FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Multi-Agent Deep Reinforcement Learning for Resource Management in Earth Observation Satellite Constellations

João Henrique Luz

Master's Programme in Informatics and Computing Engineering

Supervisor: Rosaldo J. F. Rossetti

Co-supervisor: Zafeiris Kokkinogenis

July 30, 2022

Multi-Agent Deep Reinforcement Learning for Resource Management in Earth Observation Satellite Constellations

João Henrique Luz

Master's Programme in Informatics and Computing Engineering

July 30, 2022

Abstract

There are currently over 3000 active satellites orbiting Earth, performing multiple tasks with different purposes in mind. The so-called Earth Observation Satellites (EOS) and their sensors are used to collect necessary information regarding weather, environmental changes, cartography, etc. This setting is used to leverage the present work towards analysing MADRL to address the coordination and collaboration opportunities in these constellations.

Several stakeholders can request observations from these growing constellations of satellites. However, obstacles – such as the limited computational power and orbital space or even adversarial climate conditions – can limit the observability of such satellites. For this reason, carefully assigning these operations and assuming a collaborative stance (exchanging messages, sharing resources, orbital space, and even observations) can benefit the constellations' overall efficiency and efficacy.

To study and improve interactions between these satellites and make their measurements more valuable and efficient, one can rely on the Agent Paradigm. Moreover, using Deep Reinforcement Learning (DRL) in a Multi-Agent System (MAS) makes it possible to model agents that continuously interact with each other and their environment, developing policies that optimize the sequential decision-making process involved in these interactions.

This study focuses on the formulation of a Multi-agent Deep Reinforcement Learning (MADRL) problem that addresses some of the challenges in the field of EOS systems. The proposed solution offers a comparison of algorithms, as well as an analysis of how various formulations affect the agents' performance and behaviour. Besides shaping the reward function differently, the scenarios implemented study the impact of different agents' observations on their results. Moreover, some of the experiments carried out compare the use of homogeneous and heterogeneous agents.

Although the agents presented satisfactory results in multiple scenarios, the same was not the case in the most complex environment conceived. One of the possible reasons for this subpar performance is the presence of "lazy agents" whose policies opt not to contribute to the common goal of the constellation. This suggests that further research is needed to tackle the lazy agent problem in order to improve the efficacy of the system.

Keywords: Deep Reinforcement Learning, Multi-agent System, Resource Management, Earth Observation Satellites
ACM Classification: Computing Methodologies → Machine Learning → Learning Paradigms → Reinforcement Learning → Multi-agent Reinforcement Learning

Resumo

Atualmente encontram-se mais de 3000 satélites a orbitar o planeta Terra com os mais variados propósitos. Entre esses, os chamados Satélites de Observação Terrestre (EOS) e os seus sensores são usados para recolher informações necessárias sobre o clima, as mudanças ambientais, a cartografia, etc. Estas circunstâncias são aproveitadas neste trabalho sobre MADRL para abordar as oportunidades de coordenação e colaboração nestas constelações de satélites.

São várias as entidades interessadas que podem solicitar observações às constelações de EOS. No entanto, obstáculos como o poder computacional e o espaço orbital limitados ou mesmo condições climáticas adversas, podem por vezes limitar, ou mesmo pôr em causa, a capacidade de observação dos satélites intervenientes. Por esse motivo, atribuir cuidadosamente estas operações e assumir uma postura colaborativa (troca de mensagens, partilha de recursos, espaço orbital e até observações) pode ser benéfico para a eficiência e a eficácia das constelações.

Para estudar e melhorar as interações entre esses satélites e tornar as suas medições mais valiosas e eficientes, é possível recorrer ao paradigma de Sistemas Multi-Agente (SMA). Para além disso, o uso de Deep Reinforcement Learning em Sistemas Multi-agente introduz a possibilidade de modelar agentes que interagem continuamente entre si e com o ambiente envolvente, desenvolvendo estratégias que otimizam o processo de tomada de decisão sequencial envolvido nessas interações.

Este estudo foca-se na formalização de um problema de Multi-agent Deep Reinforcement Learning (MADRL) que aborda alguns aspetos do desafio de gestão de recursos em sistemas EOS. A solução proposta oferece uma comparação entre os algoritmos usados assim como uma análise de como diferentes formalizações afetam a desempenho e comportamento dos agentes. Para além de testarem diferentes formas de moldar a função de recompensas, os cenários desenvolvidos exploram o impacto de diferentes tipos de observações. Adicionalmente, algumas das experiências, comparam o uso de agentes homogéneos e heterogéneos.

Embora os agentes tenham demonstrado um desempenho satisfatório em vários cenários, o mesmo não se verificou no ambiente mais complexo testado. Uma das possíveis razões para esta performance insuficiente é a presença de "lazy agents" - agentes que optam por não contribuir para a causa comum da constelação. Estes resultados sugerem que mais investigação é necessária para resolver este problema e para melhorar o desempenho do sistema.

Palavras-chave: Deep Reinforcement Learning, Sistemas Multi-agente, Gestão de Recursos, Satélites de Observação Terrestre
Classificação ACM: Computing Methodologies → Machine Learning → Learning Paradigms → Reinforcement Learning → Multi-agent Reinforcement Learning

Acknowledgements

As a student who is about to cross a milestone in their academic journey, I am aware that this dissertation is an opportunity to prove myself an autonomous learner and critical thinker. Nevertheless, I feel the need to recognize that my achievements are far from being the results of my efforts alone.

I want to thank my supervisors, Rosaldo J. F Rossetti and Zafeiris Kokkinogenis, for their guidance and patience throughout this process. Besides helping me find a research project that revolves around my interests and ambitions, their contributions during the most demanding stages of this work had a tremendous impact on its completion.

For helping keep me calm and being by my side every step of the way, I thank my colleague and dear friend Ana Mafalda Santos. Someone with whom I hope to develop a lot more exciting projects.

Finally, I am very grateful for my family's support. They have always been my role models and continuously encourage me to give my best in whatever I do. I hope to make them proud.

João Henrique Luz

"I visualize a time when we will be to robots what dogs are to humans, and I'm rooting for the machines."

Claude Shannon

Contents

List of Figures

List of Tables

Abbreviations

AC	Actor-Critic
A2C	Advantage Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
DDPG	Deep Deterministic Policy Gradient
Dec-POMDP	Decentralized Partially Observable Markov Decision Process
DRL	Deep Reinforcement Learning
DQN	Deep Q-Network
EOS	Earth Observation Satellite
GAN	Generative Adversarial Network
MADRL	Multi-agent Deep Reinforcement Learning
MAS	Multi-agent System
MDP	Markov Decision Process
NN	Neural Network
PER	Prioritized Experience Replay
PG	Policy Gradient
POMDP	Partially Observable Markov Decision Process
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
VDN	Value Decomposition Network

Chapter 1

Introduction

Earth observation satellites (EOS), play an essential role in our day-to-day lives. The goal of this section is to familiarize the reader with what EOS systems are, what they can do, and why they are important. In addition, this introduction will feature some of the problems the scientific community still needs to solve to optimize and make the best of these systems.

1.1 Context and Motivation

In 2021, statistics stated that there were more than 3300 active satellites orbiting the Earth. All of which could be divided into several groups depending on their usage. This dissertation will focus on the so-called Earth Observation Satellites (EOS) category, composed of around 900 satellites. These machines and their sensors collect information by reading the electromagnetic frequencies reflected by the planet's surface and atmosphere. These measurements allow the collection of detailed images considered essential for predicting the weather and dangerous environmental changes, to sectors such as agriculture, cartography, and even to guide renewable energy initiatives. There are multiple types of satellites, each with different capabilities, degrees of freedom, and different functions. The main criteria to compare them are the spatial resolution, the radiometric resolution, and finally, the temporal resolution. The first one refers to the minimum size of the objects recognized in the images. The second one describes the sensors' ability to read a specific wavelength, and the last one defines how frequently the satellite captures new observations [12].

The increase in the number of EOSs and the amount of data generated about the planet's current state led to a rise in the number of requests made by organizations that consume and depend on these observations to function. This growth makes it essential to evaluate the constellations' limitations and invest in efficiently planning and allocating the requested tasks. The management of these systems is a complex task since it has to take into account the resources of the satellites performing duties, external factors (such as adverse climacteric conditions), the interests of the entities involved as well as the economic implications of the operations [38].

Some of the problems related to EOS include: appropriately modelling and simulating these somewhat heterogeneous systems, impartially addressing the requests, managing resources (both in terms of the devices' capabilities as well as the actual orbital space in which the operations take place), scheduling operations under uncertainties, dealing with conflicts of interests in requests, dynamically make decisions and rescheduling operations and managing the interactions between satellites [32].

1.2 Objectives

As previously mentioned, improving the constellation's response to requests can require very effective management of resources as well as solid coordination among entities. For this very reason, the goal of this dissertation is to take advantage of Multi-Agent Deep Reinforcement Learning to tackle this issue.

The focus of this work is to provide a simple problem that can incorporate multiple elements of real-life scenarios as an attempt to approach this challenge from a Reinforcement Learning perspective. Besides defining such a problem, the implementation includes different scenarios in order to compare the impact of distinct formulations, more specifically of the reward functions and the scope of the agents' observations. Each variation used was enforced to implicitly suggest the environment's goal and incentivize the cooperation among satellites. While some scenarios utilize independent learners, others rely on some form of communication and full observability.

Furthermore, the study includes value-based approaches (DQN) and policy gradient models (A2C) that interact with the environment implemented to emulate the problem defined. Moreover, the environment proposed was designed as a firm foundation so that, in the future, the problem definition may grow both in realism and sophistication.

As new results came to light, the implementation went through small increments to study the impact of these modifications on the coordination and overall performance of the EOS systems.

1.3 Document Structure

This document is divided into five chapters. In the beginning, the introduction aims to provide some context to the problem as well as demonstrate its importance to the reader.The second chapter contains the analysis of the State of the Art. This chapter introduces the most relevant concepts to provide the necessary background of Reinforcement Learning, Deep Reinforcement Learning and Multi-Agent Deep Reinforcement Learning. This literature review also includes a description of the problems identified in EOS systems and some related past work accomplished in this domain. The third chapter contains a more detailed description of the problem to be solved, as well as the methodology behind the proposed solution. After presenting and discussing the results obtained in chapter 4, the dissertation moves on to the conclusion of the work and some considerations regarding future work.

Chapter 2

Literature Review

To better understand the problem at hand, there are some concepts one should first grasp. Similarly, before starting the implementation of a new solution to the issue at hand, it is essential to explore and analyse the past work that other authors have done to tackle similar problems.

This section aims to give the reader some background knowledge and showcase the relevant work that will guide the solution proposed in this dissertation. Firstly, it will go over the taxonomy of the most pertinent Reinforcement Learning and Deep Reinforcement Learning methods and demonstrate the idea behind their conception. Once this background is established, the section exhibits some of the leading domain problems. Finally, this literature review provides an explanation of some of the efforts put into solving these issues.

