INSTITUTO TECNOLÓGICO DE AERONÁUTICA

Nicholas Scharan Cysne

DEEP REINFORCEMENT LEARNING FOR UAVS IN COVERAGE MISSIONS

Final Paper
2022

Course of Electronics Engineering

CDU ???.??

Nicholas Scharan Cysne

DEEP REINFORCEMENT LEARNING FOR UAVS IN COVERAGE MISSIONS

Advisor

Prof. Dr. Carlos Henrique Costa Ribeiro (ITA)

Co-advisor

Prof. Dra. 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 UAVs in Coverage Missions / Nicholas Scharan Cysne.
São José dos Campos, 2022.
35f.

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

1. Multi-Agent Systems. 2. Deep Reinforcement Learning. 3. Cooperative AI. 4. Formation Control. 5. UAVs. I. Instituto Tecnológico de Aeronáutica. II. Title.

BIBLIOGRAPHIC REFERENCE

SCHARAN CYSNE, Nicholas. **Deep Reinforcement Learning for UAVs in Coverage Missions**. 2022. 35f. Final paper (Undergraduation study) – Instituto Tecnológico de Aeronáutica, São José dos Campos.

CESSION OF RIGHTS

AUTHOR'S NAME: Nicholas Scharan Cysne
PUBLICATION TITLE: Deep Reinforcement Learning for UAVs in Coverage Missions.
PUBLICATION KIND/YEAR: Final paper (Undergraduation study) / 2022

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

DEEP REINFORCEMENT LEARNING FOR UAVS IN COVERAGE 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: junho 15, 2022.

À minha família.

Acknowledgments

To be written.

"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, auxiliar em locais de desastres ou atuar 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 alta conectividade e resistência à ataques. Este trabalho trata em especial de missões de cobertura, caracterizada por utilizar uma rede de VANTs a fim explorar uma região pré-determinada.

De forma geral, a cooperação entre robôs é um grande desafio técnico, e muitas vezes é abordado por simples protocolos de comunicação e combinações de heurísticas com leis de controle. Missões de cobertura compõe um cenário complexo em que a cooperação de robôs é essencial para o sucesso da missão, 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 único. Este trabalho propõe aplicar técnicas de Aprendizado por Reforço Multiagente para treinar um agente capaz de cooperação com replicas de si mesmo a fim de criar topologias de rede capazes de operar em missões de cobertura.

Abstract

Unmanned Aerial Vehicles (UAVs) can act in a 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, assisting in disaster sites or perform military operations. These missions commonly employ multiple UAVs forming a 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 high connectivity and resistance to attacks. This work deals especially with coverage missions, characterized by using a network of UAVs in order to explore a predetermined region.

In general, cooperation between robots is a major technical challenge, and is often addressed by simple communication protocols and combinations of heuristics with control laws. Coverage missions compose 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 single unique model. This work proposes to apply Multi-Agent Reinforcement Learning techniques to train an agent that is capable of cooperation with replicas of itself to create a network topology able to perform coverage missions.

List of Figures

List of Tables

List of Abbreviations and Acronyms

UAV	Unmanned Aerial Vehicle
AI	Artficial Intelligence
RL	Reinforcement Learning
ANN	Artificial Neural Network
NN	Neural Network
MDP	Markov Decision Process
DQN	Deep Q-Learning Network
PPO	Proximal Policy Optimization
TRPO	Trust Region Policy Optimization
ACER	Sample Efficient Actor-Critic with Experience Replay
MASDPG	Multi-Agent Stochastic Deep Policy Gradient

List of Symbols

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.

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

Robots can also be applied outside the industry scope. In Aeronautics, Unmanned Aerial Vehicles (UAVs) are drones which 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), sets 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 current 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 previous 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.

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.

Another field of great development in previous years is Multi-Agent Reinforcement Learning, specially in cooperative dynamics. Although so far most of the research made in this field have been on Competitive AI, such as applications in Game Theory, Multi-Agent systems and Cooperative AI is a field of great promising since it can be found in several scenarios ranging from self-driving cars, and collaborative work to global commerce, 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). With an increasing number of AI agents becoming part of our lives, it is of most importance that they learn to cooperate with each other and with our own species.

