\pi,k} \approx r + \gamma V_\pi(s') - V_\pi(s)$
    Update policy parameters: $\theta_{k+1} \leftarrow \text{argmax}_\theta(L^{CLIP}(\theta_k))$
    $t \leftarrow t + 1$
**end while**
```

3.3.6 Multi-Agent Reinforcement Learning

In the Multi-Agent Reinforcement Learning framework, the environment is acted upon by all agents at the same time, returning rewards for each of them in each iteration. Algorithm 3 below presents how this interaction occurs.

Algorithm 3 Multi-Agent Reinforcement Learning Model

```
Initialize set of N agents
Initialize step counter: $t \leftarrow 0$
Observe initial state for each agent: $s_0 = (s_{1,0}, ..., s_{N,0})$
**while** $t < T_{max}$ **do**
    For each agent $i$, select action $a_{i,t} \sim \pi(a_t|s_{i,t})$
    Execute simultaneously the actions $a_t = (a_{1,t}, ..., a_{N,t})$
    $s_t \leftarrow$ vector of agents' observations of the environment
    $R_t \leftarrow$ sum of all agents' generated reward
    Update network parameters
    $t \leftarrow t + 1$
**end while**
```

Multi-Agent Reinforcement Learning (MARL) aims to simultaneously train multiple agents to solve a given task in a shared environment. Now, the computational cost of training several agents at the same time increases exponentially with the number of agents, same with the variance during training, so a common technique to scale down this number is *parameter sharing*, where agents share some or all the same parameters of the neural network model (CHRISTIANOS *et al.*, 2021).

Naive parameter sharing is the common approach to problems where agents have similar tasks and expected behaviors, so they share all the same state-value function, action-value function, and reward function. The similarity between agents helps them to share the same abstractions within the neural network.

3.3.7 Centralized Training with Decentralized Execution

A popular paradigm in MARL is *Centralized Training with Decentralized Execution* (CTDE), in it we assume that during training all agents have access to the world state, and even can access states from other agents. On testing, we revoke the world state access of agents and they can only observe their own state.

CTDE algorithms have outperformed non-CTDE algorithms served as benchmarks in several tasks (CHRISTIANOS *et al.*, 2021).

An CTDE algorithm that gained popularity recently is the Multi-Agent Deep Stochastic Policy Gradient (MADSPG), which uses naive sharing of parameters to train the agents (KASSAB *et al.*, 2020). MADSPG is built upon the Actor-Critic method, where we optimize the parameters θ_v and θ_c. It basically sums the contribution of each agent as a $d\theta_a$ and $d\theta_v$ before updating the parameters.

At training, for a set of N agents, the algorithm samples all actions $\pi(a|s_i)$, for $i = 1, ..., N$, where $\pi(a|s_i)$ is the policy of the agent i at state s_i, and observe the resulting reward and state. Here the state s_i contains the state relative to the agent i, given that it has knowledge of the world state. The update of the actor-critic parameters is made with the resulting reward, this way each agent contributes to the update equally.

In this training framework, we set a normalized learning rate $\alpha' = \alpha/N$, so that every agent contributes with a fraction of the learning rate α. We divide by the number of agents to avoid increasing the variance in learning, which might cause the model never to converge.

4 Methodology

In this chapter, we cover the methodology to be used in this work, from tools and frameworks applied, to training tasks, and evaluation metrics.

4.1 Tools and Frameworks

In this work we used several tools and frameworks to implement the MARL model, such as PyGame, Stable Baselines 3, Petting Zoo, and Intel® DevCloud oneAPI. Below are brief summaries of these tools at our disposal.

4.1.1 PyGame Library

PyGame is a Python library designed for writing video games. PyGame allows you to create fully featured games and multimedia programs in the Python language (PYGAME, 2022).

For the visualization and evaluation of episodes, we have built a simulator using the PyGame library. Figure 4.1 shows an example of a evaluation episode, in which we have 20 drones in starting position, and 100 obstacles spread throughout the field. At the end we have the target location to which all drones should head. The yellow area represents the current area being covered by the set of drones.

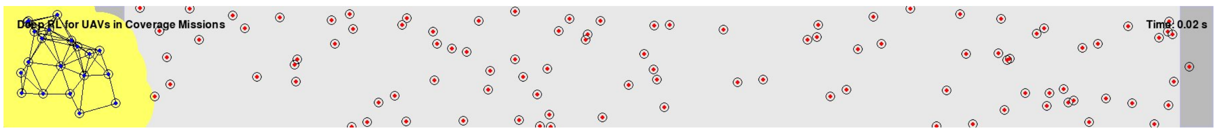


FIGURE 4.1 – Simulator built with Pygame library for evaluation.

This simulator is used for occasional visualization and evaluation of episodes, and is deactivated during training for performance purposes.

4.1.2 Stable Baselines 3

OpenAI is a non-profitable organization that designs and distribute several RL algorithms for research and educational purposes. OpenAI Baselines is a platform created by OpenAI for testing RL algorithms, originally created to be bug-free and to serve as a benchmark for these types of algorithms. The platform helps researchers so that they would not have to waste time in tuning algorithms and looking for bugs, instead of actually advancing in their research.

In 2018, a fork of OpenAI Baselines github repository was created, named Stable Baselines. This fork was generated by a group of developers with the goal of unifying the interface of the several algorithms available, and also fix some bugs and coverage issues.

For this work, we use the Stable Baselines 3 platform for the implementation of Proximal Policy Optimization.

4.1.3 PettingZoo

PettingZoo is a Python library by Farama's Foundation for development and research in multi-agent reinforcement learning, and act as a version of multi-agent Gym framework, creating environments that can interact with several agents simultaneously.

PettingZoo also enables us to work with several environments in parallel with multi-threading in order to increase training experience, this way we can train up to 8 environments in parallel, providing 8 times the number of updates that would be generated in a single run.

4.1.4 Intel® DevCloud oneAPI

The Intel® DevCloud oneAPI is a solution developed by Intel® to allow researchers to develop and train machine learning models on powerful clusters of machines. The oneAPI serves as a cloud-based free sandbox for testing models that require immense computational power (INTEL, 2022).

The platform is helpful for us since we are dealing with the training of Deep Neural Networks in the context of multi-agent systems, so this extra computational power can greatly reduce the training time required for our model.

Intel® provides up to four clusters on a free account that runs in parallel and also provides access to dedicated GPUs and TPUs, if necessary.

4.2 Problem Modeling

We model our problem as a graph $\mathcal{G}$, where $\mathcal{V}(\mathcal{G})$ and $\mathcal{E}(\mathcal{G})$ are the sets of all nodes and edges of $\mathcal{G}$, respectively. In this scenario, each node $v_i \in \mathcal{V}(\mathcal{G})$ represents an agent (drone) of our network, and every edge $e_{ij} \in \mathcal{E}(\mathcal{G})$ represents an available interaction between agents i and j. An interaction between two agents is established once they are within each other observable radius.

The network traverse a field of dimensions $50 \times 500\ m$, where are placed 100 randomly distributed obstacles. Also, we have 200 possible initial scenarios, devised by Ghedini *et al.* (2018), that marks the initial positions of all 20 drones in the field and the position of all 100 obstacles.

Each drone is estimated to have a mass $M = 10\ Kg$ and is treated as a point in space, without dimensions. The observable radius of each drone is given by $R = 16\ m$, and to avoid collisions between drones or other objects, we set a minimum distance $d_{min} = 3\ m$ between the drone and its surroundings. Each drone has a speed limit $v_{max} = 0.6\ m/s$, needing about 900 seconds to traverse the field while avoiding obstacles, giving us an episode length of 15 seconds when running at a rate of 60 Hz.

Given the context of the problem, we model our agents to have a local-ranged communication only, i.e. each agent can only access information from those drones within a distance of 2 hops from it. Thus, each drone is capable of observing the positions of all drones and obstacles within its observation radius, plus the ones that its neighbors can observe.

Figure 4.2 presents an example of a network of drones with 2-hops information access. In this network, node v_1 has access to 2-hops information from its neighbors, being able to locate nodes v_2, v_3, v_5 and v_6, and also all obstacles within their observation range.

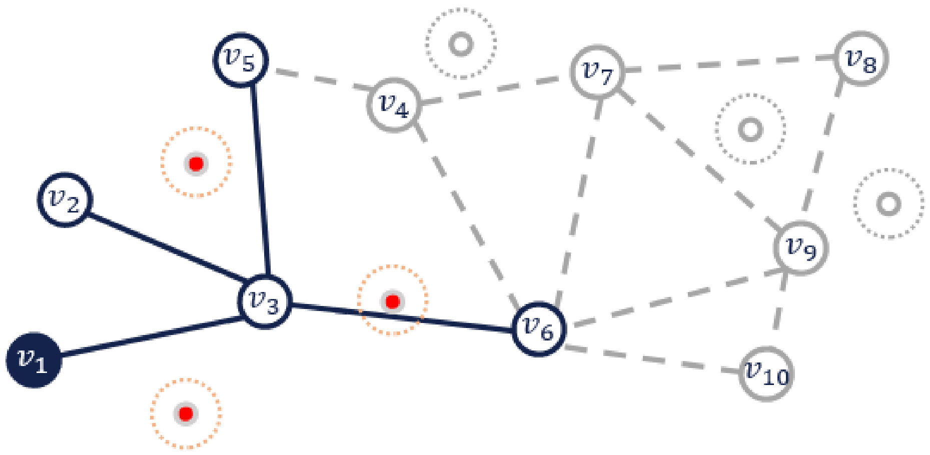


FIGURE 4.2 – Example of 2-hops information in a network of drones.

4.2.1 Collision Avoidance

In order to avoid the collision between drones and obstacles, or even between themselves, we model a high-level controller that acts upon the agent's Neural Network output. The controller is implemented as a potential field that creates a repulsion force $\vec{f}_{rep}$ between drones and obstacles, given by

$$\vec{f}_{rep} = \begin{cases} -k_{rep}\frac{1}{d-d_{min}}\vec{r} & \text{if } d < R, \\ \vec{0} & \text{otherwise,} \end{cases} \tag{4.1}$$

where R is the Observation Radius of a drone, d_{min} the minimum distance to maintain from obstacles and other drones, and $\vec{r}$ a unitary vector giving the direction of the force.

The agent's velocity is composed by two parts, $\vec{v} = \vec{v}_a + \vec{v}_p$, one given by the output of the model's neural network, $\vec{v}_a$, and another from the acceleration produced by the resulting potential field $\vec{F}_p = \vec{F}_n + \vec{F}_o$, consisting of the sum of the resulting potential field due neighbors, $\vec{F}_n$, and due obstacles, $\vec{F}_o$, being calculated as $\vec{v}_p = \vec{a}_p * \Delta t = \frac{\vec{F}_p}{M} * \Delta t$.

Also, if the potential field is not enough to make the agent avoid stepping into an obstacle, we devised a controller to check if the next action is valid. This way we can control the agent not to hit obstacles and to maintain itself within the field boundaries.

4.2.2 Problem Modelling as a Markov Decision Process

Additionally, the problem is modeled as a MDP, with multiple agents interacting with the environment, and follows a Deep Reinforcement Learning implementation. Next, we define each component of the tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$.

4.2.2.1 State Space

In our scenario, due to its movement, each drone has a varying number of neighbors and obstacles in its observation range at each time step. This causes it to need an alternative method of how to interpret these objects in order to feed the neural network a fixed input. For this, instead of passing the original positions of neighbors and obstacles, we devised an abstraction that matches our purposes with the resulting potential field vectors of neighbors and obstacles.

By using the resulting potential field components due to neighbors the agent can have an estimation of its position relative to the network, in terms of direction and absolute distance. We devised a similar metric for obstacles, but in this case only with the closest obstacle to the agent. We decided to feed only the closest one so that the agent do not fall in local minimums, generated by the joint action of multiple obstacles.

Alongside these two metrics, we also feed the agent its position in the field, its own Normalized Betweenness Centrality, so it can identify if it is a prominent node or not, and network's Algebraic Connectivity, so that it identifies if the network is close to disconnect or not, resulting in the following State Space:

- Position: $[p_x, p_y] \in [0, W] \times [0, H]$;
- Normalized Betweenness Centrality of node v: $g_v \in [0, 1]$;
- Algebraic Connectivity: $\lambda \in [0, N]$;
- Resulting potential field due to neighbors in agent's position: $\vec{F_n} = [F_{nx}, F_{ny}]$;
- Potential field due to closest obstacle in agent's position: $\vec{F_o} = [F_{ox}, F_{oy}]$.

In training the drones have a universal perception of the environment, so the resulting potential field due to neighbors take into account all drones in the network, while in evaluation it takes into account only the drones with distance of 2 hops from it. There is also a change in the Betweenness Centrality of a node and its Algebraic Connectivity, which are replaced by local estimates using the information available within a distance of 2 hops.

4.2.2.2 Action Space

The Action Space consists of the velocity of the agent, $\vec{v}_a = [v_{ax}, v_{ay}]$, determined by the neural network's output.

4.2.2.3 State Transition Matrix

Given we are working with a function approximator instead of a tabular method, the state transition probability matrix $\mathcal{P}$ is encoded in the simulation environment and it is not known by the agent.

4.2.2.4 Reward Function

The rewards functions were tailored individually for each training task, and are presented in Section 4.4.

4.2.2.5 Discount Factor

In this problem, we set a discount factor of $\gamma = 0.99$.

4.2.3 Failures on the Network

Our model focus on failures of prominent nodes (attacks) in a way that every episode ends with only 30% of the network remaining.

For the purposes of our work, we only enable attacks on the network after reaching the last task, Robustness to Failures, where we deal with 20 agents in the network. This way, for a network of 20 drones, only 6 will remain by the end of each episode.

Since the episode is bounded by a 15 second window, and with 14 attacks to occur, we have 1 attack/sec, removing from the network always the node of highest Betweenness Centrality at the time, simulating an attack on that drone.

4.3 Multi-Agent Reinforcement Learning Model

In order to save us the work of training 20 different agents with 20 different neural networks, which would greatly increase the computational cost needed, we implement the Naïve Parameters Sharing method with the Multi-Agent Deep Stochastic Policy Gradient (MASDPG) model, in which all agents share the same network parameters, and every agent equally contributes to parameters update during training.

Figure 4.3 presents a schematic of Naïve Parameters Sharing, in it all agents share the same parameters θ of the Deep Neural Network model that represents the policy $\pi(a|s)$.