2.1 Background Concepts

2.1.1 Reinforcement Learning

Reinforcement Learning (RL) is a subfield of machine learning which tries to find the ideal chain of actions to solve a particular problem. In this kind of solution, the agent observes its current circumstances and attempts to map situations (referred to as states) to actions. This mapping aims to provide a strategy to achieve a specific objective – maximize a reward.

Advancements in RL have proven that this approach can provide the way to create agents capable of solving a wide range of problems and tasks, as well as adopt intelligent behaviours and strategies in dynamic and complex environments.

An essential aspect of this type of learning is that the agent relies on limited information to develop its strategies. Firstly, the agent is unaware of what actions lead to what reward. And secondly, nothing indicates the consequences that each action has in the state of neither the agent nor the surrounding environment. Since this information is never explicitly disclosed, at each step throughout several episodes, the agent has to choose between exploring the consequences of their actions and benefiting from the ones that have yielded the most rewards in the past. This

duality is known as the **exploration-exploitation trade-off** [43]. Investing a reasonable amount of exploration is crucial to the discovery of new and improved strategies. However, controlling when and how to search new alternatives is still a challenge in RL problems [33].

There are many problems and environments for which one can resort to supervised learning approaches to solve. For instance, when dealing with strategy games, one could elaborate a large dataset, and map possible game states to optimal actions. This dataset could then be used to train a model to emulate the ideal behaviour modelled by the dataset. However, this approach has two obvious disadvantages. Firstly, one would need to be able to provide a substantial and useful dataset – this, by itself, introduces some overhead to solving the problem. Secondly, a very strict mapping of states to actions can introduce some human-bias, meaning that the agent would only be as good as the player it is imitating and that by following a highly deterministic behaviour it would never develop new and improved strategies.

Reinforcement Learning, solves these problems by developing agents that "learn on the job" by interacting with their surroundings. Since all of this occurs within a feedback loop, in which the agent collects information about the state of the problem before acting and observing the reward and the new consequential state – this sequential decision problem can be described as a **Markov Decision Process** (MDP) [3].

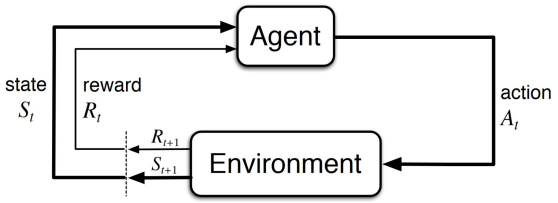

Figure 2.1: Agent-environment interaction in MDPs [42]

MDPs provide a formal framework for stochastic control optimization problems. These can be described using a 5-tuple (S,A,T,R,P). In this notation, S describes the set of possible states (state space). A denotes the set of possible actions at each state (action space). The experience of the agent can be quantified by the number of time steps t it was given the chance to choose an action. T represents the set of all steps for which the agent makes a decision, $t \in T$. The transition probability function P states the conditional probability of reaching state *s'* by leaving state s after executing action a. Finally, R gives the expected immediate reward received for reaching state s' after performing action a.

A fundamental property of MDPs is that they assume that the probability of transitioning to a state S_{n+1} does not depend on the entire history of steps that led the agent to its current position, but on the previous state S_n and the chosen action.2.1

$$P[S_{n+1} = s_{n+1} | S_n = s_n] \quad = \quad P[S_{n+1} = s_{n+1} | S_1 = s_1, S_2 = s_2, \ldots S_n = s_n] \tag{2.1}$$

The actions taken by the agents in RL problems follow a policy π. This policy states what actions to take in which state. Policies can be either deterministic – always stating a specific action to be taken for each state – or stochastic, the action is chosen according to a probability.

In most situations, agents are not able to fully observe the state of their environment. There can be aspects of the state that are unknown to the agent. This also makes it harder for the learner to understand all the intricacies of the environment's dynamics. These problems are often referred to as Partially Observable Markov Decision Processes (POMDPs)

Although the rules and dynamics of the environment are never given, the agent is able to eventually succeed by understanding what states or what state-action pairs are more profitable. Since the end goal of the learner is to maximize the expected returns, one can generally define a function G_t as follows:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T \tag{2.2}$$

This approach assumes that the environment interaction is a finite set of episodic tasks. Meaning, the agent lives in a loop of distinct experiences beginning in a *starting state* (which might be sampled from a distribution of starting states or simply predetermined) and finally ends in one of the possible *terminal states*. To adapt this concept to continuous tasks, which cannot be divided into distinct episodes, a discount factor γ. This discount is introduced as such (2.3):

$$G_t \doteq R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 \cdot R_{t+3} = \sum_{k=0}^{T} \gamma^k \cdot R_{t+1+k} = R_{t+1} + \gamma \cdot G_{t+1} \tag{2.3}$$

Taking this factor into account allows the sum of all discounted rewards in a continuous infinite task to converge, as long as γ is bounded, $0 < \gamma < 1$. This discount factor also serves to praise more immediate rewards while dwarfing distant ones. The intention is to transmit to the agent the idea that it's better to get a reward earlier rather than later. Depending on the value of γ, the agent can be considered "myopic", if γ is close to 0, or "far-sighted" γ is closer to 1.

For every state s_i in the state space S, the expected return of starting from s_i, taking the action a and following π defines the action-value q function and can be described by:

$$q_\pi(s,a) \doteq E_\pi[G_t|S_t = s, A_t = a] \doteq E_\pi[\sum_{k=0}^{T} \gamma^k \cdot R_{t+1+k}|S_t = s, A_t = a], \forall s \in S, \forall a \in A \tag{2.4}$$

Similarly, for every state s_i in the state space S, it is possible to define the expected return $V_\pi(s)$ the agent will get by following their policy, π, starting from s.

$$V_\pi(s) \doteq E_\pi[G_t|S_t = s] \doteq E_\pi[\sum_{k=0}^{T} \gamma^k \cdot R_{t+1+k}|S_t = s], \forall s \in S \tag{2.5}$$

To take actions into account, this function can be written as a function of q_π multiplied by the

probability of taking an action a according to π:

$$V_\pi(s) \doteq \sum \pi(a|s) \cdot q_\pi(s,a), \forall a \in A \tag{2.6}$$

One of the most computationally expensive methods in RL is using **Monte-Carlo** methods. This approach depends on simulating multiple interactions with the environment for each of the possible states (this strategy works for discrete action and state spaces) and examining the feedback.

One can opt for a **Model-Free** or a **Model-based** approach to solve an RL problem. While the first one relies on a value function to predict the expected return and attempts to maintain an explicit representation of the policy, the latter tries to manage a model of the environment to make more satisfying predictions of the transitions and rewards [13].

2.1.1.1 On-policy vs. Off-policy Learning

There are two main strategies regarding the evolution and improvement of the agents' inference, these are *On-policy* and *Off-policy* learning. The main difference between the two is that while the in the first approach, the agent updates the same policy it follows to choose new actions, the latter improves a policy distinct from the one creating the transitions and consequently the experience necessary for learning [42]. This distinction, makes the off-policy methods much more sample efficient, since they allow the use of experience replay, by sampling previous non-correlated transitions to update the policy. By contrast, doing the same with on-policy methods would introduce a bias since the transitions between states would not be acquired by resorting to the current policy π.

2.1.1.2 Q-Learning

The family of **value-based** methods constitutes the set of RL approaches that define a policy by first perfecting its definition of the value-function for the environment.

Q-Learning is a model-free *off-policy* value-based algorithm. This method learns the optimal policy based on the experience gathered, regardless of what decisions the agent makes. In this method, a Q-table, the action-value function *Q(s, a)*, defines the maximum expected reward from picking an action in a particular state [18]. This table is initialized with zero values for each state-action pair. These values are updated as the agent explores the environment and observes the rewards received. This update is done resorting to the Temporal Difference error (TD) 2.7, and according to the Bellman equation 2.8, where α represents the learning rate.

$$\delta = R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t) \tag{2.7}$$

$$NewQ(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{2.8}$$

As the equation 2.8 shows, the agent does not update its policy but only its estimations of *Q(s,a)*. Given an optimal Q-value function, $Q^*(s,a)$, the ideal policy $\pi^*(s)$ can be derived as in 2.9.

$$\pi^*(s) = argmax(Q^*(s,a)), a \in A \tag{2.9}$$

When choosing the action, it is usual for the agent to follow an **Epsilon-Greedy** approach, in which ε dictates the probability of choosing a random action over the current best alternative. This probability decreases as time goes on to incentivize the agent to change from an explorative behaviour (testing out the possible actions and studying their feedback) to an exploitive behaviour, taking advantage of the knowledge acquired to maximize the rewards.

The **SARSA** algorithm is the *on-policy* version of Q-learning. The main distinction is that the Temporal Difference aspect takes into account the action to be taken on the resulting new state using a greedy method. This can be seen in the following equation 2.10.

$$NewQ(S_t,A_t) = Q(S_t,A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1},A_{t+1}) - Q(S_t,A_t)] \tag{2.10}$$

With respect to the agent's environment knowledge, it is essential to point out that faithful representations of real-world scenarios can lead to highly complex problems. This may raise the need to use high-dimensional states, which require more computational power to handle. Managing large dimension state and actions spaces by resorting to tabular methods can become very inefficient and, most importantly, ineffective. These methods construe every state as a unique example while ignoring any similarity to other states, making it hard to generalize and consequently learn. Furthermore, this type of problems introduce the necessity of **sample efficiency** - making the most out of all the gathered information. For these reasons, the scientific community has turned to **Deep Reinforcement Learning** to solve problems with continuous action and state spaces.

2.1.2 Deep Reinforcement Learning

Even though RL approaches have shown promising results when solving several sequential decision problems, the trial-and-error process to maximize rewards and minimize punishments has revealed inadequate in more complicated scenarios. Manual efforts such as Reward Engineering to favour some behaviours over others have given place to more sophisticated Deep Reinforcement Learning approaches. These methods allow the processing of much more information-rich inputs and the extraction of meaningful features to draw better conclusions.

As universal function approximation algorithms, **Neural Networks** (NNs) occupy an important position in Machine Learning and consequently Deep Reinforcement Learning. Although the use of neural networks can open the doors to solving more complex problems, they also introduce a plethora of new challenges. As opposed to traditional techniques, their implementation must consider their architecture, hyperparameters, and the substance and quality of their inputs.

2.1.2.1 Deep Q-Network (DQN)

Deep Q-Network (DQN) [28] is a value-based approach that combines the concepts of NNs with Q-learning. As stated previously, a tabular method would be ineffective to map a vast number of states to a vast number of actions, thus the DQN method attempts to approximate the *Q(s,a)* value function using neural networks capable of generalization. A prediction network is introduced to act as a substitute for a "traditional" stored Q-table, outputting the Q-values of each action for a given state. Although, advantageous, this change brought up some issues. One of which is that, in order to update Q, the target used depends on its current definition. This fact introduces instability in the optimization as it "chaises its own tail" when minimizing the TD error, never actually converging and therefore learning.