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".

Coverage missions are characterized as missions where a network of agents explores a region, avoiding obstacles (radars or enemy drones) while maintaining connectivity with each other and communicating the discovery of possible points of interest. The success of such missions in most cases depends on: area coverage, connectivity maintenance, and robustness maintenance.

While area coverage is more closely related to the mission purpose itself, connectivity and robustness maintenance are more related to safe-guarding the conditions of the network throughout the mission. Robustness is of singular importance in this scenario, given that we are dealing with an Ad Hoc Network, and one that can suffer from enemy attacks eliminating a node, so it is important that the network maintain local configurations of high connectivity.

The topology of the network is something dynamic and defined by the agents themselves, so to achieve the expected objectives, each drone must be able to cooperate and exchange information with others.

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 network topology able to perform coverage missions. To achieve this goal, we break the problem in smaller objectives as follow:

1. Build an agent that moves from left to right to target location with obstacle avoidance by using potential field algorithm.
2. Train a small group of RL agents that executes "Objective 1" while performs:
    a. Area coverage.
    b. Connectivity maintenance.
    c. Robustness maintenance.
3. Gradual increase in the number of agents in the network.

The final trained agent performance will be compared with the performance of an agent following the control laws formulated by Ghedini *et al.* (2018). For the comparison to be valid, we set a target number of agents in the network of 20 agents, this way we can compare behaviors that are expected to be present in large networks.

1.4 Literature Review

In the last years the field of Deep Reinforcement Learning delivered several interesting results, 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). OpenAI Five is an example of that, which shows five RL agents in a Cooperative/Competitive environment, where they must cooperate to achieve the ultimate goal of winning a game of Dota 2 against a human-only team. Another recent researches in Cooperative AI includes research in communication, commitment, and trust between agents (DEEPMIND, 2020).

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

Ghedini *et al.* addresses the issue 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 improvement (GHEDINI *et al.*, 2018). The model resulted in interesting results which demonstrated that it's possible to determine a set of weights for each control law that achieves a resilient network topology. The topology is able to enhance it's coverage performance while avoiding collision and maintaining the high network connectivity high and robustness.

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

The application of Deep RL to network topologies of drones have long been studied, generating interesting results (GUPTA; NALLANTHIGHAL, 2021), but the applications of Deep RL specifically for coverage missions is scarce. In this sense, we will be basing our work in papers similar to Gupta *et al.* (2021).

1.5 Contributions

With this work we intend to propose a Deep Reinforcement Learning methodology for training teams of UAVs for coverage missions. Our main contribution is to promote a simple RL-based algorithm that trains a network of agents capable of obstacle avoidance, limited-ranged communication, connectivity maintenance and robustness maintenance in coverage missions.

1.6 Outline of this Work

The remainder of the work can be described as follows:

- Chapter 2 - Covers Network Topology theory necessary for problem modeling.
- Chapter 3 - Covers Deep Learning and Reinforcement Learning theory.
- Chapter 4 - Outlines problem modeling, core tools to be used, and action plan.

2 Multi-Agent Network Topologies

2.1 Network Theory

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 entries. For example, Figure 2.1 presents a network that represents the relationships of inventors and inventions in Silicon Valley from 1986 to 1990.