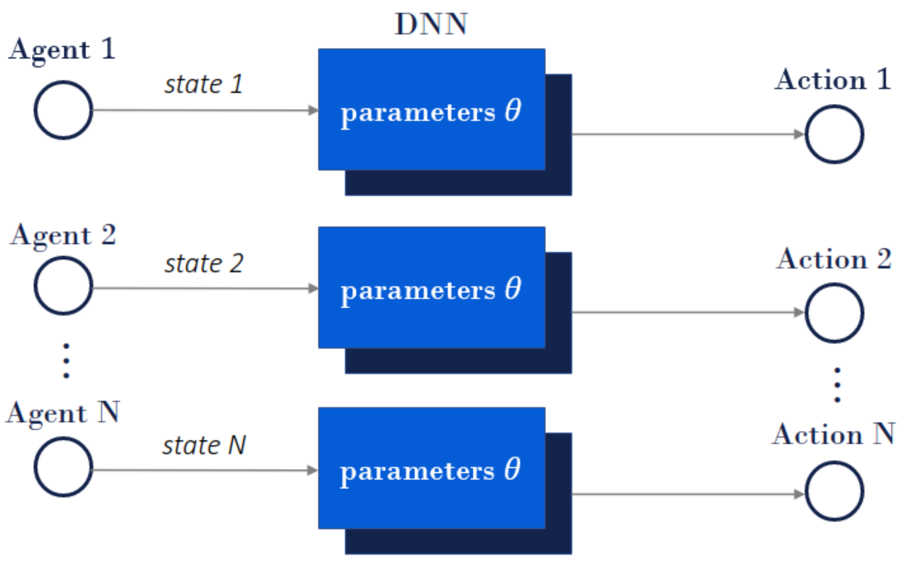


FIGURE 4.3 – Example of N agents using the same DNN's parameters θ with Naïve Parameters Sharing in a MADSPG model.

4.3.1 Policy Parameters

PettingZoo's Actor-Critic policy instantiates two networks that do not share the same parameters, but use the same learning rate, thus $\alpha^\theta = \alpha^W = \alpha$. For the policy optimizer algorithm, we lean on Stable Baselines 3 PPO implementation.

Table 4.1 presents the default values of hyperparameters used in PettingZoo's Actor-Critic policy and Stable Baselines 3 PPO's implementation.

TABLE 4.1 – Actor-Critic and PPO's Default Hyperparameters.

Hyperparameter	Description	Default value
Learning Rate (α)	Rate of updates to the policy's parameters	$3 \cdot 10^{-4}$
Epochs	Number of epochs when optimizing the surrogate loss	10
Batch Size	Minibatch size	64
Timesteps per update (T_{update})	Number of steps to run for each environment per update	2048
PPO Clip (ϵ)	Policy ratio clipping parameter	0.2
Entropy Coefficient (ϵ_{coeff})	Entropy coefficient for the loss calculation	0.01
Discount Factor (γ)	Discount factor	0.99
General Advantage Estimation (λ)	Factor for trade-off of bias vs variance for GAE	0.95
ADAM (ϵ)	Numerical stability	10^{-5}
Neural Network Architecture	Number of neurons in each layer	$[64, 64]$
Activation Function ($\phi(.)$)	Non-linear activation function	tanh

Most hyperparameters were left as default in this work, but to implement the MARL framework a few adjustments were necessary, as setting the learning rate α as α/N for each training task with a N agents on the field or defining the network's architecture.

Also, given an episode length of only 900 timesteps, the value of T_{update} is set to 900, so that we have an update on the policy every ending of episode, and a Batch Size of 60, to match the new value of T_{update}.

4.3.2 Actor-Critic Network Architecture

Following the Actor-Critic model of Reinforcement Learning, we implemented two networks separately, but with the same architecture, presented in Table 4.2.

TABLE 4.2 – Actor-Critic Model Network's Architecture.

Layer	# Neurons	Activation
Input Layer	8	tanh
Hidden Layer 1	32	tanh
Hidden Layer 2	32	tanh
Hidden Layer 3	16	tanh
Output Layer	2	tanh

The network architecture was defined after running the first training task with 15 different network configurations, varying the number of hidden layers and number of neurons per layer, and was selected by comparing time spent to converge and overall performance of the model once trained.

Table 4.3 presents the altered training hyperparameters.

TABLE 4.3 – Model's Specific Hyperparameters.

Hyperparameter	Description	Default value
Learning Rate (α)	Rate of updates to the policy's parameters	$(3/N) \cdot 10^{-4}$
Batch Size	Minibatch size	60
Timesteps per update (T_{update})	Number of steps to run for each environment per update	900
Neural Network Architecture	Number of neurons in each layer	$[32, 32, 16]$

4.4 Training

As mentioned in Section 1.3, we can derive smaller objectives to achieve our main goal. First, implement a potential field algorithm that enables our agent to avoid collisions with other drones and obstacles. With this we have the basic physics implemented in the environment so that the agent can start its training.

Second, start training using CTDE with the objective of moving a small set of agents from start to end positions in the field. Then, we begin gradually increasing the number of agents in each training task as we begin to tackle other behaviors, this is because some metrics as robustness level and algebraic connectivity perform better with a large number of nodes. In our case, we cannot choose to increase the number of drones with a simple task and then train the remaining behaviors like Area Coverage and Robustness to Failures, for they will most likely not converge to an optimal solution, given that there will be high variance due to 20 different agents acting each time step, neither can we train all behaviors with a small number of drones and then increase then with time, for some metrics as Robustness Level does not give much information for a 2 or 3-drones network for example.

In this case, we order the tasks that the network learn: Connectivity Maintenance, Area Coverage, and Robustness to Failures. We train the first task until the network reaches 10 drones, then we introduce the reward for Area Coverage. Once the network reaches 20 drones, we introduce Robustness to Failures and disturbances in the network. The attacks are introduced only later so that the network is matured enough in the other behaviors, since an attack in smaller networks would highly affect the training. We use the Betweenness Centrality of the nodes to target the attack, always focusing in attacking the node with higher BC, this way trying to disrupt as much as possible the network.

The environment in which the agents train is a field with 100 randomly generated obstacles throughout the field following Ghedini *et al.* (2018) scenarios. The final objective is that the network traverse this field left to right, maximizing the area coverage, maintaining connectivity, and improving robustness to failures.

During training, we take advantage of Intel® DevCloud's oneAPI solution to optimize our Deep Reinforcement Learning model, using a cluster of 8 GPU-enabled machines. The oneAPI servers act as 8 environments running in parallel, so at the end of a single episode the model receives the equivalent experience of 8 episodes, speeding up training. With this, when we train our model for 2,000 episodes for example, the experience gathered is equivalent to 16,000 episodes. This number increases even more when we consider that each agent contributes to training equally, so for a network of 20 drones, the number of equivalent episodes completed grows to 320,000.

4.4.1 Task 1 – Move to Target Location

As discussed in Section 4.1.1, we implement a simple potential field for each drone and each obstacle to avoid collision, being the only constraints existing in the environment. The following repulsion functions were used for drones, and obstacles, respectively

$$\vec{f}_{rep,drones} = \begin{cases} \frac{-1}{d-3}\vec{r} & \text{if } d < 16, \\ \vec{0} & \text{otherwise,} \end{cases} \tag{4.2}$$

$$\vec{f}_{rep,obstacles} = \begin{cases} \frac{-2}{d-3}\vec{r} & \text{if } d < 16, \\ \vec{0} & \text{otherwise.} \end{cases} \tag{4.3}$$

With the physics of the environment implemented, we are able to start the training of Move to Target Location task with a set of 2 agents and for 2,000 episodes.

Figure 4.4 presents one initial scenario for a network composed by 2 agents. Given that the potential field created by the obstacles on the right diminishes the chances that the network will ever reach the target location by random exploration, we implement a reward function that instead gives a penalty for staying of the left side of the field instead of an reward for reaching the far end side.

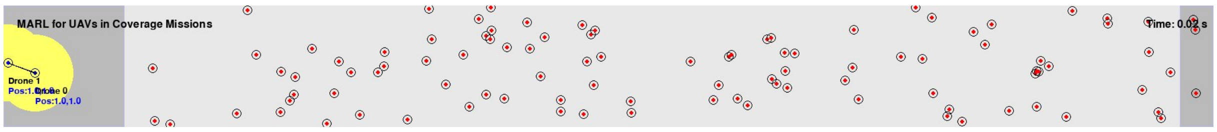


FIGURE 4.4 – Initial positions of a network for Task 1.

Algorithm 4 displays the reward function tailored for Task 1. The penalty for staying on the lower or upper borders were added so that the network is incentivised to avoid the initial position and traverse the field by moving through the obstacles in the middle.

Algorithm 4 Task 1 – Reward Function

```
# Penalty for staying of the left section or on the upper/lower border
if position in (LEFT_BORDER, UPPER_BORDER, LOWER_BORDER) then
    reward ← $-1.0/N$
else
    reward ← 0
end if
```

4.4.2 Task 2 – Connectivity Maintenance

Once Task 1 is accomplished, we begin to train a set of 3 drones to move to target location while maintaining connectivity, i.e. the algebraic connectivity λ of the network greater than zero.

We use Transfer Learning so that the training starts on top of the previous network that has already learned to move to target location. All training following this one also take advantage of Transfer Learning to build on top of what has already been learned.

After Connectivity Maintenance is learned for a network of 3 drones, we increase the network until it reaches 10 drones. The number of training episodes for each number of agents in the network is shown on Table 4.4 below.

Algorithm 5 displays the reward function devised for Task 2 for the first 2 training sessions, with 3 and 4 agents, respectively. In theses sessions we teach the network to maintain connectivity while moving through the field.

In the reward given by Connectivity Maintenance, and others in the following reward functions, we divide the reward of 1.0 by the number of drones in the network. This is

TABLE 4.4 – Task 2 – Number of Training Episodes for each Training Session.

Number of Agents	Training Episodes
3 Agents	1,000
4 Agents	2,000
5 Agents	3,000
10 Agents	7,000

because the update step following the end of the episode considers the sum of returns over all agents, so to keep the return bounded at each timestep, we divide its value by N.

Algorithm 5 Task 2 – Reward Function for Network with 3 and 4 Agents

```
# Connectivity Maintenance
if network is connected then
    reward ← 1.0/$N$
end if
# Walking in border penalty
if position in (LEFT_BORDER, UPPER_BORDER, LOWER_BORDER) then
    reward ← reward − 1.0/$N$
end if
```

For sizes 5 and 10, we change the reward function to increase the value of algebraic connectivity, to keep this metric above the minimum threshold determined by each scenario. Algorithm 6 below presents the addition of a reward term for the maintenance of algebraic connectivity above the threshold.

Algorithm 6 Task 2 – Reward Function for Network with 5 and 10 Agents

```
# Connectivity Maintenance
if network is connected then
    reward ← 2.0/$N$
    if algebraic connectivity > THRESHOLD then
        reward ← reward + 1.0/$N$
    end if
end if
# Walking in border penalty
if position in (LEFT_BORDER, UPPER_BORDER, LOWER_BORDER) then
    reward ← reward − 1.0/$N$
end if
```

The thresholds for each scenario were determined by Ghedini *et al.* (2018) and are different for each of the 20 initial network positions.

4.4.3 Task 3 – Area Coverage

With the completion of Task 2, we continue to train the network of 10 drones with an additional reward for Area Coverage. We separate this task in 3 training sessions, with networks with incrementally larger sizes 10, 15 and 20. The number of training episodes for each number of agents in the network is presented in Table 4.5.

TABLE 4.5 – Task 3 – Number of Training Episodes for each Training Session.

Number of Agents	Training Episodes
10 Agents	7,000
15 Agents	10,000
20 Agents	10,000

For the first training session, the added a term for Area Coverage given by

$$\text{Area Coverage Reward} = \frac{\text{Area Coverage}}{\text{Total Coverage}} \tag{4.4}$$

which calculates the current area being observed by all 10 drones in the network, by the theoretical maximum that could be covered, $10 \cdot \pi \cdot R^2$.

For the last 2 training sessions, this ratio was replaced by each drone's relative position to the network's center of mass shifted and scaled by the Observable Radius

$$\text{Area Coverage Reward} = \frac{|\text{position} - CM| - R}{R}. \tag{4.5}$$

Also, the reward to Connectivity Maintenance was increased to 5.0 to match the bigger rewards from Area Coverage. This reward function is presented by Algorithm 7.

Algorithm 7 Task 3 – Reward Function for Network with 15 and 20 Agents

```
# Area Coverage
reward ← (|Agent's Position - Center of Mass| - R )/ R
# Connectivity Maintenance
if network is connected then
  reward ← reward + $5.0/N$
  if algebraic connectivity > THRESHOLD then
    reward ← reward + $1.0/N$
  end if
end if
# Walking in border penalty
if position in (LEFT_BORDER, UPPER_BORDER, LOWER_BORDER) then
  reward ← reward − $1.0/N$
end if
```

4.4.4 Task 4 – Robustness to Failures

To start training Task 4, we first implement attacks on the network, following the higher Betweenness Centrality metric explained in Section 4.2.3. For a network of size 20, the attack policy implemented removes 70% of the network, or 14 drones, by the end of each episode. Given that an episode lasts 15 seconds, it gives us one attack per second on the network.

First, we analyze how the current agent acts under attacks and if it already can maintain the network connected when suffering these perturbations. Then, we reformulate the reward function to add the Robustness to failure term.

At this stage, we test 2 reward functions, one just adding an extra term to address Robustness to Failure, shown in Algorithm 8, and another by also removing the Connectivity reward, to test if the similarity between tasks 3 and 4 can be used to simplify the reward function, presented in Algorithm 9.

In both functions, the robustness term is calculated by the Robustness Level of the network, i.e. the number of drones needed to be removed to disconnect divided by the total number of drones, minus a threshold of 0.1, which stands for the case of 2 drones to be removed, $2/20 = 0.1$. We give the network a positive reward only when 3 or more drones need to be removed to disconnect, all other scenarios receive zero or negative rewards due to the threshold.

Algorithm 8 Task 4 – Reward Function

```
# Robustness Controller
reward ← $reward + (robustness\ level - 0.1)/N$
# Area Coverage
reward ← (|Agent's Position - Center of Mass| - R )/ R
# Connectivity Maintenance
**if** network is connected **then**
    reward ← $reward + 5.0/N$
    **if** algebraic connectivity > THRESHOLD **then**
        reward ← $reward + 1.0/N$
    **end if**
**end if**
# Walking in border penalty
**if** position in (LEFT_BORDER, UPPER_BORDER, LOWER_BORDER) **then**
    reward ← $reward - 1.0/N$
**end if**
```

For this task, we trained the network of 20 drones on top of all other tasks, giving each reward function a training time of 15,000 episodes. Also, we trained the network in two different scenarios, with and without attacks, to check if turning on attacks during training time helps enhances results in evaluation later.