With this in mind, DQN introduces a target network - a separate network whose function is to predict the target value. Given a network P that approximates the Q-function and is parameterized by θ, $Q(s,a;\theta)$, and a target network T parameterized by θ^-, $\hat{Q}(s,a;\theta^-)$, the loss function takes into account the predictions of both networks. The neural network predicting $Q(s,a;\theta)$ is repeatedly updated through *backpropagation* in order to minimize this loss, described by the equation 2.11. Each transition used to compute the loss includes an initial state s, the action taken a, the reward obtained r, and the consequential next state s', additionally the calculation includes the value of the action to be taken in the next state according to the target network.

$$L(\theta) = E[r + \gamma \cdot max(\hat{Q}(s',a';\theta^-)) - Q(s,a)] \tag{2.11}$$

The implementation tackles the moving target issue by only updating the target network T every fixed number of updates to the first network P. This correction to the target is often referred to as a "hard-update", since the values of θ are copied onto the parameters θ^-, and keeps the two networks from diverging from each other.

Another instability factor that DQN attempts to mitigate is the high correlation between sequential transitions. When learning certain tasks, such as navigating a maze, the agent can become overfit to certain sets of sequential transitions. For instance, this could translate to the agent only learning how to navigate a few portions of the maze, not being able to generalize to others. To break this correlation, DQN introduces the **experience replay** technique. This is done by managing a replay buffer in which all transitions experienced are stored. When the buffer is sufficiently full, these ($< s,a,r,s' >$ tuples) are randomly sampled, meaning the agent will learn from recent experiences as well as older ones. This experience replay increases the **sample efficiency**, as it minimizes the necessary experience for the desired results [28]. Several improvements have been introduced since the first publication of DQN. One of which is the use of a Prioritized Experience Replay (PER) [36], that, as the name suggests, assigns a priority to transitions and uses it to shape of the probability distribution of actions for sampling instead of doing it randomly. It is worth to mention that, similarly to Q-Learning, DQN uses an **Epsilon-Greedy** approach. As in the tabular method, the probability of exploration is decreased as time goes on. However, this reduction only occurs after each update to the Q-Network. This fact creates a direct relation between the

batch size and the exploration rate. The bigger the batch size, the longer it takes to update the network and decrement ε. For this reason, the number of sampled transitions has a direct effect on the agent's performance. Although this hyperparameter should be adjusted to each particular problem, some studies suggest that using a batch of 512 transitions is a good rule of thumb [40].

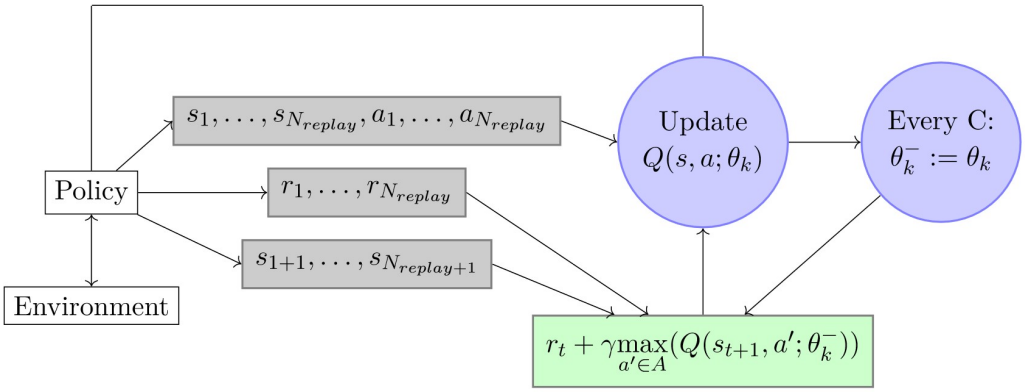

Figure 2.2: DQN components and interactions [13]

2.1.2.2 Policy Gradients

The performance of Reinforcement Learning approaches may be harmed by multiple factors. One of which is that there is a high dependence on the hyperparameter chosen to train the agents. For instance, selecting a learning rate too large, can limit the agent to gather experience under a very poor policy that changes very abruptly. This can make the agent's learning process too instable and, consequently, delay its convergence. On the other hand, besides slowing convergence, choosing a learning rate too little might cause the agent to get stuck on local optima, never really developing the ideal policy. Another important aspect is that using methods like Q-Learning, the agent always develops a deterministic policy. Outside a training context, the agent chooses the best possible action according to the value function approximated during training. When used in competitive scenarios, this certainty can be a substantial disadvantage, since the agent's behaviour becomes predicable, and therefore exploitable by its opponent. In such situations, introducing a stochastic action inference might be the most profitable strategy.

It is also relevant to considered that in order to succeed in certain problems or environments, it might be more effective to directly discover the ideal sequence of actions – approximating the optimal policy $\pi^*(s,a)$ - rather than learning the expected rewards for each station-action pair. With this in mind, researchers have developed Policy Gradient methods. This approach focuses on approximating the ideal policy $\pi^*(s,a)$ through iteratively applying gradient updates to the parameters θ of the current policy π_θ. Ideally, by training an agent using policy gradients methods,

their policy will output a probability close to one for the ideal action a^* for any given state s. Meanwhile, for the same state, the remaining available actions become less likely to be chosen.

Unlike Deep Q-learning, which learns by sampling from all of the past experience, these methods resort to Online learning. Instead of storing all the past transitions in a replay buffer, these transitions are used to update the policy before being discarded. When applying policy gradient methods, it is important to keep in mind that this single use of transitions substantially reduces the sample efficiency.

To update the policy, one must first define the objective function which will guide the correction to be applied. After computing the gradient of this function with respect to the policy parameters, θ, the policy is updated by gradient ascent. In stochastic approaches to policy gradient, such as REINFORCE [], the objective function depends on the rewards obtained from each state until the end state and is defined as in 2.13. For simplicity, τ can be used to represent the transition starting in s by taking the action a, thus $R(s,a)$ and $R(\tau)$ are equivalent.

$$J(\pi_\theta) = E_{\pi_\theta}[\sum_{t=0}^{T-1} R(s_t, a_t)] = E_{\pi_\theta}[R(\tau)] \tag{2.12}$$

When working with a discrete action space, one can define the expected value of choosing an action based on the probability of the policy selecting said action and the return it yields. With this in mind, the equation 2.13 describes the gradient of the objective function.

$$\nabla J(\pi_\theta) = \nabla E_{\pi_\theta}[R(\tau)] = \nabla_\theta \sum_\tau P(\tau|\theta) \cdot R(\tau) = E_{\pi_\theta}[\nabla_\theta log P(\tau|\theta) \cdot R(\tau)] \tag{2.13}$$

After computing the gradient of $J(\pi_\theta)$, the policy parameters are updated using gradient ascent as follows 2.14.

$$\theta = \theta + \alpha \cdot \nabla J(\theta) \tag{2.14}$$

Actor-critic (AC) methods follow the policy gradient approach while also resorting to Temporal Difference. They also employ a strategy similar to Generative Adversarial Networks (GANs), where a generator outputs fake images subject to the evaluation of a discriminator. The discriminator states whether it "believes" the images are real. As more feedback is provided, the generator improves until the discriminator is no longer able to distinguish real images from fake images.In Actor-Critic methods, employs an actor network which decides what action to take, and a Critic network expresses how valuable the transition must be. This feedback is then used to tune the actor network. Unlike GANs, in AC both the actor and critic evolve over time. This results in an actor that predicts better actions for any given state and a critic capable of providing a more insightful evaluation.

In the case of Actor-Critic techniques, such as the Advantage Actor-Critic (A2C) [42], the loss is defined as the expectation over the *log* of the action probabilities given by π multiplied by

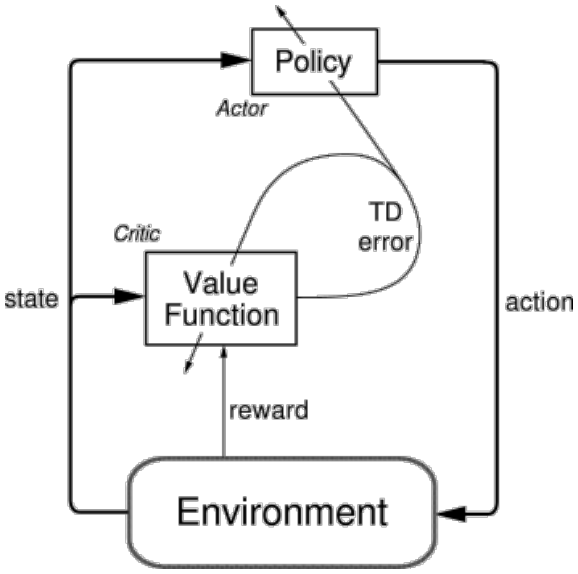

Figure 2.3: Actor-Critic architecture [42]

and *Advantage* function $\hat{A}_t$, as shown in 2.15.

$$L(\theta) = \hat{E}_t[log\pi_\theta(a_t|s_t) \cdot \hat{A}_t] \tag{2.15}$$

The purpose of the advantage function is to compute the difference between a baseline of the expected value of taking an action (the value function) relative to its actual value, the sum of discounted rewards. As such, the advantage is described by the equation 2.16. By feeding into a neural network the state of each transition and the corresponding reward, the agent approximates the value function *V(s)* with estimates the discounted rewards starting from *s*. This approximation follows the usual flow of a supervised learning problem.

$$A^\pi(s,a) \quad = \quad Q^\pi(s,a) - V^\pi(s) \tag{2.16}$$

It is relevant to consider that since this is estimation is the output of a neural network, this "vanilla" policy gradient approach provides a very noisy estimate of *V(s)*. Nevertheless, this approximation allows us to measure how far the current prediction of the value function was from the reward the agent actually got. If this reward was higher than expected, then the advantage will be positive. The objective function as defined by 2.15 leads to a positive gradient when the advantage is also positive, consequently increasing the probabilities of choosing the advantageous action in question. Naturally, the opposite effect occurs for negative advantages.

To address the noisy predictions and prevent the network from getting updated too erratically, the Trust Region Policy Optimization algorithm adapts the objective function to consider the comparison between the updated and the old policy. By using KL-divergence to measure the discrepancy between the two probability distributions and limiting this difference to a small value

parameterized by δ.

$$maximize_{\theta}\hat{E}_t[\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)}\cdot\hat{A}_t] \tag{2.17}$$

Although this improvement prevents the network update from overcorrecting, it introduces a significant overhead in the operation. For this reason, Proximal Policy Optimization follows the same strategy with a slight improvement [37]. It introduces the $r_t(\theta)$ ratio between $\pi_{\theta}(a_t|s_t)$ and $\pi_{\theta old}(a_t|s_t)$ which is bounded between 0 and 1. It also shapes the objective function differently (2.18). The use of the clip operator prevents the change in the policy to fall out of the $[1-\varepsilon, 1+\varepsilon]$ interval, where ε is a hyper-parameter previously established.

$$maximize_{\theta}E_t[min(r_t(\theta)A(s_t,a_t), clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)A(s_t,a_t))] \tag{2.18}$$

The **Asynchronous Advantage Actor-Critic (A3C)** [27] is a technique that introduces the concept of **parallel training**. This algorithm uses multiple agents, each interacting with a copy of the environment. The results of these interactions are asynchronously sent to a global network that learns from all the experience gathered. As the name implies, the advantage function is used to update the global network parameters. Although DQN relies on the replay buffer to sample old transitions for faster convergence, this technique is an improvement since it uses more current knowledge while still profiting from the decorrelation benefits.