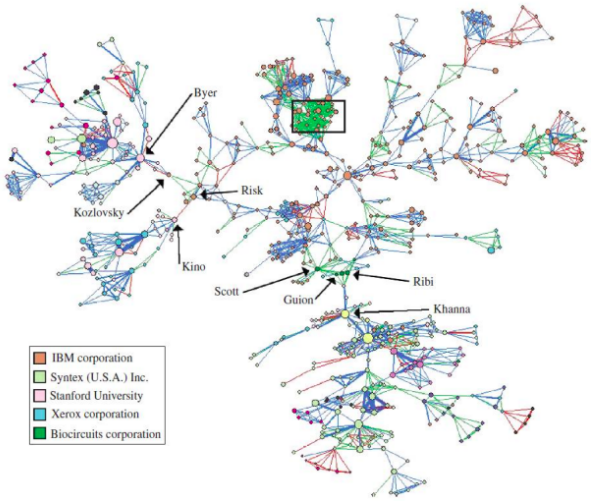

FIGURE 2.1 – Inventors of Silicon Valley's largest component in 1986 - 1990 by Assignee and Importance of Inventions (FLEMING *et al.*, 2007).

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

2.1.1 Algebraic Connectivity

For all following concepts, consider an undirected graph $\mathcal{G}$, 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 $\mathcal{G}$. The algebraic connectivity helps in analyzing the robustness of networks.

2.1.2 Betweenness Centrality

Betweenness Centrality (BC) is a measure of influence a node has in a graph, 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.

Equation (2.2) represents the betweenness centrality of node v.

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

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.

2.1.3 Robustness of a Network

We may define the *Robustness* of a network 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).

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.

3 Machine Learning

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

3.1 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. This kind of problems needs a certain abstraction from the machine to understand for example what a car is, or how a person differs from a post light on the corner of a street. 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 goes beyond the simple and well-scoped problems of regression, classification, clustering, and others. 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.

3.1.1 Deep Neural Networks

We can build a Deep Neural Network by stacking multiple layers of neurons on top of each other, as shown in Figure 3.1. In the literature, a neural network with 2 or more hidden layers can already be considered a deep neural network (GOODFELLOW *et al.*, 2016).

On Deep Neural Networks, similar to simple neural networks, the main objective is that the model learns a function ψ that solves our problem. For this, we create a model $y = f(x, \theta)$, where θ is a vector of parameters of the model to be learned, that can learn $\psi(x)$ with the help of an optimization algorithm. The optimization algorithm updates the

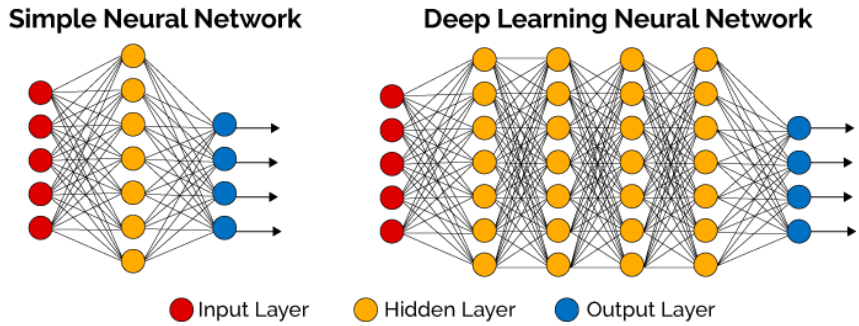

FIGURE 3.1 – Comparison of a Shallow Neural Network (left) to a Deep Neural Network (right).

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.

3.2 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).

3.2.1 Key Definitions

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

Figure 3.2 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}.

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 functions as shown in (3.1),

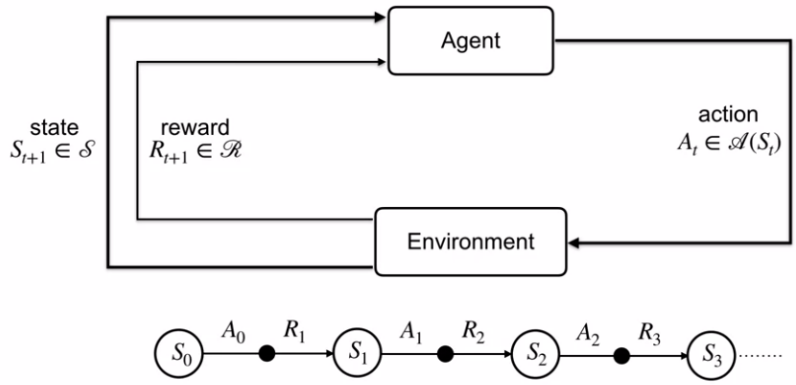

FIGURE 3.2 – Reinforcement Learning framework of agent - environment interaction (SUTTON; BARTO, 2018).