Algorithm 9 Task 4 – Simplified Reward Function

```
# Robustness Controller
reward ← $reward + (robustness\ level - 0.1)/N$
# Area Coverage
reward ← (|Agent's Position - Center of Mass| - R )/ R
# Walking in border penalty
if position in (LEFT_BORDER, UPPER_BORDER, LOWER_BORDER) then
    reward ← $reward - 1.0/N$
end if
```

4.5 Evaluation Metrics

To evaluate the model when learning each of the task, besides the visual aid of the simulator, we use 3 main metrics: Algebraic Connectivity λ, Robustness Level $\Theta(\mathcal{G})$, presented in Sections 2.1.1 and 2.1.3, and Area Coverage Percentage, which is the ratio of the current covered area given the current topology by the total possible area to cover, shown in (4.4).

To summarize the evaluation of the model over all 200 scenarios, we calculate the Mean Algebraic Connectivity $\overline{\lambda}$, Mean Robustness Level $\overline{\Theta}(\mathcal{G})$, and Mean Area Coverage Percentage, throughout an episode. These metrics show how each behaves throughout an episode on average for all 200 possible scenarios.

Also, for each training session we present the Accumulated Reward (AR) plot, which consists of all the cumulative sums of rewards generated throughout training. The accumulated reward plotted for an episode is the resulting average from all 8 parallel environments.

5 Results and Discussion

This chapter presents the results of training in each task using Centralized Training with Decentralized Execution, including the evolution of accumulated reward per episode, as well as the evaluation based on the metrics of Algebraic Connectivity, Area Coverage and Robustness Level of the network. The chapter also contains detailed descriptions of the behavior learned in each task and special characteristics observed of the network[1].

5.1 Task 1 – Move to Target Location

In this first task, we trained a set of 2 agents with a simple reward function with a learning rate $\alpha = 3 * 10^{-4}/2 = 1.5 * 10^{-4}$, shown in Algorithm 4, that penalizes the agent if it stays at the left border of the field, which it would normally be by action of the potential field generated by all the obstacles to the right.

This reward function worked well for our purposes and generated the following Accumulated Reward (AR) plot shown in Figure 5.1. The plot is clipped after 1000 episodes to better visualization and presents the accumulated reward at each episode alongside an rolling moving average of 25 episodes, AR^{MA}, of the Accumulated Reward.

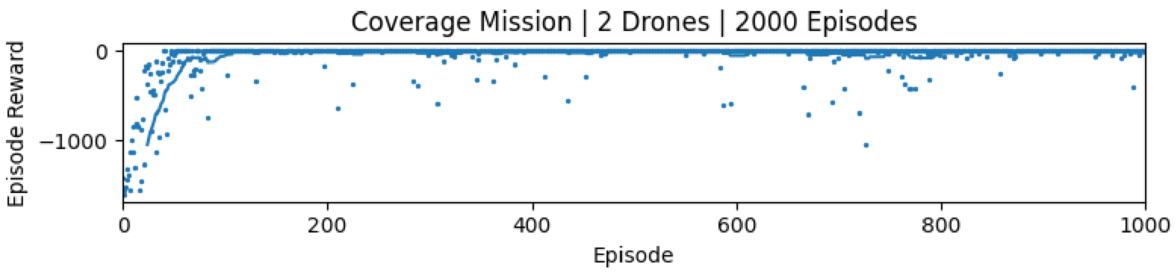


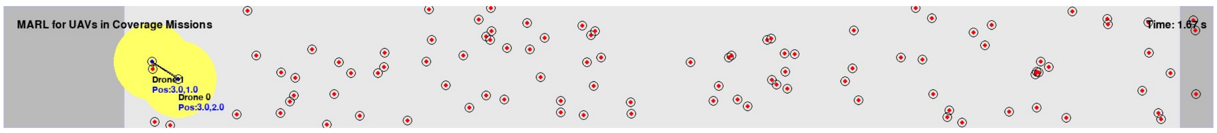
FIGURE 5.1 – Task 1 – Accumulated Reward per Episode, AR, for the first 1000 episodes (scatter plot). Accumulated Reward Moving Average of 25 episodes, AR^{MA} (blue line).

The network received updates based on the experience generated by 2 agents, and

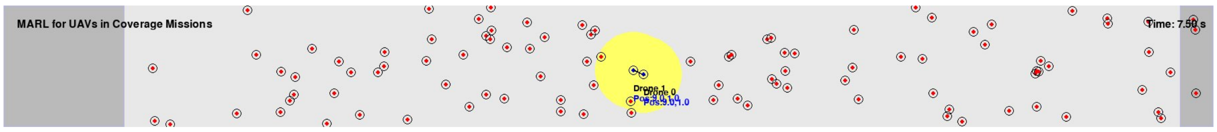
[1]A YouTube's playlist was created, in which we stored videos recording the learned behaviors for each task: https://www.youtube.com/playlist?list=PLxrRmCt2ARrhGBiw67ywoDEXkT2DznoLm

converged after about 120 episodes. Since it is quite a simple task, to learn to move right, it was expected that it would converge quickly to a solution.

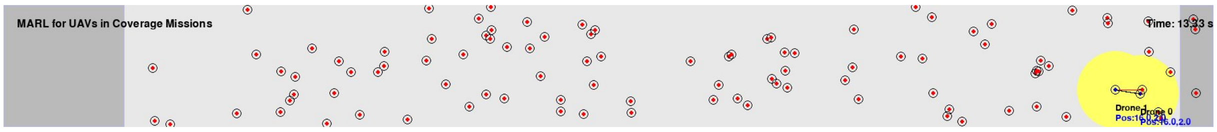
Figure 5.2 presents 3 snapshots at timesteps $t_1 = 100, t_2 = 450$ and $t_3 = 800$ of the evaluation of these agents after 2000 episodes of training. Note that the agents learned to move away from the initial position and towards the right end[2].



(a) Network of 2 drones at $t_1 = 100$.



(b) Network of 2 drones at $t_2 = 450$.



(c) Network of 2 drones at $t_3 = 800$.

FIGURE 5.2 – Task 1 – Behavior learned for a set of 2 drones.

Although the agent learned to move to target location, it is noticeable that it is still very much guided by the potential field generated by nearby obstacles and the other drone, and does not present any complex behavior like trying to avoid the obstacles before their potential field takes effect. Also, since we did not added any reward for connectivity, the agents performed their actions individually, not trying to keep close to each other or acting as a network.

5.2 Task 2 – Connectivity Maintenance

In this section we present the results for Connectivity Maintenance, separated by the size of the network during training and what behaviors it learned.

5.2.1 Training with 3 agents

To begin the training of Task 2, we took advantage of Transfer Learning to continue the training of the agent created in Task 1, this way we could build on top of the behavior previously learned to move to target location to compose a 3-drone network that also addresses Connectivity Maintenance.

[2]Link for YouTube's video of this behavior: https://www.youtube.com/watch?v=YlQ7jOcb7MI

In this scenario, a new reward was introduced, showed in Algorithm 5, in which the network received +1.0 every timestep it was connected. This creates an incentive for it to stay connected.

Figure 5.3 presents the AR per episode for this task after training for 1000 episodes with a learning rate $\alpha = 3 * 10^{-4}/3 = 1.0 * 10^{-4}$. By increasing the number of agents in the system, it is possible to observe the equal increase in variance in the early stages of training, which delays the convergence of the model, been reached after 250 episodes.

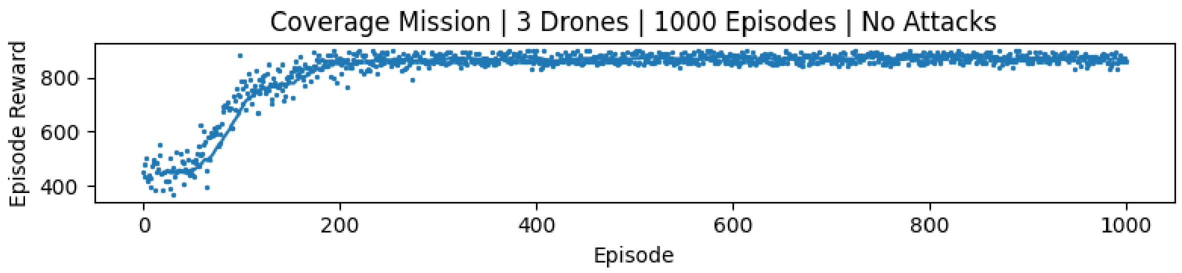
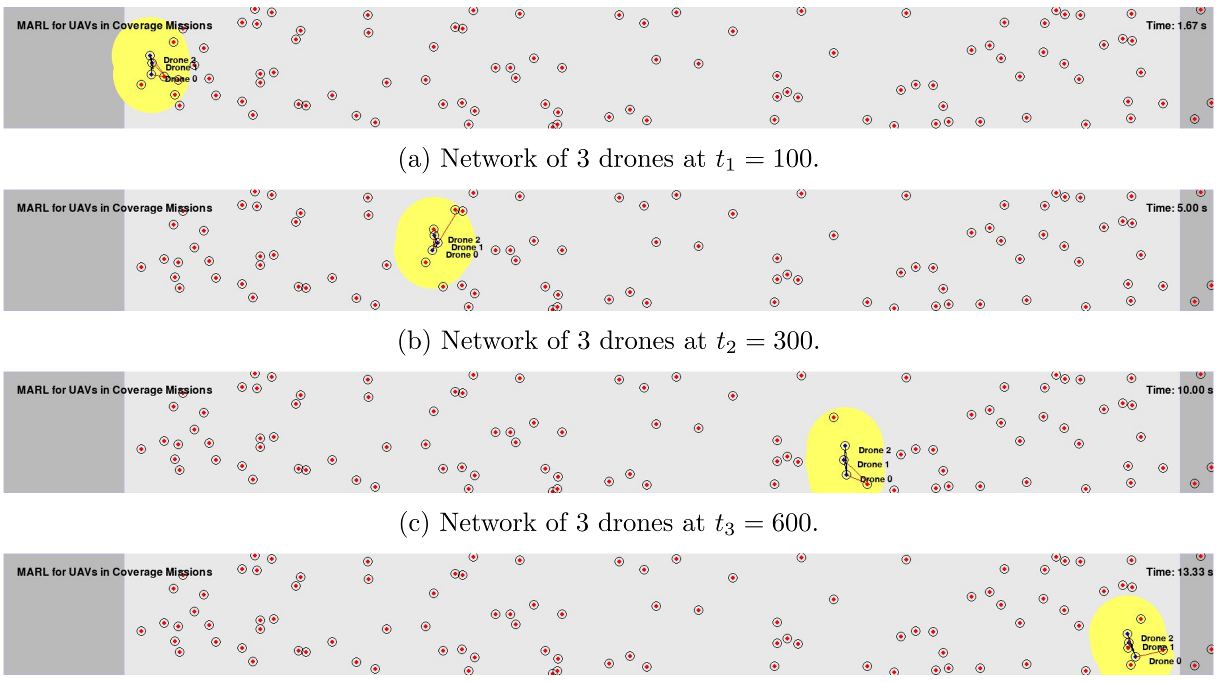


FIGURE 5.3 – Task 2 – Accumulated Reward per Episode, AR, for a network of 3 agents (scatter plot). Accumulated Reward Moving Average of 25 episodes, AR^{MA} (blue line).

By analyzing the network behavior, Figure 5.4, notice that the agents learned to keep high proximity between one another, increasing the chances of maintaining connectivity even when avoiding obstacles[3].



(a) Network of 3 drones at $t_1 = 100$.

(b) Network of 3 drones at $t_2 = 300$.

(c) Network of 3 drones at $t_3 = 600$.

(d) Network of 3 drones at $t_4 = 800$.

FIGURE 5.4 – Task 2 – Behavior learned for a network of 3 drones.

[3]Link for YouTube's video of this behavior: https://www.youtube.com/watch?v=EXRNTNIGZ2U

But, same as the previous task, the movement of the network is still mostly guided by the potential field, making them avoid obstacles and other drones without any complex decisions. By checking the output of the neural network, we perceive an output that makes the agent always follow right, with a slight bias to the center of the network, creating this strategy of close proximity.

We believe that a more simple strategy was learned because there is not a lot of complex scenarios to learn from. In later stages of training, with more drones in the network, we begin to see this characteristic change.

To evaluate how well this network performs the task of Connectivity Maintenance, we calculated the Mean Algebraic Connectivity $\overline{\lambda}$ throughout an episode over all 200 scenarios, presented in Figure 5.5.

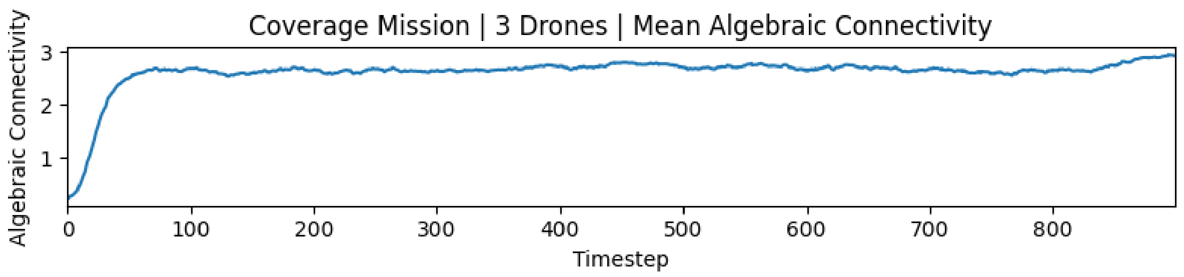


FIGURE 5.5 – Task 2 – Mean Algebraic Connectivity $\overline{\lambda}$ throughout an episode for a network of 3 drones over 200 scenarios.

Analyzing $\overline{\lambda}$ we perceive a significant increase in its value in the initial time steps, time in which the agents start at random positions and soon rearrange themselves to form a more connected topology. The mean algebraic connectivity $\overline{\lambda}$ remains stable for most of the episode, having little to no disruptions. At any point we see $\overline{\lambda}$ reach zero, showing that the network learned Connectivity Maintenance for a network of size 3.