Deep Deterministic Policy Gradient (DDPG) [24] is also a relevant DRL Actor-Critic algorithm. This off-policy model-free approach combines the Deep Policy Gradient and Deep Q-Network methods to simultaneously find an optimal deterministic policy and action-value function. It also extends the use of DQN to problems with continuous action spaces, making it extremely useful in fields such as robotics, where problems usually depend on applying continuous voltages to electric actuators. As mentioned, DDPG employs features from DQN, namely the use of target networks. In fact, as the agent trains, it maintains and updates four networks: one actor, one critic, one target actor and one target critic. However, for simpler discrete action space problems, simpler architectures can be used. For instance, one can use a single network implementation, that uses the lower layers to learn features of the surrounding environment, and branches into two different sets of upper-layers, one for the action prediction and one for the critic evaluation.

A particular aspect of the actor network in DDPG is that it outputs the continuous action values as opposed to action probabilities. As it develops a deterministic policy, for each possible state, this actor network will always output the same value. Another similarity to DQN is the presence of the exploration-exploitation dilemma. However, instead of relying on a decreasing ε value, the exploration is achieved by applying some noise to the actor network outputs, which will provide the agent some unexpected transitions. This technique will enrich the replay buffer with experiences that could not be obtained otherwise, therefore allowing the agent to learn new insights into the environment dynamics. Although the original implementation utilizes Ornstein–Uhlenbeck noise, which is a model of Gaussian processes and physical system, it increases the complexity

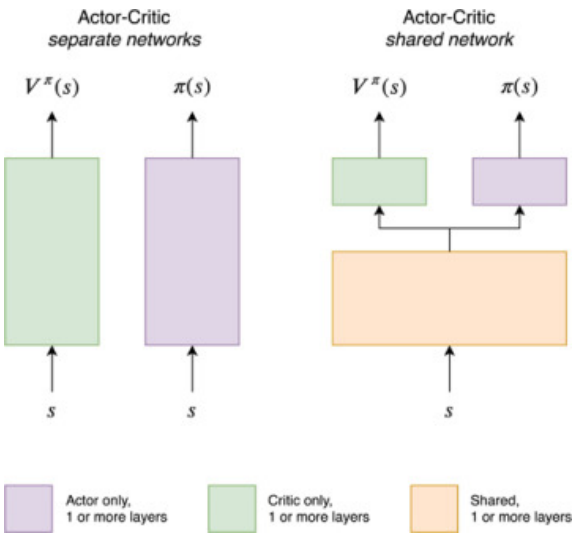

Figure 2.4: Different Actor-Critic architectures: using distinct neural networks vs using a single network [20]

of the implementation. With this in mind, a sufficiently similar effect can be obtained by using Gaussian noise. As seen in DQN, the agent interacts with the environment to collects transitions. The actions are chosen according to the current state of the policy at the corresponding time step. The resulting transitions are stored to later be used to update the policy that will likely have a different parameterization, making this an off-policy algorithm. "Because DDPG is an off-policy algorithm, the replay buffer can be large, allowing the algorithm to benefit from learning across a set of uncorrelated transitions" [24]. As soon as there are enough transitions to sample, it is possible to update the networks.

The actor network, here referred to as μ, is updated according to the direction stated by the gradient of the objective function with respect to the policy parameters θ (2.19). Here θ^{μ} refers to the parameter vector of the actor network, similarly, Q and θ^{Q} refer to the critic network and its parameters.

$$\nabla_{\theta^{\mu}} J \approx E_{s_t \rho^{\beta}}[\nabla_{\theta^{\mu}} Q(s,a|\theta^{Q})|s = s_t, a = \mu(s|\theta^{\mu})] \tag{2.19}$$

As the goal is to simultaneously improve the critic network to make better value estimations, DDPG minimizes the mean squared error loss function described in 2.20. The target y_i is computed by feeding the ending states of the sampled transitions to the target actor network μ' and using the target critic network Q' to predict the value of these potential transitions. The estimations of Q' is then multiplied by the discount factor γ and added to the actual reward r_i obtained by the transition.

$$\begin{aligned} L &= \frac{1}{N} \cdot \sum_i (y_i - Q(s_i, a_i|\theta^{Q}))^2 \\ y_i &= r_i + \gamma \cdot Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'}) \end{aligned} \tag{2.20}$$

After the corrections to both the actor and critic networks, the target networks μ' and Q' go through a "soft update". This requires the parameters of μ and Q to be copied onto their targets subject to an attenuating factor τ (bounded between 0 and 1) (2.21).

$$\begin{aligned} \theta^{\mu'} &\leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'} \\ \theta^{Q'} &=\leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'} \end{aligned} \tag{2.21}$$

2.1.3 Multi-Agent Deep Reinforcement Learning

Multi-Agent Deep Reinforcement Learning (MADRL) introduces the advantages of DRL to solve problems in which multiple agents must cooperate or compete to achieve a particular goal [7]. These agents try to learn a policy to maximize their own long-term personal returns and the collective, cumulative rewards. This field also raises some new obstacles that researchers need to face, specifically, the fact that with each decision, every agent changes the state of the shared environment, making it harder to understand the dynamics at play and possibly learn what decisions to make.

2.1.3.1 Independent Learners

The simplest Reinforcement Learning approach to Multiagent setting, is to implement Independent Learners. This consists in simply developing independent agents that improve their own policy or value-function, in the case of Independent Q-learning (IQL), in a Decentralized Partially observable environment (Dec-POMDP), while not addressing the multiple agent-to-environment nor the agent-to-agent interactions [26]. Past work has shown that although tabular IQL tends to be a practical and successful tool for small size problems, this is usually not the case for DQN adaptations [26].

From the perspective of every learner, the state now consists of both the shared environment and the internal state of the remaining individuals. In most real-world problems, it is impossible to collect all of this information before each step. This points towards a stochastic game or Markov Game. Here, the Markov assumption becomes invalid unless communication processes are put into action.

These situations may render Q-learning ineffective due to the fact that the environment becomes non-stationary once multiple agents are introduced. In single agent approaches to multiagent problems, this happens because all agents are evolving and interacting simultaneously, and from the perspective of each agent, these third parties are just part of the surroundings [30]. From the point of view of each agent, the actions of other agents create transitions between states even if the agent is idle. The state transitions result from the joint-action of all agents. This makes the environment dynamics much harder to learn and navigate. Introducing parallel training makes these "incomprehensible" changes even more difficult to predict. As the remaining learners are also undergoing constant corrections, they behave in a very consistent manner before their policies eventually converge.

As in many MARL and MADRL problems, the single agent reward may depend on the joint-action of all agents involved. This causes an increase in the variability of the reward signal, exacerbating the high-variance to which Policy Gradients are prone. This high-variance is usually associated with the so-called **Reward Attribution Problem**. Especially in sparse reward settings, where the agent rarely get a reward unless they successfully achieve the goal, it is hard for the agent to compute which of the previous actions most contributed to the reward obtained. The introduction of multiple agents in the same environment interferes with this extrapolation, since the policies are being updated solely considering the agents of one agent while the rewards take every action into account. The Proposition 1 in [25] attempts to mathematically suggest this idea:

"Consider N agents with binary actions: $P(a_i = 1) = \theta_i$, where $R(a_1, \ldots, a_N) = 1_{a_1=\cdots=a_N}$. We assume an uninformed scenario, in which agents are initialized to $\theta_i = 0.5 \forall_i$. Then, if we are estimating the gradient of the cost J with policy gradient, we have:

$$P(\langle \hat{\nabla} J, \nabla J \rangle > 0) \propto (0.5)^N \tag{2.22}$$

where $\hat{\nabla} J$ is the policy gradient estimator from a single sample, and ∇J is the true gradient."

The equation 2.22, shows that as the number of involved agents grows, the probability of updating the gradients in the ideal direction decreases, therefore harming the learning process.

Although, full observability and unconstrained communication are not usually possible in real-time problems. One could shape the local observations of agents to include some additional information shared among learners. This communication could be in the form of sharing what, or part of what, agents can observe, and/or the actions they intend to perform.

The agents in MARL problems may be homogeneous or heterogeneous, depending on whether they have the same kind of observations and available actions. In situations in which all agents are homogeneous, one can rely on parameter-sharing to more efficiently address the problem. Parameter-sharing consists on only optimizing one policy or value function that is used to infer the actions of every agent based on their local observation. For instance, if all the N agents must have the same behaviour, instead of training several actor-critic agents (which can require managing up to 4 networks each), one can use the experience of all agents to train one set of networks. This approach also introduces variability to the training process, because the network updates rely on transitions that are acquired by interacting with the environment from different perspectives.

2.1.3.2 Fully Observable Critic

There are many architectures one could adapt to tackle MADRL. For instance, while some approaches may be fully centralized – a single entity optimizes a policy and chooses what each agent does – others can be decentralized - each individual develops its own policy – which is useful for coexisting agents with distinct goals and tasks. The Multi-Agent Deep deterministic Policy Gradient (MADDPG) is one of the main techniques proposed. As the name suggests, this is a Multi-Agent Actor-Critic method based on DDPG. The focus of this algorithm is to apply

centralized training together with decentralized execution, this way different agents can approximate their own policies resorting to local and global observations. Agents in MADDPG can approximate very different policies making this an appropriate strategy to tackle environments that require, cooperation, competitions, or even both. The architecture can be seen in figure 2.5, each critic deals with global information as the observations and actions of all agents are fed into the critics. On the other hand, during execution, only the local observations serve as input to the agent's policy networks.