where for each state there is only one possible action.

$$\pi(s) = a. \tag{3.1}$$

Stochastic policies return a probability distribution over the action space given a current state, as shown in (3.2). This way, an agent at time t and state s_t has a probability p_i of performing the action $a_{t,i}$. Also, from now on, due to the scope of the work, we are going to treat all policies as stochastic policies.

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]. \tag{3.2}$$

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 follows:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}. \tag{3.3}$$

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 in the final iteration or episode. The constant *gamma* 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 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 returns (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 shown in (3.4).

$$V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]. \tag{3.4}$$

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 shown in (3.5).

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]. \tag{3.5}$$

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:

$$\mathcal{P}_{a,s,s'} = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a]. \tag{3.6}$$

The matrix stores all probabilities of the state transitioning from s to s' given an action.

3.2.2 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 in (3.7).

$$\mathbb{P}[S_{t+1} = s'|S_t = s] = \mathbb{P}[S_{t+1} = s'|S_t = s_t, S_{t-1} = s_{t-1}, ..., S_1 = s_1]. \tag{3.7}$$

3.2.3 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 $J(\theta)$, shown in (3.8), is maximized with gradient ascent to achieve the optimal policy, (3.9).

$$J(\theta) = \mathbb{E}_{\pi\theta}[G_t], \tag{3.8}$$

$$\theta_{k+1} = \theta_k + \alpha_\theta \nabla_\theta J(\theta), \tag{3.9}$$

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. First, policy gradients tend to take a long time to converge and have high variance. Additionally, they usually converge only to a local optima (SUTTON; BARTO, 2018). Second, 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 $J(\theta)$:

$$\nabla_\theta J(\theta) = \int \nabla_\theta \pi_\theta(\tau) G_t(\tau) d\tau. \tag{3.10}$$

Now, we can use the identity showed in (3.11) to rearrange (3.10) in the next step.

$$\nabla_\theta f(\theta) = \frac{f(\theta)\nabla_\theta f(\theta)}{f(\theta)} = f(\theta)\nabla_\theta \log(f(\theta)), \tag{3.11}$$

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

where (3.12) can be simplified as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\theta\pi}[\nabla_\theta \log(\pi_\theta) G_t], \tag{3.13}$$

which is a sampling technique for estimating the gradient.

3.2.4 Actor-Critic Methods

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 Methods.

In this method we have two models operating at the same time, as defined:

- *Actor*: model that interacts with the environment and that 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 calculate the Q-Value for that scenario with parameters θ_v.

For the Critic to be able to the evaluate the Q-value of the Actor's actions, we define the Advantage Function $A_\pi(s, a)$, given by (3.14).

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s) \tag{3.14}$$

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 in the calculation of these values of action-value and state-value functions, we can use the Temporal Difference approximation:

$$A_\pi(s, a) \approx r + \gamma V_\pi(s') - V_\pi(s). \tag{3.15}$$

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: θ_a and θ_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

3.2.5 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.

Equation (3.16) presents the clipped cost function, also called clipped surrogate objective function.

$$L^{CLIP}(\theta) = \mathbb{E}_{\theta,\pi}[min(r_t(\theta')A_w, clip(r_t(\theta')A_w, 1-\epsilon, 1+\epsilon))], \tag{3.16}$$

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

$$r_t(\theta') = \frac{\pi_\theta(a|s)}{\pi_{\theta'}(a|s)}, \tag{3.17}$$

and $[1-\epsilon, 1+\epsilon]$ if the bounded update interval for policy π_θ. We can see its implementation with Actor-Critic in Algorithm 2.