This result was actually expected, since it is quite a simple network, and all agents are close to each other, thus under almost the same resulting potential field from the environment.

5.2.2 Training with 4 agents

The training for network of 4 drones was done with the same reward function as previous training session, Algorithm 5, and it trained for 2000 episodes with a learning rate $\alpha = 3 * 10^{-4}/4 = 0.75 * 10^{-4}$. Figure 5.6 presents the AR per episode for this session. The highlighted line represents the 25-Moving Average of the AR.

In this training session the objective was to analyze if the increase in number of agents affects the training time, overall performance, and if the behavior learned previously is

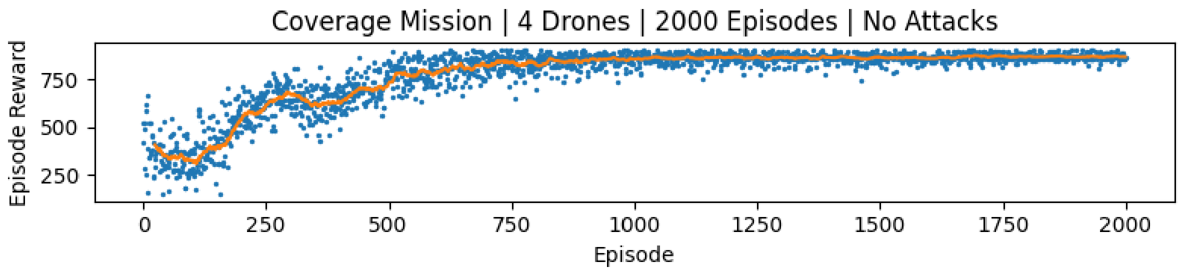
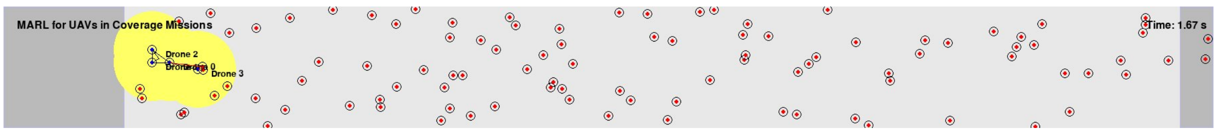


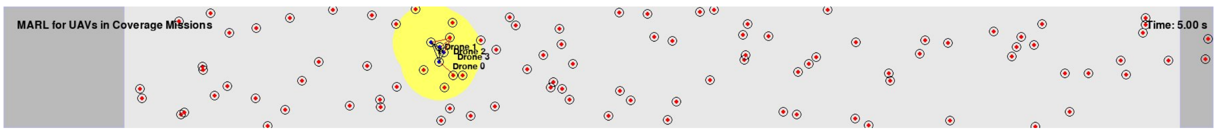
FIGURE 5.6 – Task 2 – Accumulated Reward per Episode, AR, for a network of 4 agents (scatter plot). Accumulated Reward Moving Average of 25 episodes, AR^{MA} (orange line).

kept. Observing the snapshots taken in Figure 5.7, notice that the network continues to move right without complications. In a few occasions, to avoid obstacles, one or two drones distance themselves from the network, but they quickly return to their original position once the obstacle is surpassed[4].

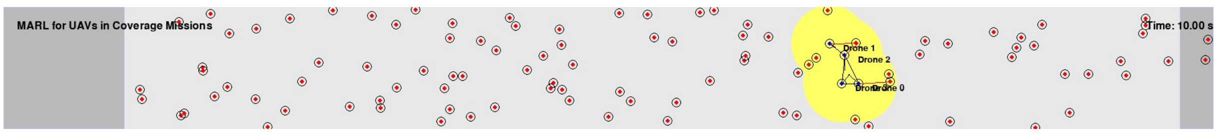
Analyzing the AR plot, Figure 5.6, the training of the network took more time to converge this time due to the increase in variance the addition of a drone causes. In this session, convergence happened only by the mark of 750 episodes. But also, is possible to affirm that the network already starts training with a relatively good performance, with rewards in the range of 400-500 points, showing that Transfer Learning is being used successfully.



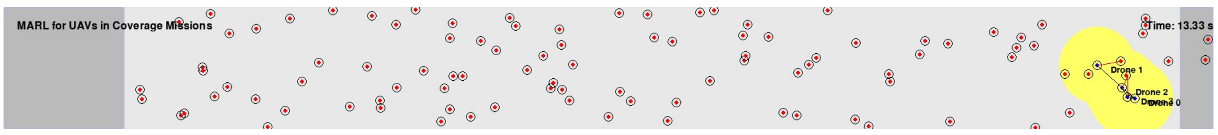
(a) Network of 4 drones at $t_1 = 100$.



(b) Network of 4 drones at $t_2 = 300$.



(c) Network of 4 drones at $t_3 = 600$.



(d) Network of 4 drones at $t_4 = 800$.

FIGURE 5.7 – Task 2 – Behavior learned for a network of 4 drones.

[4]Link for YouTube's video of this behavior: https://www.youtube.com/watch?v=1PZGYOuUgM8

Figure 5.8 presents the network's Mean Algebraic Connectivity $\overline{\lambda}$ throughout an episode over 200 benchmark scenarios. Although the stable behavior remains, its value diminishes throughout the episode compared to the previous network, only increasing again at the end of the episode.

This later surge in value can be explained by verifying that all drones reached target location by the end of the episode, and at this position they rearrange themselves in a better topology now that are free from the presence of obstacles.

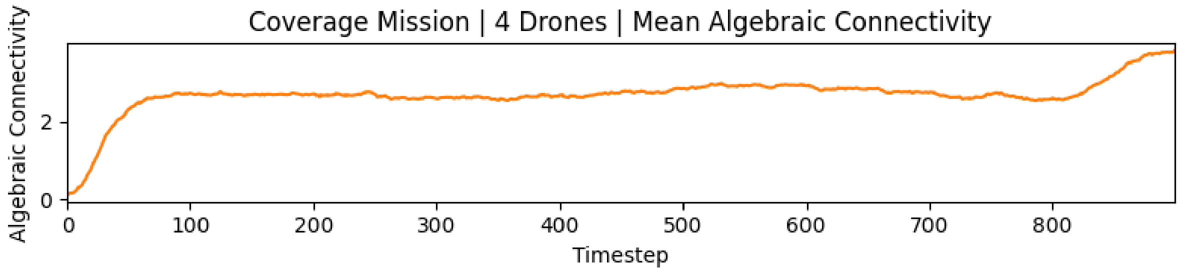


FIGURE 5.8 – Task 2 – Mean Algebraic Connectivity $\overline{\lambda}$ throughout an episode for a network of 4 drones over 200 scenarios.

5.2.3 Training with 5 agents

For the training of 5 agents we change the reward function so that there is an extra reward if the network manages to keep its algebraic connectivity above a certain threshold, as shown in Algorithm 6. We trained for 3,000 episodes with an $\alpha = 3{*}10^{-4}/5 = 0.6{*}10^{-4}$.

Figure 5.9 presents the Accumulated Reward for this training session. In it, we can see that the AR reached the mark of 2700, due to the increase of the connectivity reward and the extra reward for being above the threshold, but there is no apparent change in the network's behavior by analyzing Figure 5.10[5].

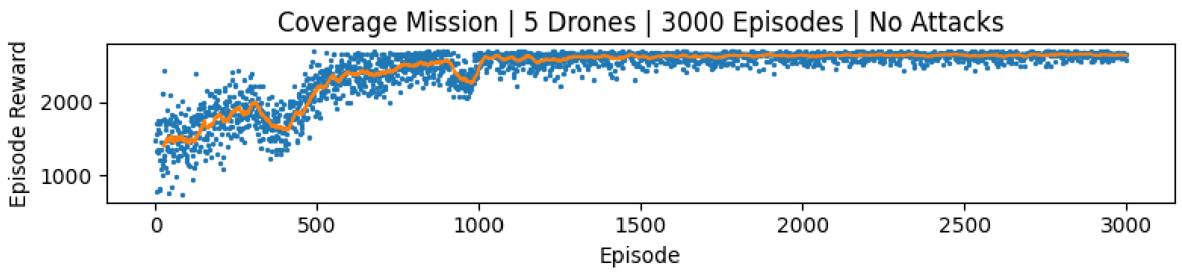
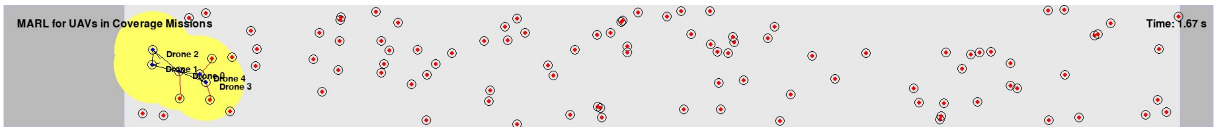


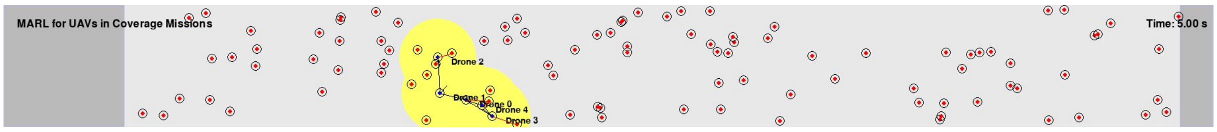
FIGURE 5.9 – Task 2 – Accumulated Reward per Episode, AR, for a network of 5 agents (scatter plot). Accumulated Reward Moving Average of 25 episodes, AR^{MA} (orange line).

[5]Link for YouTube's video of this behavior: https://www.youtube.com/watch?v=UrwlK4xIiTc

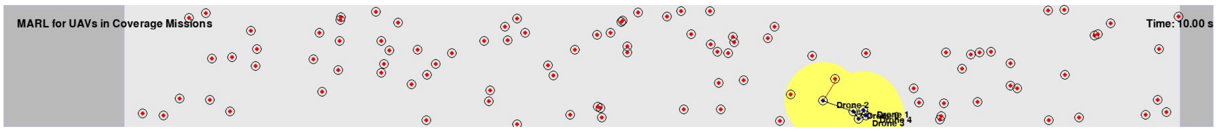
A reason for this may be the small number of drones in the network, given that the thresholds were defined for a network of size 20 under attack, only 5 drones without disturbances may be generating a λ naturally larger than the threshold, given that there is not much complexity yet in the network.



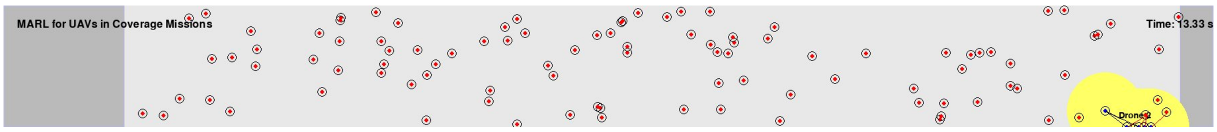
(a) Network of 5 drones at $t_1 = 100$.



(b) Network of 5 drones at $t_2 = 300$.



(c) Network of 5 drones at $t_3 = 600$.



(d) Network of 5 drones at $t_4 = 800$.

FIGURE 5.10 – Task 2 – Behavior learned for a network of 5 drones.

Figure 5.11 shows the Mean Algebraic Connectivity $\overline{\lambda}$ throughout an episode for this network over 200 scenarios. We can see that this is definitely the case, $\overline{\lambda}$ is presented to be above 2 at almost all times as were previous networks, way above the thresholds defined for a network of 20 agents, that are real values smaller than 1.

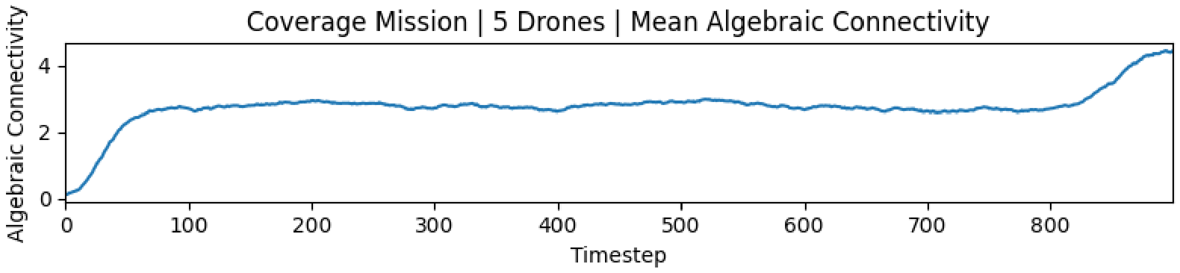
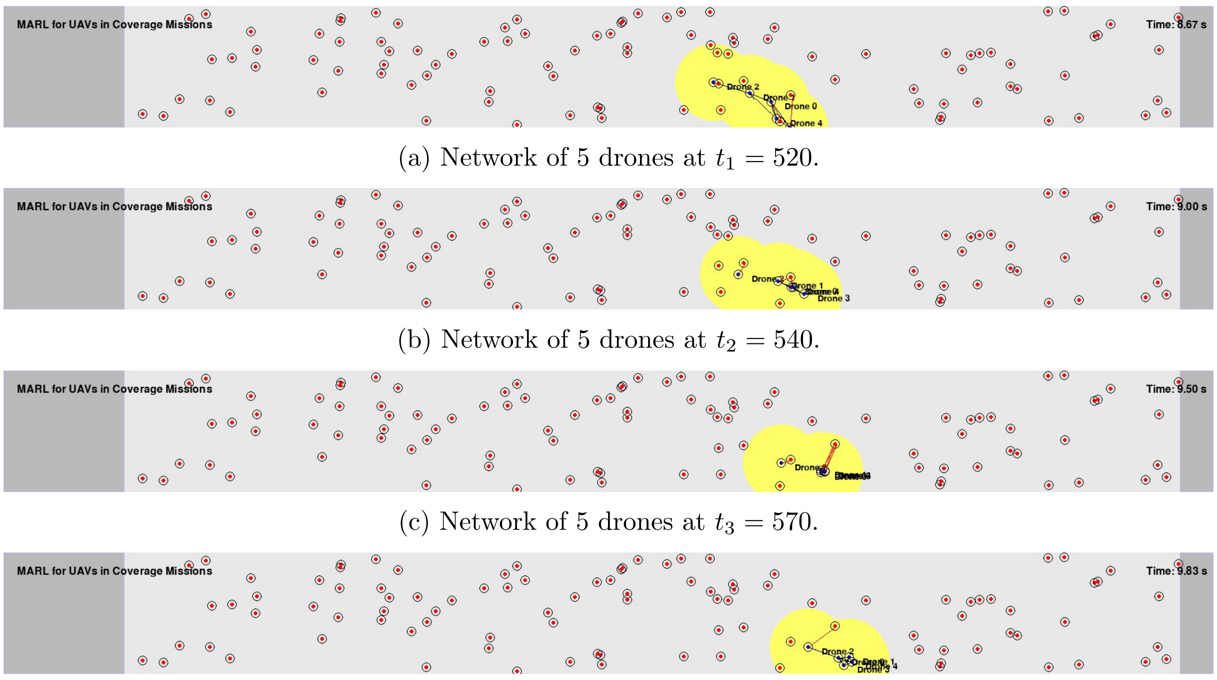


FIGURE 5.11 – Task 2 – Mean Algebraic Connectivity $\overline{\lambda}$ throughout an episode for a network of 5 drones over 200 scenarios.