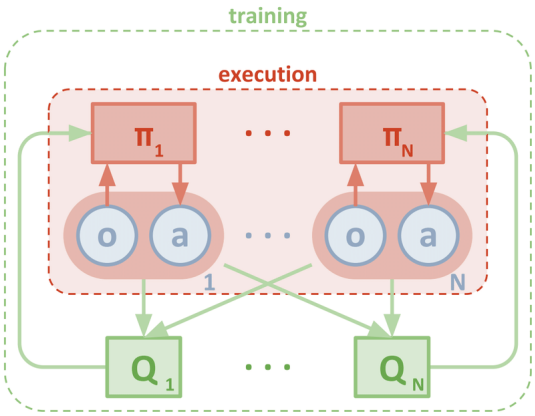

Figure 2.5: MADDPG architechture [25]

As the gradient for DDPG can also be written as in 2.23, the gradient for MADDPG follows a similar strategy 2.23. The main difference is the usage of the centralized action-value function Q_i^{π} for each agent. As the equation suggests, the actor network μ receives as input the actions a_i and the agent's local observations o_i. On the other hand, the critic network receives the full set of observations $x = (o_1, \ldots, o_N)$, as well as all of the actions of each agent $(a_1, \ldots, a_N)$

$$\nabla_{\theta_i} J(\mu_i) \approx E_{x,a\setminus D}[\nabla_{\theta^{\mu}} Q(s,a|\theta^{Q})|s = s_t, a = \mu(s|\theta^{\mu})] \tag{2.23}$$

Although this is not a very scalable solution – with each agent, the networks' input also increases for all agent – it still outperforms most RL techniques [25], proving its value in a lot of different scenarios.

2.1.3.3 Coordination, Cooperation, and Competition

As stated in [47], "*coordination is a property of a system of agents performing some activity in a shared environment. The degree of coordination is the extent to which they avoid extraneous activity by reducing resource contention, avoiding livelock and deadlock, and maintaining applicable safety conditions*". Coordination problems can be roughly divided between Cooperative and Competitive, although real-life problems usually.

As demonstrated, MADRL problems can assume very different shapes and therefore call for different strategies. One of the main issues in Cooperation and Competition problems is how to distinguish the importance of each agent's participation in the final outcome. This is known as the Credit Assignment Problem.

This contribution measure became the incentive to multiple coordination approaches, namely the Value Decomposition Networks (VDNs) [41]. As the name implies, the focus of VDN is to decompose the impact that each agent has on the collective rewards. With this in mind, VDN assumes that the joint value function can be represented as the sum of each contribution, as stated by the value function of each agent. This representation is illustrated by the equation 2.24, where τ is a joint-action-observation and u . This kind of methods tries to mitigate the so-called "lazy agents"

$$Q_{tot}(\tau, u) = \sum_{i=1}^{n} Q_i(\tau^i, u^i; \theta^i) \tag{2.24}$$

By adding some restrictions, QMIX has been proposed as an improvement to VDN [34]. To ensure a monotonic improvement, QMIX enforces that the gradient of the global joint action value functions with respect to each local value function to be positive as shown in 2.25.

$$\frac{\partial Q_tot}{\partial Q_a} \geq 0, \forall a \in A \tag{2.25}$$

Another aspect of QMIX is that, similarly to MADDPG, it relies on centralized training and decentralized exception. The architecture, shown in 2.6, shows that each agent has their own Q-function, evaluating local observations and actions. Additionally, it introduces a centralized Mixing Network, which receives as input the joint observations and actions of all agents. Further research and improvements on QMIX gave rise to QTRAN, which has been show to be useful in a larger number of tasks without as many constraints for monotonicity [39].

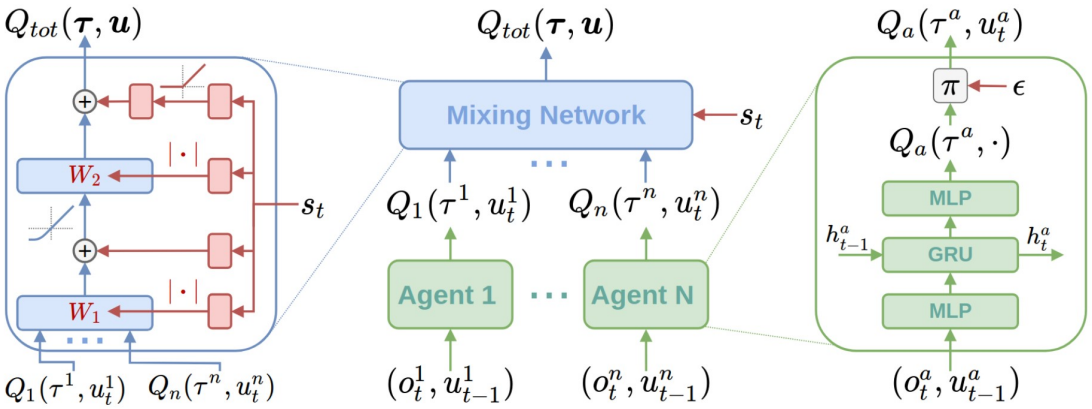

Figure 2.6: QMIX architecture [34]

Besides computing individual contribution, it is also possible to achieve coordination by making decisions that take the behaviours of others into account. Some researchers have used Bayesian Action Decoder [11], or Simplified Action Decoder [16] methods to infer information by analysing other agents' behaviours.

2.2 Satellite Constellation Problems

The evolution in EOS systems has proven advantageous in multiple sectors by extending the information available for a diverse set of processes. Although valuable, these advances have raised a vast number of complex problems that need to be addressed if EOS constellations are to be considered beneficial. These issues range from how the main entities are organized to how they interact and behave.

As discussed, the Multi-Agent System paradigm has been helpful to study an extensive spectrum of technological environments – EOS constellations are no exception. In this field, several approaches have proposed ways to verify these systems' commitment to demanding performance [2] and safety prerequisites [35]. These analyses are made possible by carefully modelling the central entities – including satellites, ground stations, and relay communication satellites – and simulating the processes involved. These representations are essential to represent the roles that each entity plays as well as their interests, and goal [48]. The problem of deciding what agents to include in a cooperative team as well as their role inside such organizations is a recurrent challenge in MAS [1]. The problem of Constellation Design consists in finding the optimal constellation architecture to satisfy a comprehensive set of constraints without compromising the observations, and the goals of stakeholders [19]. Being an optimization problem, researchers have tackled this issue by resorting to iterative improvement algorithms such as genetic algorithms [9] and swarm optimization algorithms [46]. MAS methods allow evaluating how agents act and interact with each other. For this reason, there is also a need to reconsider the communication processes among entities. With significant amounts of information flowing in EOS systems, authors have tried to evaluate what information needs to be shared, with which entities, and under which conditions. Although multiple factors also condition communication among agents, its improvements are essential to the cooperative behaviour of satellites. With this in mind, some solutions have tried to achieve better results using various communication protocols and negotiation [4]. Other authors have recognized that besides aiding cooperation during image acquisition, enhancing the exchange of information can improve the response time necessary to return results to consumers [6].

However, solving communication problems serves only as a tool to respond to more challenging issues in EOS constellations, namely resource allocation and task scheduling. Earth Observation Satellites need resources of different kinds to conduct their measurements. Some of these are relevant only for each agent (as is the case of on-board memory and computational power), while others concern multiple agents simultaneously. Communication introduces the possibility of using intricate protocols that aim to define the resources each stakeholder has access to. Some of the solutions proposed include scalable architectures that allow negotiation through auctions.

These negotiations take into account the characterization of resources (divisible, sharable, static, or perishable) [8].

With respect to resource division, some authors have also proposed solutions to better define and achieve Fair Allocation [5].

Finally, concerning observation scheduling, it is possible to point out a few more issues. Firstly, it is relevant to study how to express satellites' preferences for some observations over others. These may differ due to resources, sensor capabilities, climate conditions, or even the priority specified by whoever requested the observation [44]. Dealing with the intentions of so many entities is not a trivial task. Consensus Optimization may be an instrument to defuse conflicts of interest among stakeholders [29]. Moreover, observation scheduling problems in EOS systems include dealing with the unpredictable consumption of resources by observation and unexpected adverse weather conditions that can limit the visibility of some areas of the globe. For these reasons, systems should be able to plane adaptively [45].

2.3 Related Work

As previously explained, this work will focus on the resource management problems described, and for this reason, this literature review features a survey of the past work done.

Some implementations tackle this issue as a task scheduling problem with resource constraints. To find scheduling solutions, some authors relied on traditional optimization techniques such as linear programming. The optimal solutions are then found, resorting to Branch and Bound methods, Restriction Programming, and Dynamic programming [22]. In [10], the authors also maximize the value of the satellite resources by optimizing the task scheduling. While relying on the MAS paradigm, this work makes use of Contract Net Protocols to assign the tasks to the most apt satellite.

Similar to the Constellation Design problem, other authors approached this problem employing iterative improvement algorithms, namely Swarm Optimization, Simulated annealing [14], and Ant-Colony Optimization [17].

By taking advantage of Multi-Agent Systems, some proposed solutions include the implementation of agents capable of combining Greedy algorithms (that could improve the performance of each agent on its own) with negotiation protocols that could benefit the system as a whole [38].

The combination of MAS with DRL methods was already employed by some research teams. The architecture chosen for the MAS has an impact in every aspect of the research, from the information available to each agent to the communication protocols established. Some researchers opted for fully decentralized systems in which agents cannot communicate but whose experience was shared during the training phase; this resulted in agents that did not depend on communication (which, as discussed, can be faulty) but that took less time to train [23].

Another proposed architecture for a MADRL system included the concept of different agent roles. In this solution, one agent, considered more powerful and knowledgeable than the remaining, would have access to a large amount of information and gradually choose what information

to disclose. This new information would weigh in on the decisions of the rest of the agents to hopefully improve their performance [15].

The work developed in [31] is of special relevance to this study. Besides revolving around the cooperative behaviour of satellites in a resource management problem, it does so by using DRL. It also offers the comparison of three value-based approaches, namely Q-Learning, Deep Q-Learning and Double Q-Learning.

In [21], the authors attempt to tackle the problem of finding an optimal allocation of resources in satellite constellations. While doing so, this work presents an insightful discussion on how to define the concept of utility given the different interest of the stakeholders involved as well as the fairness of allocation with respect to their investments in the system.

Although it does not aim to promote collaborative behaviour, the authors of [49] offer an interesting study on the use of image compression techniques to maximize the number of observations stored in memory.

2.4 Development Tools

Multiple tools may be helpful in the development of RL solutions. To choose between them, it is important to evaluate if they are suitable for the problem one wants to solve. Other factors such as the compatibility with other tools or even community support can, and should, influence this decision.

OpenAI Gym is a library that provides a series of tools to develop Reinforcement Learning models. It provides a standard way for authors to implement, evaluate and compare algorithms. Even though this tool allows working with different types of environments, each tailored for a specific kind of problem, one significant aspect is that it also enables the implementation of custom environments. Relying on an interface facilitates working with different environments since the interactions are always made the same way. This library also includes implementations of some essential RL and DRL algorithms provided by the TensorFlow and PyTorch libraries.

Other noteworthy RL libraries and frameworks include TensorTrade, Unity ML-Agents, for single and Multi-Agent game-like environments, and DeepMind Control Suite for physics-based problems.

Since the implementation will be done using Python, libraries such as NumPy and Pandas also provide efficient data structures and operations to handle information efficiently.

Finally, modelling and simulating the satellites' surroundings and dynamics is not a trivial task. For this purpose, it is possible to use technologies like Basilisk or the SSA-Gym. These provide detailed representations of the entities involved in aero spatial settings facilitating this part of the solution. Most importantly, both of these options are compatible with OpenAI Gym, making them good candidates for the final prototype.