Algorithm 2 Proximal Policy Optimization with Clipped Objective Function

```
Initialize step counter: t ← 0
Get initial parameters: θ_0
while t < T_max do
  // Actor-Critic Interaction
  Collect set of trajectories D_k on policy π(a|s, θ_k)
  Compute TD approximation: A_{π,k} ≈ r + γV_π(s') − V_π(s)
  Update policy parameters: θ_{k+1} ← argmax_θ(L^{CLIP}(θ_k))
  t ← t + 1
end while
```

3.2.6 Multi-Agent Reinforcement Learning

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, so a common technique to scale 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.

3.2.7 Multi-Agent Deep Stochastic Policy Gradient

A popular paradigm in cooperative MARL is *Centralised Training with Decentralised 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 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. The update of the actor-critic parameters is made with the resulting reward, this way each agent contributes to the update equally.

4 Methodology

In this chapter, we cover the methodology to be used in this work and also presents a plan of future activities.

4.1 Problem Modeling

We will 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 of our network, and every edge $e_{ij} \in \mathcal{E}(\mathcal{G})$ represents an available interaction between drones i and j.

Additionally, the problem will be modeled as a common MDP, with multiple agents interacting with the environment, and will follow a Deep Reinforcement Learning implementation.

The input of the Deep Neural Network will be the state of the agent and the output will be its action, being direction and step size.

4.2 Tools and Frameworks

In this work there will be several tools and frameworks besides the MADSPG model, such as PyGame, Stable Baselines 3, and Intel® DevCloud oneAPI. Below are brief summaries of these tools and frameworks at our disposal.

4.2.1 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 of training and evaluation of episodes, we will be building a sim-

ulator using the PyGame library for Python. Most of the simulation logic and evaluation metrics was already designed by Ghedini *et al.* (2018) for MatLab®, so we will take advantage of the material already built for our own simulator by transcribing these tools to Python.

This simulator will be used for occasional visualization only, and will be deactivated during training for performance purposes.

4.2.2 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 would help researchers so that they would not have to waste time in tuning algorithms and looking for bugs, instead of actually advancing in their research.

For this work, we will be using the Stable Baselines 3 platform for the implementation of Proximal Policy Optimization and MADSPG.

4.2.3 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 until our model begins to show results.

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.

4.3 Training

As mentioned in Section 1.3, we can derive smaller objectives to achieve our main goal. First, we will use a potential field algorithm implemented by Ghedini *et al.* (2018) to enable our agent to move to target location with obstacle avoidance.

Second, we will start training a set of RL agents in three separate tasks: area coverage,

connectivity maintenance, and robustness maintenance. This is trained on top of the controller that enables the agent to move to target location.

For each of the three tasks we will derive a reward function. When the agent learns to perform one of the tasks, we can compose ensembles of reward functions based on the previously derived rewards, first trying to compose two functions and then finally three functions to try the best way of training the agent on all tasks simultaneously. Here the agent must learn to perform the given tasks while moving to target location.

As a final step, we will increase the number of agents in the network to study how the network topology behaves. We expect to see different and more complex strategies being executed with the increase of the number of agents. We set a target of a minimum of 20 agents in the network so we can compare its results with the control law proposed by Ghedini *et al.* (2018).

The scenario in which the agent will be training will be a field of predetermined area, and will have randomly generated obstacles throughout the field, varying in number and position. The network will receive random attacks on its nodes, eliminating them. The final objective is that the network traverse this field left to right, optimizing the area coverage, maintaining connectivity and robustness to attacks.

4.4 Plan of Action

We propose the following action plan for this work:

- **15/July**: Modeling of the problem as a MDP.
- **29/July**: Define reward signals for each of the tasks.
- **29/July**: Define Deep Neural Network's architecture.
- **12/August**: Design simulator and implement motion plan with potential fields.
- **12/August**: Start training of the network.
- **23/September**: Present results for:
  - Area coverage;
  - Connectivity maintenance;
  - Robustness maintenance.
- **21/October**: Present results for:
  - Gradual increase of number of agents in the network;

Bibliography

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

CHRISTIANOS, F.; PAPOUDAKIS, G.; RAHMAN, A.; ALBRECHT, S. V. Scaling multi-agent reinforcement learning with selective parameter sharing. 2021. Disponível em: <https://arxiv.org/pdf/2102.07475.pdf>. Acesso em: 06 jul. 2022.