Another result emerged in this training session is an interesting side-effect of increasing the reward on connectivity. Sometimes, due to the positioning of obstacles, a drone can get stuck and disconnect from the rest of the network, Figures 5.12a and 5.12b present this scenario, interesting enough the network learned to slow down its movement and regroup so that the disconnected drone can rejoin the network, Figures 5.12c and 5.12d.



(a) Network of 5 drones at $t_1 = 520$.

(b) Network of 5 drones at $t_2 = 540$.

(c) Network of 5 drones at $t_3 = 570$.

(d) Network of 5 drones at $t_4 = 590$.

FIGURE 5.12 – Task 2 – Regroup behavior learned for a network of 5 drones.

Although this situation is rare, mostly because the network learned to perform well the Connectivity Maintenance task for a network of 5 agents, we ran a few tests on which this situation is presented. We measured the output of the RL agent to check what actions are being executed when the network is disconnected. Once a state of disconnection is set, the agent's output starts to take into account what it seems to be the neighbors potential field, so it still has a component that leads the agent to target location, but another that makes the network as a whole regroup and slow down.

For the disconnected drone, since the potential field is zero, it has no knowledge of where their neighbors are in the field, so its actions remain the same, to move to the target location.

This analysis shows that a major issue rises then if the disconnected neighbor happens to be in any other direction from the network but behind. It seems that if a drone is left behind, it will always rejoin the network, but if it is on top, or in front of the network, it will never reach the group again, and keep moving forward, while the network will focus in regroup and slow down the execution of the mission.

5.2.4 Training with 10 agents

Given that the network with 5 drones did not extracted any new knowledge from the insertion of the threshold in the reward function, instead of changing the function again we chose to increase the number of drones to 10 in the attempt of achieve a more complex behavior.

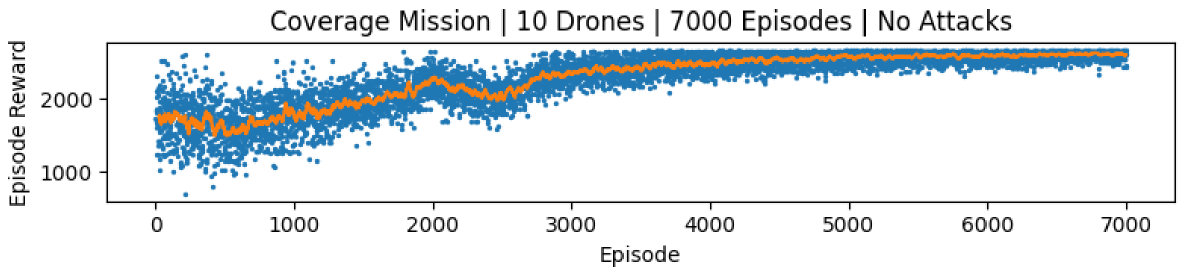


FIGURE 5.13 – Task 2 – Accumulated Reward per Episode, AR, for a network of 10 agents (scatter plot). Accumulated Reward Moving Average of 25 episodes, AR^{MA} (orange line).

Figure 5.13 presents the Accumulated Rewards per episode during training for 7000 episodes, with an $\alpha = 3{*}10^{-4}/10 = 3{*}10^{-5}$. The first thing to notice is a great increase in variance during training, which leads to convergence after almost 5000 episodes. Second, the AR also reaches the mark of 2700 points, showing that the network is being able to maintain the Algebraic Connectivity above the threshold for a certain period of time, thus receiving the due reward for this, but not because it learned to do so, it just happens to naturally maintain a tight configuration from start, leading to a high $\overline{\lambda}$ as well.

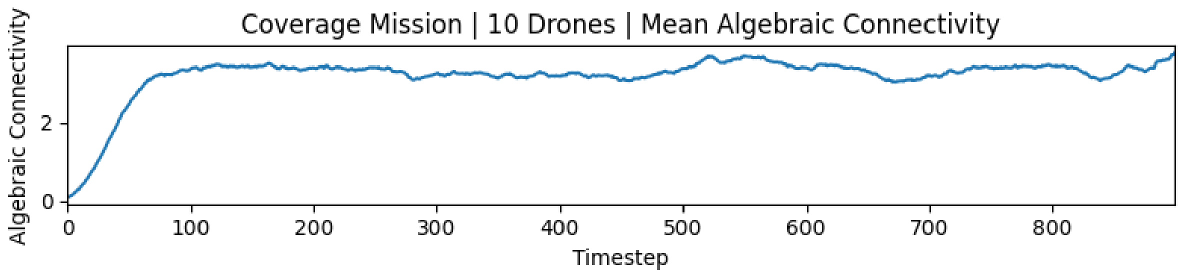
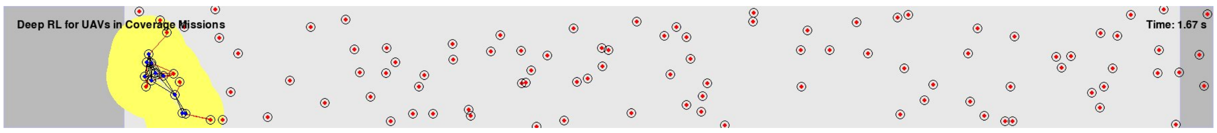


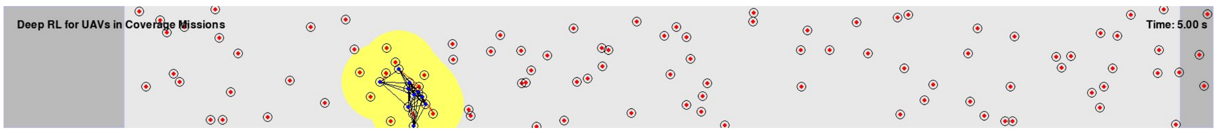
FIGURE 5.14 – Task 2 – Mean Algebraic Connectivity $\overline{\lambda}$ throughout an episode for a network of 10 drones.

Figure 5.14 shows the mean algebraic connectivity $\overline{\lambda}$ throughout an episode for this network over 200 scenarios.

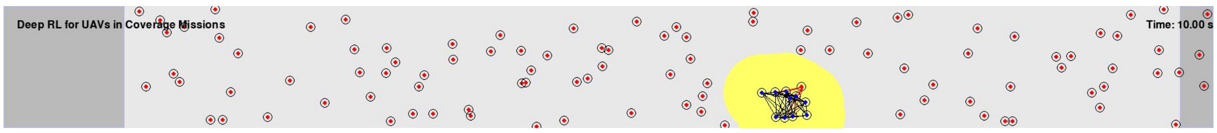
Analyzing Figure 5.15, we can observe how the network maintains and improves the behavior of close proximity between drones, leading to a high value of λ. This behavior became widely observed in almost all scenarios. After training Connectivity Maintenance for 13000 episodes throughout all network sizes, the agents learned that keeping a tight topology helps them avoid disconnecting. One potential reason for this is that being all close to each other all of them suffers almost the same resulting potential field from the environment, so the network does not need to learn more complex behaviors[6].



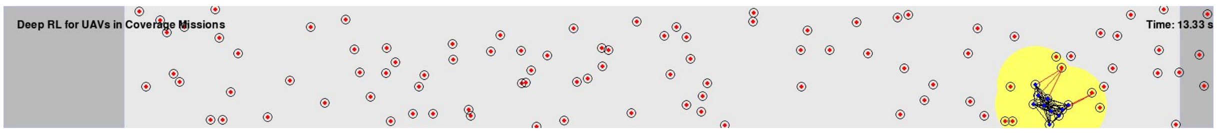
(a) Network of 10 drones at $t_1 = 100$.



(b) Network of 10 drones at $t_2 = 300$.



(c) Network of 10 drones at $t_3 = 600$.



(d) Network of 10 drones at $t_4 = 800$.

FIGURE 5.15 – Task 2 – Behavior learned for a network of 10 drones.

Another behavior that became clear in this training session is network's bias of always traversing the field more closely to the lower border, instead of trying to traverse through the center. This behavior is observed until the end of this work.

By presenting these results, we consider that the agents have successfully learned to maintain a network topology that performs Connectivity Maintenance while moving to target and avoiding obstacles.

5.3 Task 3 – Area Coverage

In this section we present the results for Area Coverage, separated by the size of the network during training and what behaviors it learned.

Building on top of the behaviors learned of Moving to Target Location and Connectivity Maintenance, we now focus of training the network for expanding its observable area, a fundamental task for Coverage Missions.

[6]Link for YouTube's video of this behavior: https://www.youtube.com/watch?v=yD98inuUeMo

5.3.1 Training with 10 agents

To train the network to expand its topology, a few ideas were implemented. First, we tried to feed the network a reward for expanding its observable area based on the total area currently covered divided by the total possible area the drones can cover.

Unfortunately, this term did not help the network to learn how to expand its domain. Figure 5.16 shows how the accumulated reward did not increased much from its previous value, having just been offset by a constant value, with a little variance at the beginning.

At the evaluation step we also did not perceive any major change in behavior, so the evaluation snapshots are omitted here.

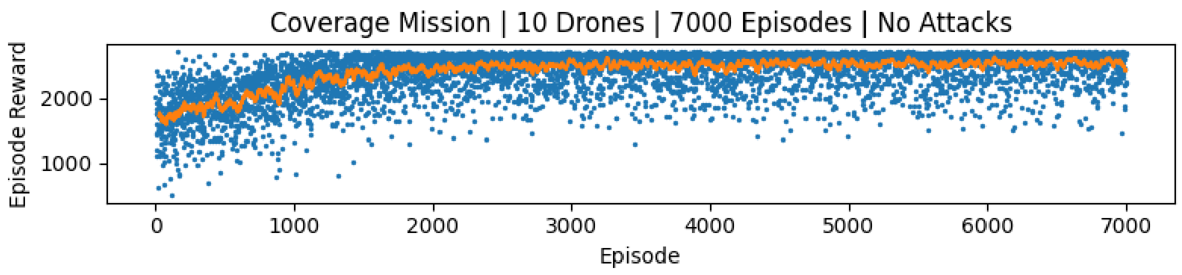


FIGURE 5.16 – Task 3 – Accumulated Reward per Episode, AR, for a network of 10 agents (scatter plot). Accumulated Reward Moving Average of 25 episodes, AR^{MA} (orange line).

This issue may have risen for a number of reasons. First, that the ratio being always positive, the reward gives little to no information to the agent if its actions benefits or not the Area Coverage. Second, drones on the border of the network contributes way more to this reward than inner drones, causing these to act within the network with no regards to Area Coverage, leaving this task to the agents on the border.

5.3.2 Training with 15 agents

To address the problem found in the network with 10 drones, the reward function was reformulated, presented in Algorithm 7, in which we reward drones that are more distant from the center of mass of the network. In this function, the Observable Radius is subtracted from the calculated relative position to also penalize those inner agents that are concentrated on the center of mass.

Figure 5.17 shows us how that training converged, but with the high variance included we cannot really confirm that the network is properly learning the task. Observing the behavior of the network in Figure 5.18 we still notice a large concentration of the drones around a single point, showing the predominance of Connectivity Maintenance[7].

[7]Link for YouTube's video of this behavior: https://www.youtube.com/watch?v=hlOaMlbRM4g

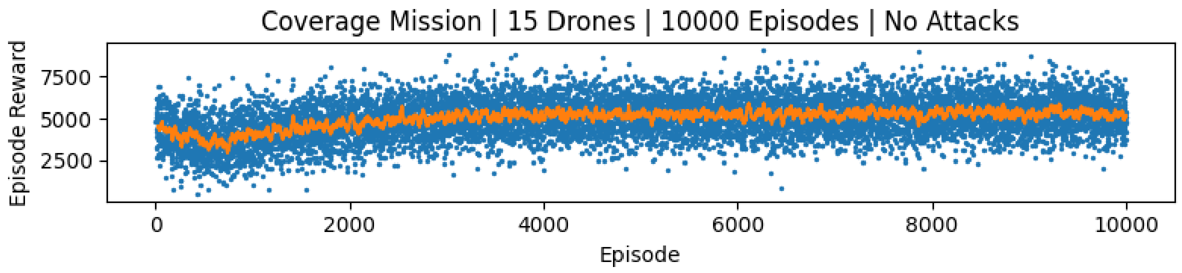
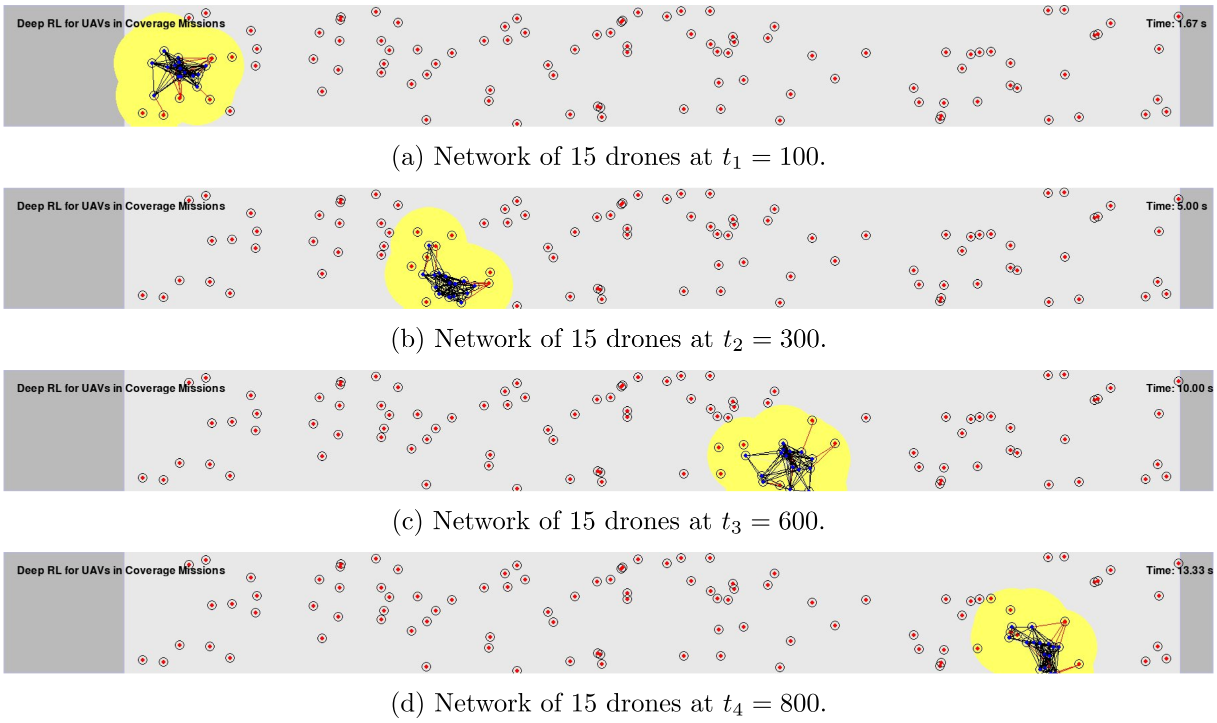


FIGURE 5.17 – Task 3 – Accumulated Reward per Episode, AR, for a network of 15 agents (scatter plot). Accumulated Reward Moving Average of 25 episodes, AR^{MA} (orange line).