Chapter 3

Proposed Solution

The aim of this dissertation is to try to approach this cooperative resource management problem of EOSs as a Multi-Agent Deep Reinforcement Learning problem. To provide such a study, one can rely on a multitude of strategies, formalizations, and general parameter combinations. Moreover, the weight given to such factors and the multiple decisions made, ultimately shape the implementation and therefore the results obtained. Rather than trying to make an exhaustive exploration and comparison of such a vast space of combinations, this work focuses on two particular aspects of the formulation, reward and observation, and how their shape affects the agents' learning process. As such, it was first necessary to provide a formalization of the problem as a *Dec-POMDP*.

This formalization should include what relevant information is available in the agents' observations, as well as the action-space from which action can be picked. Although, the transitions might not be practical to define mathematically, it was important to define how the environment would react to each action. These descriptions would work as guidelines to the development of the solution. Afterwords. the implementation proposed truthfully represented the main concepts, entities and inner mechanisms of the problem's environment and agent.

The problem created, tries to include some cooperation with respect to resource management, present in the context of EOS systems. As the area of study is MADRL, the problem tries to guarantee the necessary conditions to train autonomous agents. These conditions were chosen in spite of ignoring some aspects that would make the solution a viable product, and therefore able to integrate in the EOS workflow. Even so, this is done in the hopes of establishing the solid foundations that will allow more complex and sophisticated solutions to be applied for this application.

3.1 Problem Description

This definition assumes that the satellites are positioned in an immutable orbit whose trajectory the agents are unable to control or choose beforehand. Throughout their orbit, the satellites are able to perform several target observations. The agents are unaware of how many target observations they will encounter before returning to the beginning of the orbit. As in a real life scenario, the

acquisition of Earth images implies an investment of resources, for simplicity's sake, this problem will only consider that capturing an image only affects the memory.

Another crucial aspect of these constellations of satellites is their ability to transmit the data that is collected back to Earth. These transmissions are done by establishing a communication between the satellites and the ground stations and are generally referred to as **Downlink** operations. In this problem, a single ground station will occasionally be accessible to satellites as they get physically closer. This relationship between ground stations and satellites is not only beneficial because it gets the information to its consumers, but also because it makes it possible for the satellites to regain the resources necessary to capture new images. Transferring data between satellites is also a possibility, and could be used to their advantage, as it liberates the resources of a satellite while limiting the other. However, this transmission of data is limited by the available resources at the receiving end – only a satellite with free memory can receive the images captured by another agent.

3.1.1 Observations

The satellites have a very limited scope, meaning they can only gather information about what is closest to them at each time step of the MDP problem. As they move through their orbit, the agents must take in some of this data before deciding what to do. Although the exact content and granularity of the observations will be discussed in the methodology section, the information available to compose the agent's observations is described in 3.1:

$$Obs = \langle c, d, a, s, g \rangle \tag{3.1}$$

Where:

- c represents a cost with $c \in \{-1, 0, 1\}$, where -1 means that the satellite is able to downlink information to the ground station, 0 implies a neutral impact in resources, and 1 represents the cost in memory associated with an image captured.
- a defines what available resources does the satellite have at its disposal, where $a \in \mathbb{Z}^+$ and $0 \leq a \leq 10$;
- s states what available resources do the other satellites have at their disposal, where $s \in \mathbb{Z}^+$ and $0 \leq s \leq 10$;
- g is a boolean variable expressing whether or not the satellite is close to a ground station, where $g \in \{0, 1\}$.

3.1.2 Action Space

An important aspect of the problem formalisation is the definition of the action-space A, which is set defining the operators available for the agents to perform the actions. In this case, A is defines in the expression below 3.2.

$$A = \{a_1, a_2, a_3, a_4\} \tag{3.2}$$

Where:

- a_1 results in capturing an image;
- a_2 causes the satellite to downlink all images to a ground station;
- a_3 results in the transition of an image to a nearby satellite;
- a_4 is a typical dummy *noOp* operator, meaning the agent does not act.

The goal of the agents is to continue to capture as many images as they can, but they can only do so as long as there are still available resources in the constellation to respond to observation requests. Bearing this in mind, each agent must acquire as many images as possible while refraining from ever reaching a state of resource starvation. This objective requires a careful consumption and allocation of the satellites' resources. Such resource management calls for a good coordination between the agents, as all of them need to acquire images without jeopardizing the mutual good. In figure 3.1 it is possible to see the cycle that each agent goes through at each time step of each episode.

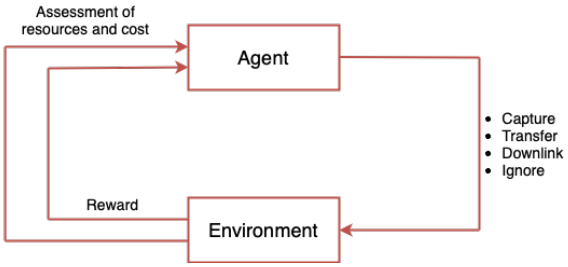

Figure 3.1: EOS Agent-environment interaction

3.1.3 Problem Assumptions

Given that the study focuses on the domain of MADRL, some assumptions were made so as to allow the implementation to focus on the real aspects that integrate the problem definition. Although the implementation welcomes new modifications, it assures the realism within the following boundaries:

- **Immutable orbits and satellite relative position**: since the satellite orbits do not play a role in the problem formulation, the study considers that the satellites are unable to change course and that all agent travel in parallel, always keeping the same distance between them.
- **Flawless communication**: although this could be introduced as a way to improve the learning process, the communication between satellites and with the ground stations is considered flawless. The transmission of images between agents may be constrained by resources, but not by noise in the communication.
- **Resource consumption**: one of the biggest simplifications made was the assumption that the only resource the satellite has to manage is memory. In reality, there is a variety of resources involved with operating a satellite, fuel among others. Similarly, time is also considered as an unlimited resource, as capturing each image requires no overhead and is seen as an instantaneous action. The reason for this decision, is that it simplifies computation of the cost of each action. Otherwise, this could require a cost function to use a specific weight for each possible resource and to return the total cost of the operations.
- **Identical images**: although this could be easily configured, all the images captured by the agents have the same impact on the memory available. Moreover, every image requires the same sensors to be acquired. Although in real-life situations, some observations can be time sensitive, to provide information about emerging events or conflict regions, in this implementation every image has the same level of priority.

3.2 Methodology

To tackle the problem established in the previous section, the implementation required several steps. This section aims to explore and illustrate these stages, as well as the reasoning behind the decisions taken in each one of them. This process demanded multiple moments of assessment that guided the development and tuning of the environment, the choice of the reinforcement learning models and architectures, and finally, the scenarios to be studied.

The tool used to develop the environment was the *Python* library *OpenAI Gym*. It was chosen since it facilitates the creation of custom environments made specifically for the problem at hand. Moreover, the library requires the implementation of a standard interface, enforcing a universal way of interacting with these environments—this uniformity promotes the inclusion of future contributions. To develop the agents' neural networks, the implementation resorted to the *Keras* Python library. This choice was influenced by the fact that this technology is easy to use and has a lot of community support, thus facilitating the development of the proposed solution.

3.2.1 The Environment

The first stage of the implementation was to design the environment in which the agents would live. The intention was to create a simple environment that met the requirements stated in the problem description.

Firstly, it should faithfully represent the problem setup chosen. This meant that it should include abstract representations of the multiple entities identified in the problem formulation.

Furthermore, it should be suitable to train the agents. As such, it should always react accordingly to the actions taken by each agent. Training in such environment would only be reasonable if given the possibility to collect useful information to evaluate the environments and agents behaviour. Most importantly, this data should allow the creation of the metrics used to evaluate the progression and the results of each strategy used.

Finally, one of the most important requirements is the ability to experiment with different scenarios. Moreover, as will be discussed in the future work section, the implementation should permit easy modifications as to allow further improvements, namely, a more realistic modelling of the real systems.

The implementation started by defining the Satellite object which is described the following attributes (representing the satellite's state):

- **Name** – its unique identification;
- **Position** – its current position in orbit;
- **Current Available Memory** – a quantization of the resources still available to capture observations;
- **Initial Available Memory** – the initial available memory is stored to calculate the ratio of spent resources compared to the beginning of the episode;
- **Access to Ground Station** – boolean value that states whether the satellite is close to a ground station, allowing communication to transfer observation data;

As the inference of actions will be done by satellites, the terms "agent" and "satellites" will be used interchangeably. Additionally, the observations to be performed by the satellites – which are referred to as "Target Observations" or simply "images" to distinguish from the agent's observations in the context of RL – only have two attributes: their position and their cost in terms of resources (more specifically memory). Since the target observations and the ground stations have a passive role in this problem specification, the Target Observation class is used to represent both. The main difference between the two is that the ground stations have a negative memory cost. For this implementation, the satellites had an initial memory equivalent to the cost of **10** images. Given the fact that the methodology followed the assumption that all images require the same resources, their memory cost attribute has the value of **1**.

An important aspect of the environment simplification is that, as the orbit is not a factor in the mathematical formulation of the problem, the satellites travel side by side. In figure 3.2 it is possible to see a simple rendering of the state of the environment. Although it is an oversimplification of a real-life system, this visual representation was useful to observe the agents' behaviour and to shine a light on unpredicted events usually related with implementation errors.

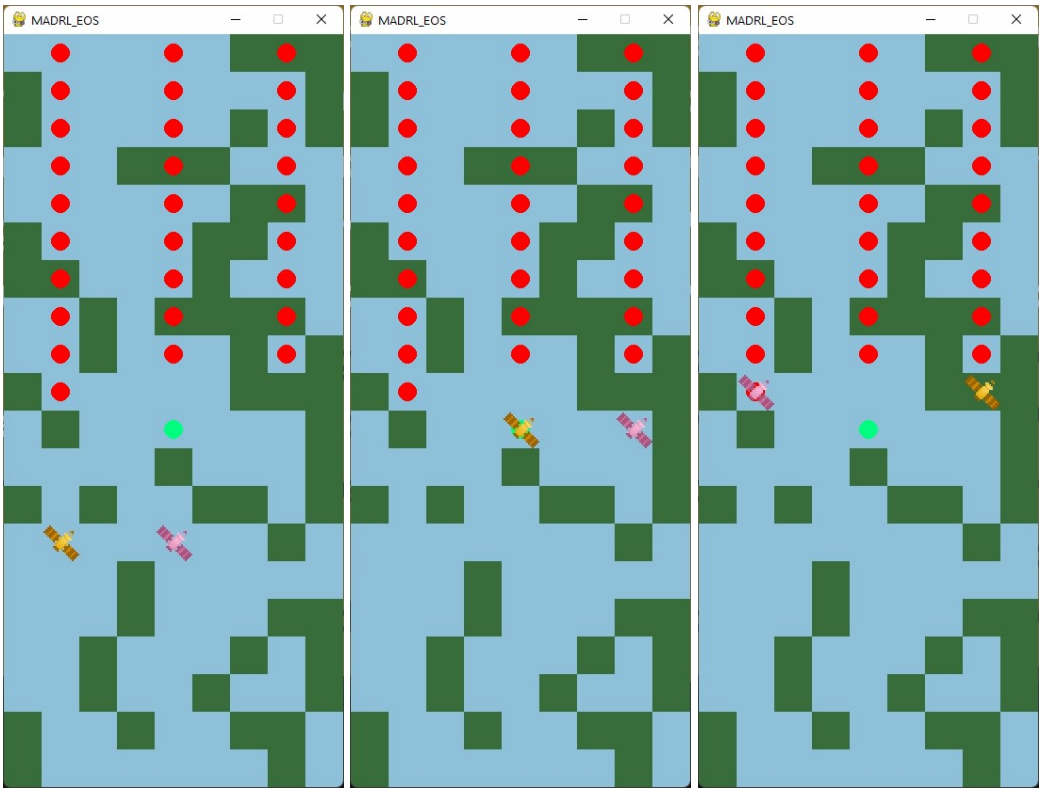

Figure 3.2: The three initial orbits are concatenated to form one long orbit, as satellites reach the end of each segment, they move to the adjacent one.