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

DEEPMIND: Alphazero: Shedding new light on chess, shogi, and go. 2018. Disponível em: <deepmind.com/blog/alphazero-shedding-new-light-on-chess-shogi-and-go>. Acesso em: 03 jul. 2022.

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

DEEPMIND. Open problems in cooperative ai. **NeurIPS 2020 Cooperative AI Workshop**, 2020. Disponível em: <https://arxiv.org/abs/2012.08630>. Acesso em: 03 jul. 2022.

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

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. Disponível em: <https://www.scitepress.org/Papers/2021/102413/102413.pdf>. Acesso em: 04 jul. 2022.

HENTOUT, A. Human–robot interaction in industrial collaborative robotics: a literature review of the decade 2008–2017. **Advanced Robotics**, v. 33, n. 15–16, p. 1043–1061, 2019. Disponível em: <https://doi.org/10.1080/01691864.2019.1636714>. Acesso em: 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. Disponível em: <https://arxiv.org/abs/2004.02656>. Acesso em: 06 jul. 2022.

LEIBO, J. Z.; ZAMBALDI, V.; LANCTOT, M.; MARECKI, J.; GRAEPEL, T. Multi-agent reinforcement learning in sequential social dilemmas. 2017. Disponível em: <https://arxiv.org/abs/1702.03037>. Acesso em: 04 jul. 2022.

MCGEE: Employees work alongside robotic arms, manufactured by kuka, at mercedes-benz's factory in bremen, germany. 2017. Disponível em: <https://www.ft.com/content/f80c390c-5293-11e7-bfb8-997009366969>. Acesso em: 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. Disponível em: <https://doi.org/10.1007/978-3-030-05816-6_25>. Acesso em: 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. Disponível em: <https://arxiv.org/abs/1502.05477>. Acesso em: 06 jul. 2022.

SCHULMAN, J.; WOLSKI, F.; DHARIWAL, P.; RADFORD, A.; KLIMOV, O. Proximal policy optimization algorithms. 2017. Disponível em: <https://arxiv.org/abs/1707.06347>. Acesso em: 06 jul. 2022.

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. Disponível em: <https://arxiv.org/abs/1611.01224>. Acesso em: 06 jul. 2022.

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

FOLHA DE REGISTRO DO DOCUMENTO

1. CLASSIFICAÇÃO/TIPO	2. DATA	3. DOCUMENTO Nº	4. Nº DE PÁGINAS
TC	15 de junho de 2022	DCTA/ITA/DM-018/2022	35

5. TÍTULO E SUBTÍTULO:

Deep Reinforcement Learning for UAVs in Coverage 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; Reinforcement Learning; Formation Control; UAVs

9. PALAVRAS-CHAVE RESULTANTES DE INDEXAÇÃO:

Multi-Agent Systems; Reinforcement Learning; Formation Control; UAVs

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

Trabalho de Graduação, ITA, São José dos Campos, 2022. 35 páginas.

11. 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, auxiliar em locais de desastres ou atuar 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 alta conectividade e resistência à ataques. Este trabalho trata em especial de missões de cobertura, caracterizada por utilizar uma rede de VANTs a fim explorar uma região pré-determinada. De forma geral, a cooperação entre robôs é um grande desafio técnico, e muitas vezes é abordado por simples protocolos de comunicação e combinações de heurísticas com leis de controle. Missões de cobertura compõe um cenário complexo em que a cooperação de robôs é essencial para o sucesso da missão, 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 único. Este trabalho propõe aplicar técnicas de Aprendizado por Reforço Multiagente para treinar um agente capaz de cooperação com replicas de si mesmo a fim de criar topologias de rede capazes de operar em missões de cobertura.

12. GRAU DE SIGILO:

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