At this point a few variations of this reward function were experimented, but none of them resulted in a better result, so we decided to continue training with the reward function defined in Algorithm 7.



(a) Network of 15 drones at $t_1 = 100$.

(b) Network of 15 drones at $t_2 = 300$.

(c) Network of 15 drones at $t_3 = 600$.

(d) Network of 15 drones at $t_4 = 800$.

FIGURE 5.18 – Task 3 – Behavior learned for a network of 15 drones.

We exhibit the Mean Algebraic Connectivity and Area Coverage plots for this network in Figures 5.19 and 5.20. Comparing $\overline{\lambda}$ with the previous network, we see that it increased once again, while the percentage of area covered stayed around 0.07, confirming that the network is predominantly performing Connectivity Maintenance over Area Coverage.

After failing to improve the performance of Area Coverage with a network of 15 agents, we decided to increase the network size to 20, to evaluate if a larger network by itself, that creates a greater potential field for the large number of neighbors, can naturally generate better results when performing this task.

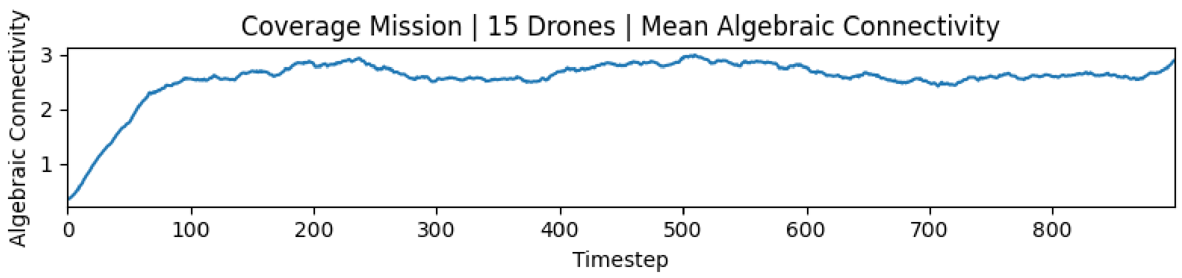


FIGURE 5.19 – Task 3 – Mean Algebraic Connectivity $\overline{\lambda}$ throughout an episode for a network of 15 drones.

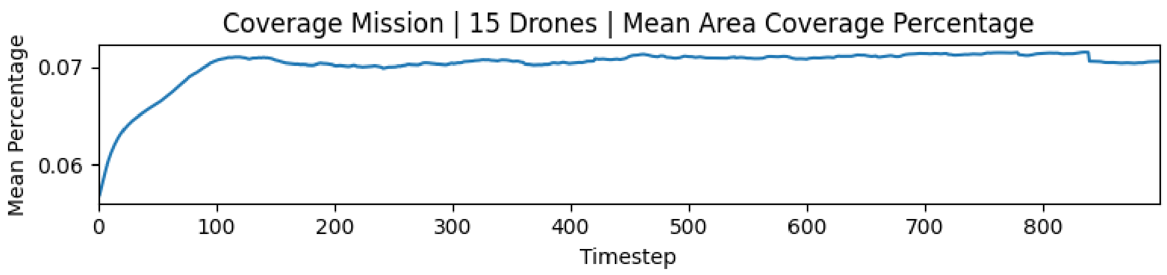


FIGURE 5.20 – Task 3 – Mean Area Coverage Percentage throughout an episode for a network of 15 drones.

5.3.3 Training with 20 agents

At this final training session, we trained a network of 20 agents for 10,000 episodes with the reward function given by Algorithm 7, and an $\alpha = 1.5*10^{-5}$. Figure 5.21 presents the resulting AR per episode, in it we can extract that after another 10,000 episodes of training the variance decreases in a way that the result seems more stable.

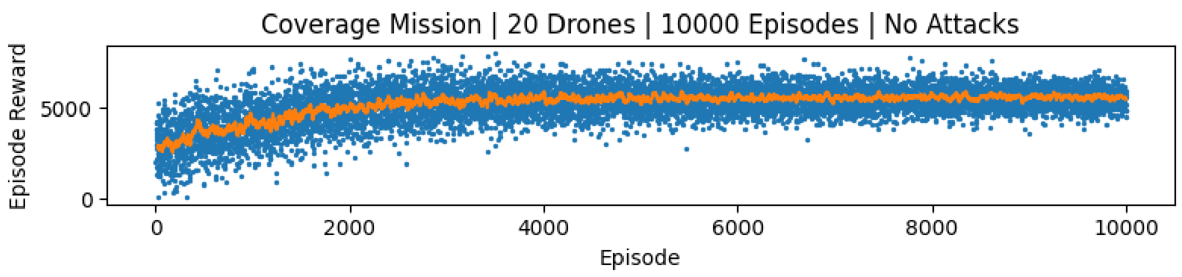


FIGURE 5.21 – Task 3 – Accumulated Reward per Episode, AR, for a network of 20 agents (scatter plot). Accumulated Reward Moving Average of 25 episodes, AR^{MA} (orange line).

For our surprise, when analyzing the $\overline{\lambda}$ of this network, its value dropped significantly, Figure 5.22. Now with a network of 20 drones, we revisited the comparison of Algebraic Connectivity and the thresholds defined to each scenario, and even so, the value of algebraic connectivity stays way above 1. This concludes that the network naturally learned to keep a high margin of safety for Connectivity Maintenance.

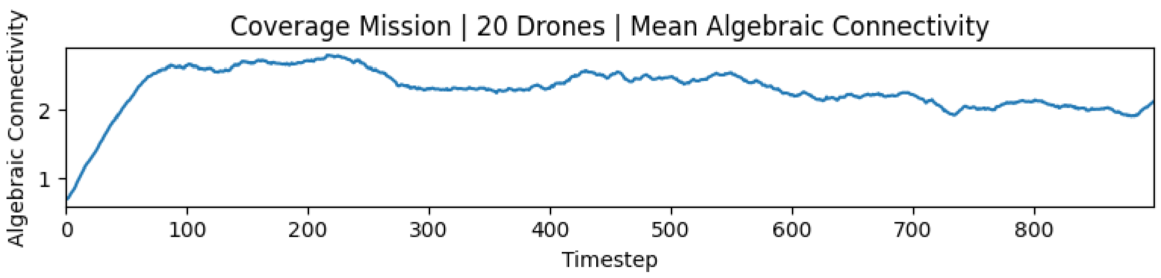
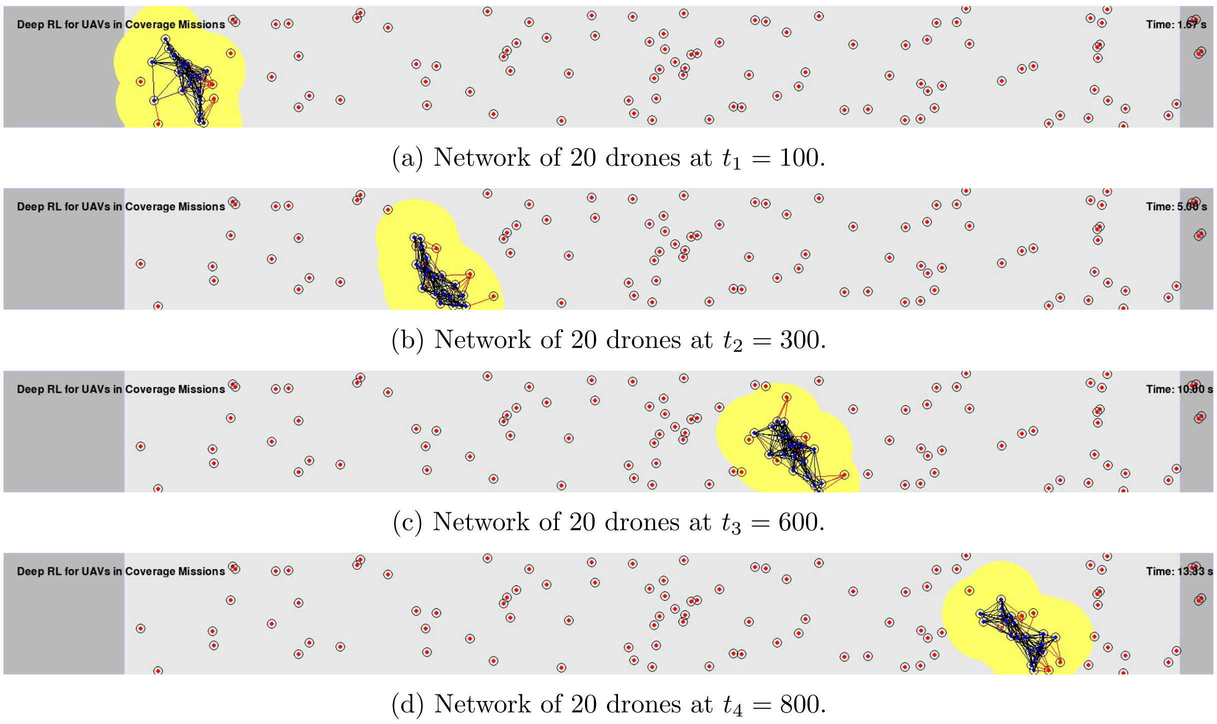


FIGURE 5.22 – Task 3 – Mean Algebraic Connectivity $\overline{\lambda}$ throughout an episode for a network of 20 drones.

Figure 5.23 presents snapshots of the network traversing the field and the behavior learned. And, confirming the result presented in Figure 5.22, the network seems more spread out, exchanging connectivity with area coverage[8].

This result may come because of the number of drones in the network, generating more potential field, causing a agent to distance itself more from its neighbors, or by the additional time of training with this particular reward. But, the result is still beneath the expectations for the task, once the network was expected to assume a topology similar to a Voronoi Tesselation, as was the network generated by Ghedini *et al.* (2018).

Figure 5.24 presents the Mean Area Coverage Percentage for a network of 20 drones. It value stayed stable, near to 6% or 7%.



(a) Network of 20 drones at $t_1 = 100$.

(b) Network of 20 drones at $t_2 = 300$.

(c) Network of 20 drones at $t_3 = 600$.

(d) Network of 20 drones at $t_4 = 800$.

FIGURE 5.23 – Task 3 – Behavior learned for a network of 20 drones.

[8]Link for YouTube's video of this behavior: https://www.youtube.com/watch?v=jpb9PPhz3BY

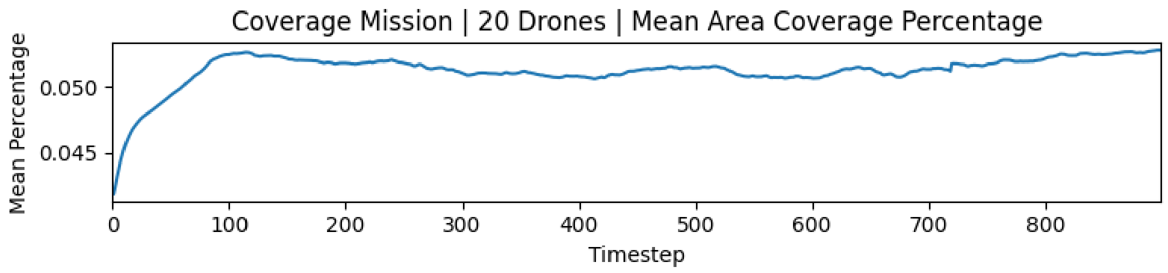


FIGURE 5.24 – Task 3 – Mean Area Coverage Percentage throughout an episode for a network of 20 drones.

With these results, we conclude that the network learned a sub-optimal solution for Area Coverage, performing the task but way below our expectations. The agents now can perform 3 out of the 4 tasks proposed: Move to Target Location, Connectivity Maintenance and Area Coverage, although this last one not entirely correct. In this last section we show the results of training the network for its final task of Robustness to Failures.

5.4 Task 4 – Robustness to Failures

In this section we present the results for Robustness to Failures, starting directly with a network of 20 drones. Building on top of the behaviors learned of Move to Target Location, Connectivity Maintenance and Area Coverage, we now focus of training the network for establishing a topology that can carry on the mission even under attack.

5.4.1 Enabling Failures and Measuring Current Performance

First, we enable the attack policy in the network, following the policy defined in Section 4.2.3, and measure the current performance of the network as is in terms of Mean Robustness Level $\overline{\Theta}(\mathcal{G})$ during an episode. This metric is calculated over 200 scenarios, and presented in Figure 5.25.