The environment and its variations have a similar structure. The three satellites' orbits are short and parallel. Throughout these trajectories, an infinite amount of target observations are placed. As the satellite approaches the target observation, it is able to capture it. This means that if everything remains as it is when the satellite revisits the same place in the orbit, this target observation will not be available to be acquired.

The environment has an initial number of observations, which is related to the percentage of steps in orbit in which the agent can find an image to capture. For instance, if this value was set to 70%, and the orbit required 100 time steps to go through, the agent would encounter 70 images before returning to the initial position. At each time step, the agent will only find one image, so they must choose whether to capture an image rather than which one to capture.

Another feature of the environment implementation is that it allows setting a minimum and maximum percentage of the orbit that should be covered by images. As satellites acquire the images, the environment checks if the number of remaining target observations drops below the

established threshold. Whenever this happens, the system proceeds to repopulate the orbits with new tasks.

With respect to the orbits, only the central one provides access to the ground station to which the data can be transmitted. However, for some experiments, the orbits were concatenated to form a single longer orbit containing the ground station around the middle. When visualizing the execution, this results in a "spring effect" caused by satellites transitioning from the end of their orbit to the beginning of the adjacent one. Figure 3.2 attempts to showcase this effect.

Once the main entities and their attributes were appropriately represented, it was possible to address the artificial intelligence aspect of the solution.

3.2.2 Algorithm Implementation

As stated previously, a simple local observation can be composed of the memory cost of the closest target observation, the available resources and whether the satellite is close to a ground station to which it will be able to flush the images. As each variable v_i in the observation has distinct possible values n_{v_i}, the number of possible states in S can become significantly large. This assessment can be done systematically by multiplying all n_{v_i} as in 3.3.

$$|S| = \prod n_{v_i} \tag{3.3}$$

Dealing with large sets of distinct states can benefit from Deep Reinforcement Learning. Trying to map the value between every possible state and every possible action would be an exhaustive task. With this in mind, the use of neural networks makes it possible to deal with a lot more states without the need to store them. Additionally, some architectures are capable of extracting meaningful features, thus acquiring more insightful information such as the similarity between states and their rewards.

Since the action space defined consisted of a discrete set of actions, the intuitive choice for a first approach to this problem was the use of DQN. As a proven solution to a lot of problems, using it seemed fitting for the problem at hand. The implementation achieved made use of most of the main components described in the literature, namely experience replay and target networks with hard updates. To allow the reproduction of the results, Table 3.1 summarizes the parameter used.

The Q-Network and the Target Network were copies of one another, they were initialized with the same set of hyperparameters and architecture: two dense layers with 256 neurons and *ReLU* activation function, followed by an output dense layer with a number of neurons equal to the number of actions.

Besides DQN, this implementation also resorted to an Advantage Actor-Critic method. As explored in the literature review, the network architecture used a shared set of initial fully connected layers and then branches out to two different output layers – one for the actions' probability distributions and one for the value-function. The first fully connected layer had the dimension of 1024 neurons and a *ReLU* activation function. The following fully connected layer had the dimension

Table 3.1: DQN Parameters

Parameter	Value
Replay Buffer Size	1000000
Batch Size	512
Initial Epsilon	1.0
Epsilon Decay (multiplied)	0.999
Minimum Epsilon	0.15
Gamma (Discount Factor)	0.99
Alpha (Learning rate)	0.001
Target Update Rate	10
Optimizer	Adam

of 512 neurons and a *Tahn* activation function. Finally, the value output layer had no activation function and 1 neuron, and the policy layer had a dimension equal to the number of actions used and a *Softmax* activation – to attribute a probability of choosing an action between 0 and 1. Both networks used a learning rate α of 0.0003 and discount factor γ of 0.99.

3.2.3 Reward Functions

As mentioned, the definition of the feedback signal reveals a significant amount of meaningful information that the agent will use to improve their behaviour and understanding of how the environment functions. Three different functions were designed and put to use to study the impact that the shape of the extrinsic reward signal would have on the learning process.

The first reward function developed, R_t^1, awarded agents with a score of 1 for every image acquired. However, once the agent's memory is complete, they are not able to capture any more images. For this reason, the reward function also benefits the agents when they perform the downlink operation. This reward is directionally proportional to the amount of data transferred to the ground station. The equation 3.4 sums up this reward function. The R_t^1 reward function was the one used during the Environment Tuning stage.

$$R_t^1 = \begin{cases} 1 & \text{, if an image was successfully captured} \\ N & \text{, N is equivalent to the amount of data transferred to the ground station} \end{cases} \tag{3.4}$$

The second reward designed, R_t^2, follows the thought process of R_t^1. The only difference is that besides rewarding captures and downlinks, it introduces a penalty of 1 whenever the agent's resources are one capture away from complete. The intention behind this negative reward is to warn the agent to steer away from termination states more explicitly. Ideally, this should be an emergent behaviour, as deep learning models should learn to pick up on specific features to draw this kind of conclusion autonomously.

$$R_t^2 = \begin{cases} 1 & \text{, if an image was successfully captured} \\ N & \text{, N is equivalent to the amount of data transfered to the ground station} \\ -1 & \text{, if an image was captured but the agent is left with 1 image worth of memory} \end{cases} \tag{3.5}$$

Finally, instead of directly imposing a penalty when the agent is close to finishing, R_t^3, described in 3.6, applies a discount to the positive reward. The computation of this discount resorts to a modified sigmoid function. This function was shaped according to the satellites' available resources at the beginning of each episode. Using a sigmoid-like function creates an elastic effect as the discount is almost insignificant for the first images captured but becomes more aggressive as the agent reaches its limit.

$$R_t^3 = \begin{cases} 1 - \frac{5}{1-e^{(10-x)}} & \text{, where x is the remaining memory after capturing an image} \\ N & \text{, N is equivalent to the amount of data transfered to the ground station} \end{cases} \tag{3.6}$$

3.2.4 Indirect Cooperation Settings

The performance of multiple agents in a cooperative setting can be affected by a multitude of intrinsic and extrinsic factors. Besides studying the impact of different reward functions, this implementation included four distinct scenarios that also tampered with the information included in the observations of each agent. Unlike in the single-agent context, the episode was not terminated once the agent ran out of resources. Instead, if some of the agents were still apt to capture more images, the remaining agents could reach a ground station to downlink data and return to their initial state. Although this change seems to facilitate the exploitation of the environment, the goal is to evaluate whether having access to additional data can help agents develop new strategies and converge faster. For this reason, communication between agents is considered in the form of fully or partially sharing their observations.

In the first three scenarios, two satellites are positioned in the different stages of the same orbit. One of the satellites starts in the portion of orbit that is closest to the ground station. As time goes on, both satellites alternate between having access to the ground station or not. The action space is the same as the one available in the environment tuning phase (capture, downlink or ignore).

The differences between the explored scenarios are summarized here:

1. **Independent Learners** - the agents have access only to their local observations, which consist of:
    - The memory cost of acquiring the closest image;
    - The available memory;
    - Whether the satellite is close to a ground station;

2. **Partial Communication** - the agents have access to their local observations as well as part of the other satellite's observation, more precisely, their current available memory. As such, the observation includes:

    - The memory cost of acquiring the closest image;
    - The current available memory;
    - Whether the satellite is close to a ground station;
    - The available memory in peer satellite

3. **Full Communication** - at every time step of the episode, the agents have access to a concatenation of their local observations to the local observations of their peer. In practice, both agents are fully observable as their observations comprise:

    - The memory cost of acquiring the closest image to the agent;
    - The agent's available memory;
    - Whether the agent is close to a ground station;
    - The memory cost of acquiring the closest image to the peer agent;
    - The peer agent's available memory;
    - Whether the peer agent is close to a ground station, always the opposite value of the first agent, since they never have access simultaneously;

When close to a ground station, the first value of the observations changes from 0 or 1, in case there is an image to capture, to -1. This value indicates that an interaction would lead to a negative cost in the agent's memory.

3.2.5 Direct Cooperation Settings

3.2.5.1 Two Heterogeneous Interacting Agents

Although the previous scenarios explore how agents can cooperate by making decisions based on the information they have available, they lack in two aspects. First, they only include homogeneous agents, all of the agents have the same types of observations and the same set of possible actions to choose from. Another characteristic of the scenarios is that the action-space only includes actions capable of affecting the agent's own state and consequently influencing how soon the episode terminates. However, none of the agent's actions have a direct impact on the other agents in the environment.

With these limitations in mind, two satellites were positioned in two separate, immutable orbits. However, only one of the orbits allowed the satellite to downlink information. As part of the focus was to use heterogeneous agents, two action spaces were defined for the two learners: the satellite with access to the ground station could capture images and downlink them to the ground station, the other satellite could only capture the images and transfer them to its peer.

By transferring data to another satellite, the agent is freeing its own resources while limiting the other's. The transferring agent is able to extrapolate the consequences of this action, since this experiment uses the observations used in the partial communication scenario. The amount of data transferred is the minimum value between the volume of data collected by the sending satellite and the available memory space on the receiving agent.

3.2.5.2 Three Homogeneous Interacting Agents

Although the focus of this work is not the scalability of the approaches used, in this scenario, a third satellite is introduced. As such, each of the satellites was positioned at the beginning of one of the three segments. All agents move simultaneously, and have access to the same action-space and observations. In this scenario, the agents are constantly aware of the resources available in the other two satellites. The discrete set of actions available includes:

- Capturing an image;
- Downlinking all images to the ground station;
- Transmitting images to the satellite on their left;
- Transmitting images to the satellite on their right;
- Ignoring;

This constellation of homogeneous agents moves along a single orbit, giving each satellite a chance to downlink information to the ground station. The expectation is that, as this set of conditions facilitates the opportunity to free resources, the agents will not be incentivized to share images. Instead, they can simply wait for their turn to downlink before capturing more target observations.