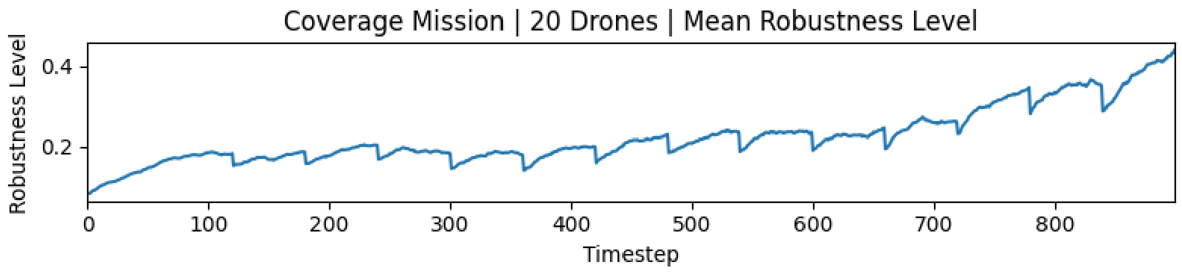


FIGURE 5.25 – Task 4 – Mean Robustness Level $\overline{\Theta}(\mathcal{G})$ throughout an episode for a network of 20 drones.

The rhythmic behavior of $\overline{\Theta}(\mathcal{G})$ shows the attacks occurring in the network. Each second a drone in the network is attacked and removed, diminishing the network's connectivity and robustness.

We can observe that given the connected behavior learned in the previous training sessions, the network keeps $\Theta(\mathcal{G})$ above 0.1 at all times, meaning that on average it would take 2 simultaneous attacks to disconnect the network. By the end of the episode, the network increases $\Theta(\mathcal{G})$ due to a smaller number of agents and a simpler topology. Also, we can observe how after each attack the network rearranges itself to increase the robustness level again, trying to maintain a resilient topology.

5.4.2 Training with 20 agents

Using the reward function given by Algorithm 8, we trained the network for a total of 15,000 episodes and with an $\alpha = 1.5 * 10^{-5}$, this time enabling attacks during training. Figure 5.26 presents the AR per episode resulted by this task. The first thing to notice is how long the network took to converge, over 10,000 episodes. This can be explained by the decreasing number of drones in the network until the end of the episode caused by the attacks, generating less and less experience towards the end.

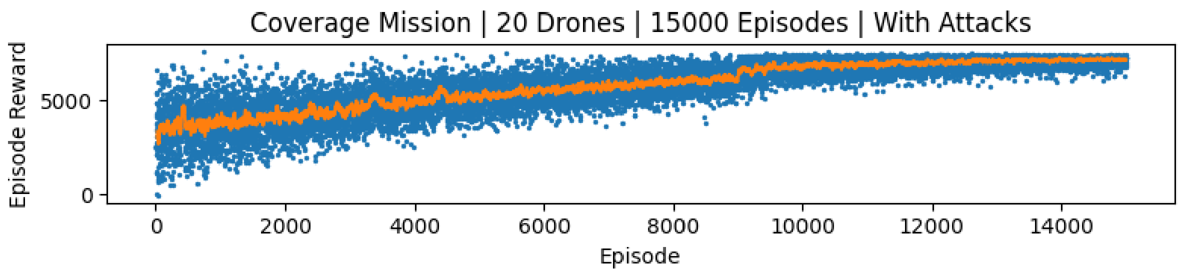


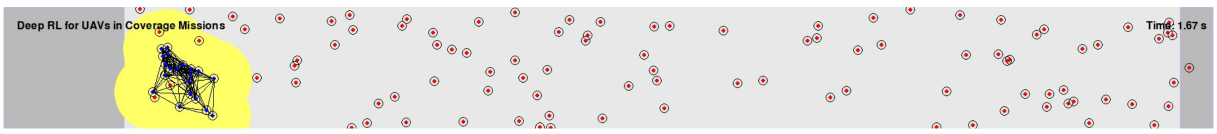
FIGURE 5.26 – Task 4 – Accumulated Reward per Episode, AR, for a network of 20 agents (scatter plot). Accumulated Reward Moving Average of 25 episodes, AR^{MA} (orange line).

Figure 5.27 presents the learned behavior[9]. On it, we can see the number of agents decreasing due to attacks on the drones of higher Betweenness Centrality. Also, note that the network maintains its members connected until the end, enduring the attacks.

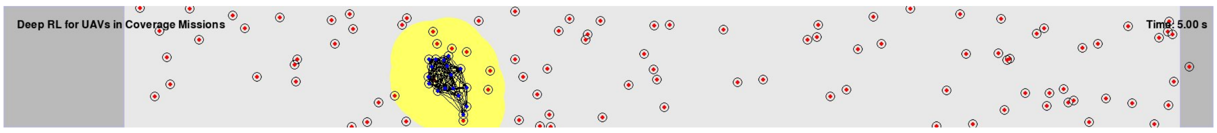
Evaluating different scenarios, the network showed consistent results in Connectivity Maintenance and Robustness to Failure by exchanging performance in Area Coverage.

To evaluate the Robustness Level of the network during an episode, we plot the Mean Robustness Level $\overline{\Theta}(\mathcal{G})$, Figure 5.28. Note that after the training the value of $\overline{\Theta}(\mathcal{G})$ had a slight increase in the beginning of episodes until 400 timesteps, now maintaining its value above 0.2 at almost all times. This suggests that the learned behavior is creating a resilient

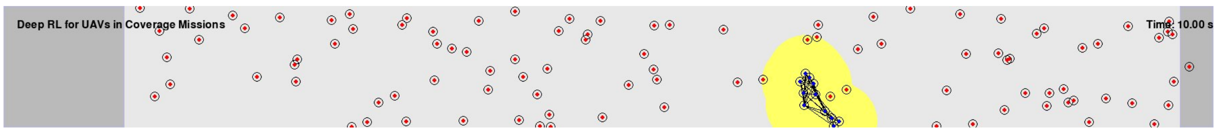
[9]Link for YouTube's video of this behavior: https://www.youtube.com/watch?v=Ga7SBLxuw04



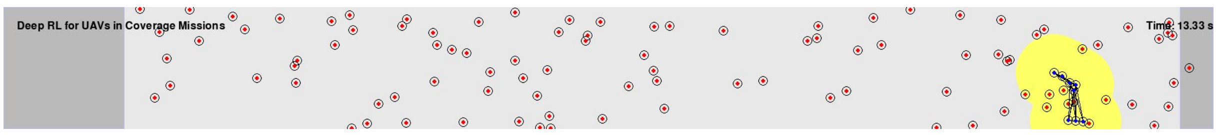
(a) Network of 20 drones at $t_1 = 100$, 19 drones remaining.



(b) Network of 20 drones at $t_2 = 300$, 15 drones remaining.



(c) Network of 20 drones at $t_3 = 600$, 10 drones remaining.



(d) Network of 20 drones at $t_4 = 800$, 7 drones remaining.

FIGURE 5.27 – Task 4 – Behavior learned for a network of 20 drones.

topology capable of enduring attacks. In addition, the rhythmic behavior continues to appear, showing that the network tries to recompose itself in a resilient manner every time after an attack.

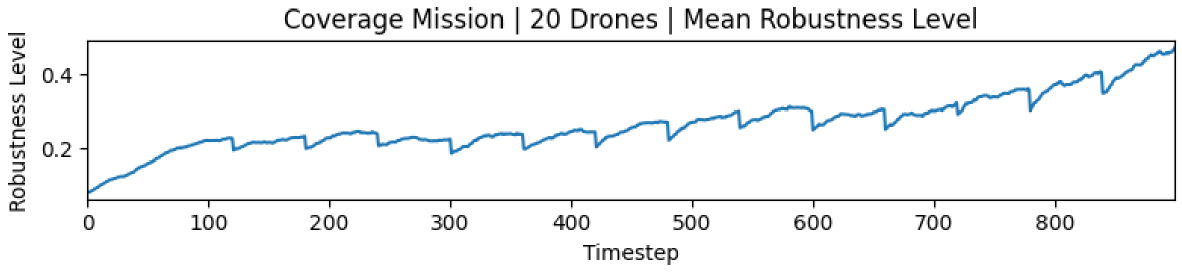


FIGURE 5.28 – Task 4 – Mean Robustness Level $\overline{\Theta}(\mathcal{G})$ throughout an episode for a network of 20 drones.

Also, to evaluate how the other tasks of Connectivity Maintenance and Area Coverage were affected, we plot the Mean Algebraic Connectivity $\overline{\lambda}$ and the Mean Area Coverage Percentage, Figures 5.29 and 5.30. This time, in comparison with Figure 5.22 that previously decreased the value of λ, the network under attack rearranges itself to keep a high value of λ. This may be caused by the similarity of the two tasks.

Interesting to see, although the thresholds devised by Ghedini *et al.* (2018) for algebraic connectivity were made for this exact scenario, 20 drones in a network under attack, and were close to 0.5, the value of $\overline{\lambda}$ seems to be never go below 2.0, showing that even though the network have space for a trade-off from Connectivity to Area Coverage, it chose to specialize in Connectivity Maintenance and Robustness to Failures. This suggests that there is still opportunity for continue learning Task 3 afterwards.

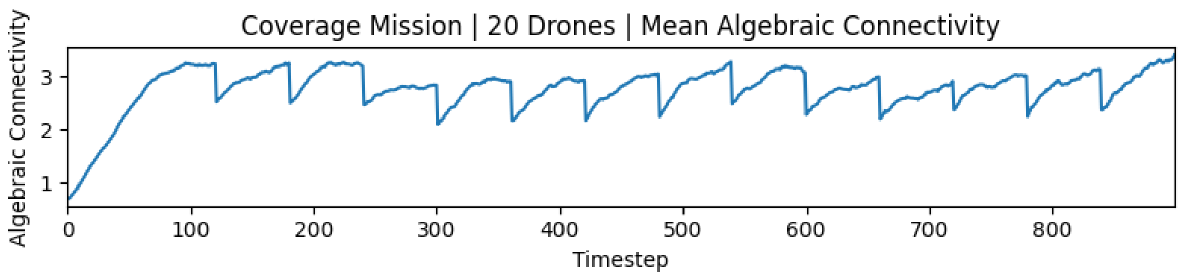


FIGURE 5.29 – Task 4 – Mean Algebraic Connectivity $\overline{\lambda}$ throughout an episode for a network of 20 drones.

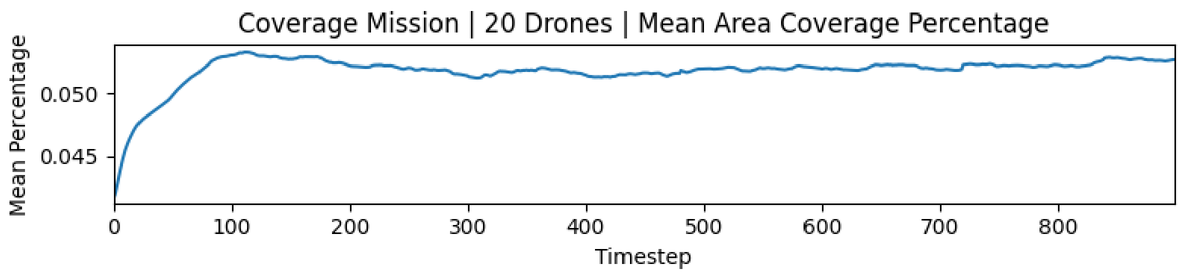


FIGURE 5.30 – Task 4 – Mean Area Coverage Percentage throughout an episode for a network of 20 drones.

5.4.3 Effect of the Connectivity Controller

After training the network with an additional reward to tackle robustness, we devised a second reward function, keeping the additional term but removing all rewards regarding connectivity of the network, Algorithm 9. This was done after realizing that the previous reward function just enhanced the behavior already learned for Task 3. The training occurred in the same standards as before, 15,000 episodes and on top of what the agent had already learned from the 3 previous tasks, but this time without the knowledge acquired from last training session for Task 4. We removed the last layer of training to be able to compare the 2 training sessions.

Figure 5.31 presents the Accumulated Reward per episode for this task. By the end of it, when comparing the behavior learned with the one previously trained with the additional connectivity reward, it is possible to say that no major change were detected. The same characteristics of maintaining a resilient topology throughout the episode and rearrangement after attacks were noticed in the retrained network as well.

This result suggests that when the Robustness to Failures is introduced in the model the Connectivity Maintenance starts to lose importance, since both have great similarity, as both try to maintain the drones as close as possible.

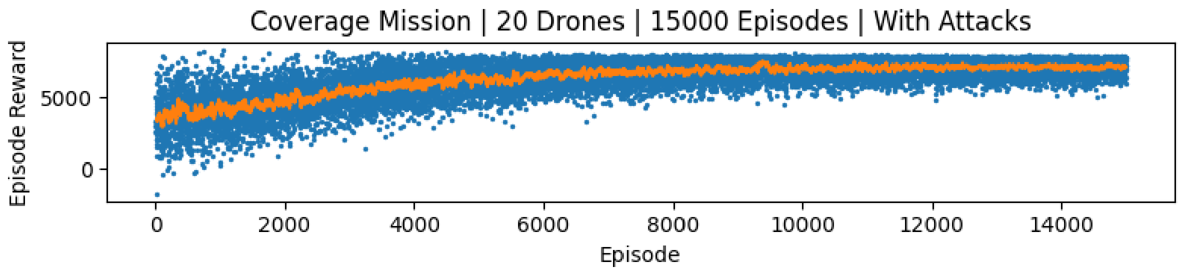


FIGURE 5.31 – Task 4 – Accumulated Reward per Episode, AR, with the simplified Reward Function for a network of 20 agents (scatter plot). Accumulated Reward Moving Average of 25 episodes, AR^{MA} (orange line).

5.4.4 Comparison of Training with and without Attacks

The last analysis to be made was if enabling attacks during training was beneficial to the agent or not. Given that attacks decrease the number of agents (and experience generated) in the network, and there is not a explicit penalization for the network when it disconnects, we raised the question if it would not be better to train the network with the Robustness Level reward without attacks, leaving this for evaluation only.

We re-ran the training of Robustness to Failures once again similarly to Section 5.4.2, with the reward function given by Algorithm 9. When comparing the Accumulated Reward per episode plots generated by the two training sessions, it was confirmed that when disabling attacks during training the agent converged faster to a solution.

In terms of behavior learned, there was not a clear difference between both networks, with Robustness Levels being similar throughout the episodes and no disparity in any other task as Area Coverage or Connectivity Maintenance. The only benefit to be cited is time spent during training before convergence.

6 Conclusion

This chapter contains the final thoughts on this work, presenting a final conclusion and analysis on what we achieved, problems that we found and possible ideas of improvements.

6.1 Final Thoughts

Summarizing all the training the network employed, we had a total of 57,000 episodes per agent, this number multiplied by the number of agents in the network, and by the 8 parallel environments running, generates experience equivalent to a few millions episodes if we were training only one agent. This marks one of the benefits of training a Multi-Agent Reinforcement Learning model with naïve parameters sharing.

In terms of the behaviors that the agent learned in the context of multicriteria missions, and more specifically coverage missions, in Task 1 - Move to Target Location it was quite straightforward with the implemented reward to teach the agent to travel through the obstacles and reach target location. This task was accomplished without any issues.

For Task 2 - Connectivity Maintenance, we started breaking this task into two subtasks, with the first tackling just the connectivity problem, and the second adding more information about the algebraic connectivity threshold. We concluded that this was not necessary, and we could have applied the same function during the entirety of the task. This would be equivalent since the Mean Algebraic Connectivity measured showed that the network learned to be well connected from the start, already surpassing any determined threshold. Also, since we were not dealing with a network of 20 agents and under attacks, for which these thresholds where devised, this comparison failed to generate any benefit for the network since the algebraic connectivity would naturally be larger.

During the task a few different behaviors generated by the agent were noticed, as following to target but always with a slight bias pointed to the center of the network. We believe this is the behavior that keeps the network connected during the episode. Also, regrouping is a behavior that we believe came by chance, and that the network did not learned this on purpose. A possible explanation for this behavior is that, when the algebraic connectivity drops to zero, the model starts to give more value to other components from the space state, one of them being the potential field vector from its neighbors. The agents that lost their neighbor seem to increase the bias of pointing to the center of the network, decreasing their speed to reach the target. The disconnected agent, although it should point the the network as well, does not have this input, so it continues to follow to target location without any bias. As mentioned before, this situation creates an issue, that if the agent is not behind the network that awaits for it, it will never find its group again, jeopardizing the mission.

A possible way to deal with this is to maintain in each agent a register containing important variables as network's last seen location, and use this information to try to regroup with the network. The RL model could be trained to do this, or could be deactivated momentarily for this.

By the end of training 10 agents, the task was considered accomplished, for the network demonstrated a high performance in traversing the field while maintaining its connectivity.

In Task 3 - Area Coverage, the agent failed to achieve the expected results. We would expected that the agent would reach a ratio of 25%, following the same results generated by Ghedini *et al.* (2018) when using a Voronoi Tesselation to adjust the topology, but it reached a maximum value of only 7%. The problem here was that the Connectivity Maintenance behavior learned previously overcame the learning of Area Coverage, causing too much overlap between the agent's observable areas.

After testing a few different reward functions to the problem, it was noted that the network was not fully learning this behavior. It reached its peak with a network of 15 agents, but soon after started to underperform when the network reached 20 agents. This task was not perfectly accomplished, and is a possible subject to further study.

In our final task, Task 4 - Robustness to Failures, given the great similarity between Task 2 and Task 4, the network started showing good results even before training. When our attack policy was enabled the network endured the attacks and kept a Robustness Level over 10% during the entirety of the episodes, already matching the threshold added in the reward function.

After training, the Robustness Level received a slight increase in the beginning of episodes, now with the entire episode being over 20%. One noticeable thing is that the network took way longer to converge in this scenario than in any other. We believe that this happened because the removal of agents in the network also removed the generated experience, causing it to suffer from the high variance.

We compared the training of the network with and without the attack policy enabled, and without the attacks the network learned the same behavior faster, with only half of the number of episodes.

Also, after seeing the results from Task 4, we re-ran the training session, this time removing the connectivity reward. Since both rewards are similar, we wanted to compare the performance of a network trained in this last step only with the robustness reward. And, similar to a result presented by Ghedini *et al.* (2018), our network learned to keep a high Algebraic Connectivity and Robustness Level even with the connectivity reward removed.

By the end, we consider Task 4 to be accomplished, and with it all tasks were completed. We can observe that the order in which each task was introduced to the network affected which it learned better. In our case, since we began by training Connectivity Maintenance, the network became an expert in maintaining a well connected behavior even during attacks and prior to any knowledge of Robustness Level. It is possible that if had began the work by training the agents to spread throughout the field, the results would have been different.

Analyzing the values of mean algebraic connectivity and mean area coverage percentage by the end of all 4 tasks, we identified that the network's algebraic connectivity remained way above the devised threshold even when under attack, about 4 times the threshold's value. This result shows that the agent is majorly focusing in keeping itself close to all other drones while overlooking area coverage. We identify that there is a trade-off opportunity, to lower the value of algebraic connectivity and increase area coverage, and for this the agent should continue the training of Task 3 on top of all learned behaviors.

For the overall purpose of a coverage mission, the network performed well in terms of connectivity and robustness to failures, but there is still room for improvement in area coverage. For a multicriteria mission, the agent presented great results in learning several different behaviors simultaneously in an extremely complex scenario.

At last, this work presented that it is possible to use Deep Reinforcement Learning techniques, in addition to Multi-Agent Reinforcement Learning, to train an agent able of cooperation with its peers and creation of a resilient network topology to perform multicriteria missions.

The presented training framework can be later used as a base to train other decentralized networks in different missions. Using a trained resilient network as starting point we can devise new reward functions based on the task in hand and take advantage of transfer learning to continue training.

6.2 Future Works

The following presents a few ideas of possible enhancements and research paths based on this work.

- Continue training of Task 3 – Area Coverage on top of the final network, focusing in exchanging algebraic connectivity for area coverage. Possibly devise a reward function that calculates the difference between the current topology and a Voronoi Tesselation of the network.
- Train different multicriteria missions besides coverage missions, teaching other behaviors on top of connectivity and robustness.
- Train a Reinforcement Learning Model to optimize the weights of the controller devised by Ghedini *et al.* (2018).
- Introduce a battery parameter to simulate the drone's decline of usability through time, and identify the changes that the topology suffers to adequate itself for a drone that is close to shutdown.
- Change the input vector of the network for an image of the agent surroundings, removing all complexity of calculating the potential field of nearest neighbors and obstacles, leaving for a Convolutional Neural Network to identify these aspects.
- Introduce moving obstacles and enemies in the field so the agent has to take into account their future positions for path planning.

Bibliography

BOYD, R.; RICHARDSON, P. J. Culture and the evolution of human cooperation. **The Royal Society - Philosofical Transactions**, vol. 364, n. 1533, p. 3281–3288, 2009. Available from Internet: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2781880/>. Cited: 03 jul. 2022.

CHRISTIANOS, F.; PAPOUDAKIS, G.; RAHMAN, A.; ALBRECHT, S. V. Scaling multi-agent reinforcement learning with selective parameter sharing. 2021.

DEEPMIND: Alphago is the first computer program to defeat a professional human go player. 2016. Available from Internet: <https://deepmind.com/research/case-studies/alphago-the-story-so-far>. Cited: 03 jul. 2022.

DEEPMIND: Alphazero: Shedding new light on chess, shogi, and go. 2018. Available from Internet: <deepmind.com/blog/alphazero-shedding-new-light-on-chess-shogi-and-go>. Cited: 03 jul. 2022.

DEEPMIND: Muzero: Mastering go, chess, shogi and atari without rules. 2020. Available from Internet: <https://www.deepmind.com/blog/muzero-mastering-go-chess-shogi-and-atari-without-rules>. Cited: 03 jul. 2022.

DEEPMIND. Open problems in cooperative ai. **NeurIPS 2020 Cooperative AI Workshop**, 2020.

GHEDINI, C.; H.C.RIBEIRO, C.; SABATTINI, L. Toward efficient adaptive ad-hoc multi-robot network topologies. **Ad Hoc Networks**, vol. 74, p. 57–70, 2018. Available from Internet: <https://doi.org/10.1016/j.adhoc.2018.03.012>. Cited: 03 jul. 2022.

GHEDINI, C. G.; RIBEIRO, C. H. Rethinking failure and attack tolerance assessment in complex networks. **Physica A: Statistical Mechanics and its Applications**, vol. 390, n. 23, p. 4684–4691, 2011. ISSN 0378-4371. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S0378437111005322>.

GODSIL, C.; ROYLE, G. **Algebraic Graph Theory**. [S.l.]: MIT Springer, 2001.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016. `http://www.deeplearningbook.org`.

GUPTA, A.; NALLANTHIGHAL, R. Decentralized multi-agent formation control via deep reinforcement learning. 2021. Available from Internet: <https://www.scitepress.org/Papers/2021/102413/102413.pdf>. Cited: 04 jul. 2022.

HENTOUT, A. Human–robot interaction in industrial collaborative robotics: a literature review of the decade 2008–2017. **Advanced Robotics**, vol. 33, n. 15–16, p. 1043–1061, 2019. Available from Internet: <https://doi.org/10.1080/01691864.2019.1636714>. Cited: 03 jul. 2022.

INTEL. **Intel® DevCloud for oneAPI**. 2022. https://devcloud.intel.com/oneapi/.

KASSAB, R.; DESTOUNIS, A.; TSILIMANTOS, D.; DEBBAH, M. Multi-agent deep stochastic policy gradient for event based dynamic spectrum access. 2020.

LEIBO, J. Z.; ZAMBALDI, V.; LANCTOT, M.; MARECKI, J.; GRAEPEL, T. Multi-agent reinforcement learning in sequential social dilemmas. 2017.

MCGEE: Employees work alongside robotic arms, manufactured by kuka, at mercedes-benz's factory in bremen, germany. 2017. Available from Internet: <https://www.ft.com/content/f80c390c-5293-11e7-bfb8-997009366969>. Cited: 03 jul. 2022.

MINELLI, M.; KAUFMANN, M.; PANERATI, J.; GHEDINI, C.; BELTRAME, G.; SABATTINI, L. Stop, think, and roll: Online gain optimization for resilient multi-robot topologies. **Distributed Autonomous Robotic Systems**, p. 357–370, 2018. Available from Internet: <https://doi.org/10.1007/978-3-030-05816-6_25>. Cited: 04 jul. 2022.

OPENAI. **OpenAI Five**. 2018. https://blog.openai.com/openai-five/.

OWEN-SMITH, J. **Network Theory: The Basics**. 2017. https://www.oecd.org/sti/inno/41858618.pdf.

PYGAME. **PyGame**. 2022. https://www.pygame.org/wiki/about.

SCHULMAN, J.; LEVINE, S.; MORITZ, P.; JORDAN, M. I.; ABBEEL, P. Trust region policy optimization. 2015.

SCHULMAN, J.; WOLSKI, F.; DHARIWAL, P.; RADFORD, A.; KLIMOV, O. Proximal policy optimization algorithms. 2017.

SUTTON, R. S.; BARTO, A. G. **Reinforcement Learning: An Introduction**. [S.l.]: MIT Press, 2018. http://incompleteideas.net/book/the-book-2nd.html.

WANG, Z.; BAPST, V.; HEESS, N.; MNIH, V.; MUNOS, R.; KAVUKCUOGLU, K.; FREITAS, N. de. Sample efficient actor-critic with experience replay. 2016.

YANG, G.-Z.; BELLINGHAM, J.; DUPONT., P. E. The grand challenges of science robotics. **Science Robotics**, vol. 3, n. 14, 2018. Available from Internet: <https://www.science.org/doi/10.1126/scirobotics.aar7650>. Cited: 03 jul. 2022.

FOLHA DE REGISTRO DO DOCUMENTO

1. CLASSIFICAÇÃO/TIPO TC	2. DATA 22 de novembro de 2022	3. DOCUMENTO Nº DCTA/ITA/TC-064/2022	4. Nº DE PÁGINAS 74

5. TÍTULO E SUBTÍTULO:

Deep Reinforcement Learning for Multi-Agent Systems in Multicriteria Missions

6. AUTOR(ES):

Nicholas Scharan Cysne

7. INSTITUIÇÃO(ÕES)/ÓRGÃO(S) INTERNO(S)/DIVISÃO(ÕES):

Instituto Tecnológico de Aeronáutica – ITA

8. PALAVRAS-CHAVE SUGERIDAS PELO AUTOR:

Multi-Agent Systems; Deep Reinforcement Learning; Multi-Agent Reinforcement Learning; Decentralized Networks; Multicriteria Missions

9. PALAVRAS-CHAVE RESULTANTES DE INDEXAÇÃO:

Aprendizagem (inteligência artificial); Sistemas multiagentes; Teoria multicritério da decisão; Dinâmicas de robôs; Máquinas aprendizes; Controle robusto; Computação

10. APRESENTAÇÃO: (**X**) **Nacional** ( ) **Internacional**

ITA, São José dos Campos. Curso de Graduação em Engenharia Eletrônica. Orientador: Prof. Dr. Carlos Henrique Costa Ribeiro; Coorientadora: Profa. Cinara Guellner Ghedini. Publicado em 2022.

11. RESUMO:

Unmanned Aerial Vehicles (UAVs) can act in a wide range of scenarios in which sending human personal is of great risk or difficulty, such as exploration and reconnaissance missions in dangerous areas or of extreme conditions for human beings, even in disaster sites or military operations. These missions commonly employ multiple UAVs, forming a decentralized network that allows communication and cooperation between them. For the success of this type of formation, it is necessary to have a robust network topology, that is, capable of maintaining certain characteristics such as connectivity between all agents in the network and resilience to failures. This work deals especially with Multicriteria Missions, characterized by having several distinct activities to be performed during the mission, composing complex scenarios in which robot cooperation is essential for mission success. Such scenarios are good candidates to apply machine learning techniques due to the large number of possible configurations and lack of a model that determines only one optimal solution. This work proposes to apply Deep Reinforcement Learning techniques to train an agent, acting as an UAV, that cooperates with replicas of itself to create a network topology able to perform Multicriteria Missions. The agent was trained following 4 distinct tasks to simulate such kind of mission, as follows: Move to Target Location, Connectivity Maintenance, Area Coverage, and Robustness to Failures. We applied Deep Reinforcement Learning and Multi-Agent Reinforcement Learning techniques, such as Proximal Policy Optimization (PPO) and Centralized Training with Decentralized Execution (CTDE), during training. As a result, we achieved at the end a network composed by 20 UAVs, all of them acting in cooperation to perform the mentioned tasks. The network presented great results in the tasks of Move to Target Location, Connectivity Maintenance, and Robustness to Failures, but underperformed in Area Coverage. These results suggest that the use of MARL is able to achieve good performance when training a set of agents for Multicriteria Missions, but there is still room for improvement and exploration of other techniques.

12. GRAU DE SIGILO:

(**X**) **OSTENSIVO** ( ) **RESERVADO** ( ) **SECRETO**