3.2.5.3 Three Heterogeneous Interacting Agents

In the previous scenario, the expectation is that the agents will only choose between when to capture and when to downlink. This anticipation suggests that a subset of the action space will become obsolete. As a way to impel the agents to take actions that affect their peers, the agents were put back into separate, shorter orbits as in 3.2.5.1. This constraint meant that only one agent had access to the ground station, and that the remaining satellites had to resort to the transmission of images to liberate memory space. To avoid using an *ad-hoc* action space that would resonate with the intuitively expected behaviour of each agent, all agents maintain the action space from 3.2.5.2. Under these conditions, the agents are heterogeneous since they have different observations and are expected to develop distinctive behaviours.

Chapter 4

Experimental Setup and Result Discussion

The purpose of this chapter is twofold. First, it will mirror the Methodology chapter by presenting the results obtained for each of the different scenarios and their variations. Once the outcomes of the experiments have been shown, the chapter will proceed to the analysis and discussion.

The figures in this chapter that depict the results obtained often plot two different lines, which are related to the same data. The first line, labelled as "Original" represents the actual values observed. However, the second line displays a running average for the same information. The purpose of this last plot is to facilitate the analysis by bringing up patterns that could not be recognized otherwise.

4.1 Environment Configurations

Before studying different multi-agent scenarios, a first approach using single-agent reinforcement learning was used. The goal was to find a set of configurations that made the environment more accessible to navigate without compromising the study. This analysis focused on the target observations distribution and frequency throughout the satellite's orbit and how this variability impacted the learning.

With this in mind, a DQN agent and an A2C agent were placed alone in the environment. The variations applied included maintaining a constant number of images covering 75% of the orbit and later a variable number of images covering from 12.5% to 25% of the orbit. In the case of A2C, the agent was also placed in a scenario where 25% to 50% of the orbit contained possible tasks.

The following are the results obtained for the environment tuning tests. It is important to mention that, ideally, the agent can learn to exploit the downlink mechanism to achieve long or

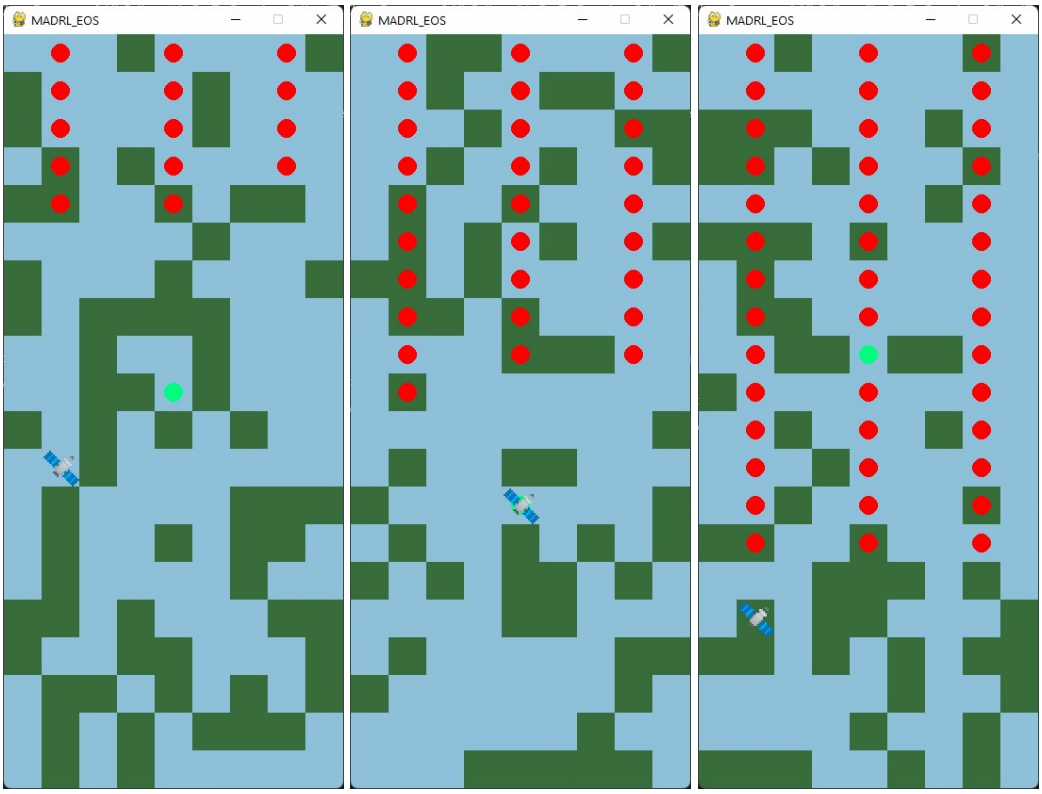

Figure 4.1: Environments with 12% to 25%, 25% to 50% and exactly 75% of orbit positions with images. Target observations are represented by red dots and the ground stations are represented by the green dot.

even infinite episodes. Since ideally the experiments could go on indefinitely, the episodes were interrupted once the agent captured a total number of images equal to 20 times the value of the initial memory. This means that for the single-agent setting, once 200 images were acquired, the agent moved on to the following episode.

4.1.1 Deep Q-Network

Starting with the DQN model, it is possible to see from Figure 4.2 that although the agents' performance seems to get better over time, it never reaches its full potential, resulting in very poor results. These results are in accordance with the expectations, since the environment does not present enough variability.

On the other hand, it is possible to see in 4.3 that increasing variability, by introducing several steps in the orbit in which the agent does not need to capture an image, produces significantly better results. This improvement is noticeable, considering that the running average of the number of captures grows from an interval of 15 to 25 images per episode to an interval of 50 to around 130. Since in both variations the agent was able to capture more than 10 images, the bare minimum to finish the episodes, it is possible to see that the satellite was able to downlink images to capture new data.

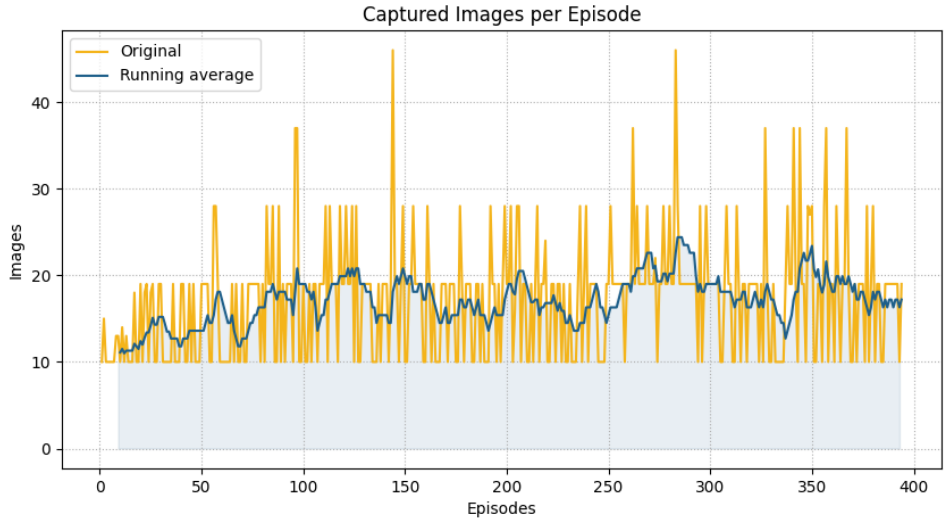

Figure 4.2: DQN single agent environment with fixed number of images (75%)

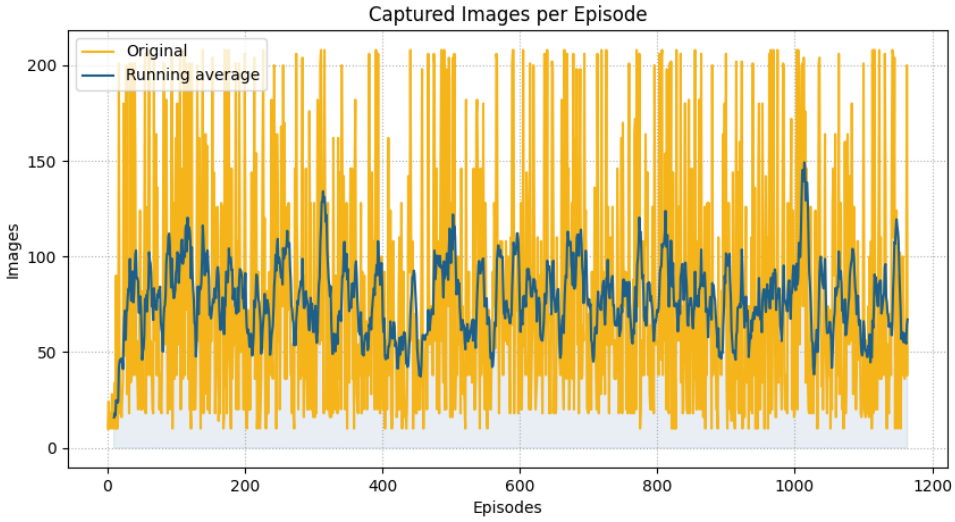

Figure 4.3: DQN single agent environment with varying number of images – from 12.5% to 25%

Another important observation is that, although both cases exhibit a highly stochastic and variable behaviour, the performances stop improving relatively soon. The explanation for this phenomenon may rely on the parameterization chosen. This stagnancy is probably due to the fact that the exploration factor ε used is reduced to its minimum too early in the training process. This drop impels the estimation of the value-function to stop improving too soon and rapidly converge to a local optima. The results obtained suggest that the agent did not find an optimal Q-function estimation and consequential policy. Instead, the model stopped exploring new possibilities and acted as well as their previous knowledge allowed. As previously explained, in DQN it is possible

to influence the exploration-exploitation trade-off in two different ways: either by changing the initial values of ε and its decrement, or by changing the size of the batch sampled from the replay buffer.

4.1.2 Advantage Actor Critic

Moving on to the A2C model, as depicted in Figure 4.4, the agent behaved very poorly in the low variability environment. In this situation, it is clear that the lack of a replay buffer, combined with a poor exploration, resulted in a locally optimal policy. A possible intuitive explanation for these results is the following. As the agent encountered a significant amount of similar states for which the action that produced the best rewards was to capture, every episode ended before the agent could explore and discover the downlink operation only accessible further ahead in orbit.

The difference to the high variability environment that maintains the number of images between 12.5% and 25% of the orbit length is truly noticeable. As one can conclude by looking at Figure 4.5, it takes around 2000 episodes for the agent to start learning and finishing episodes after reaching 200 captured images.

As a test to further understand the impact of changing the number of target observations and their disposition, a scenario with an image coverage of 25% to 50% was tested. The results exhibited in Figure 4.6 show that, while very similar, the performance shows a slight decrease from the one achieved by the 12.5% to 25% interval. In this situation, the agent seems to start learning between 2000 and 2500 episodes.

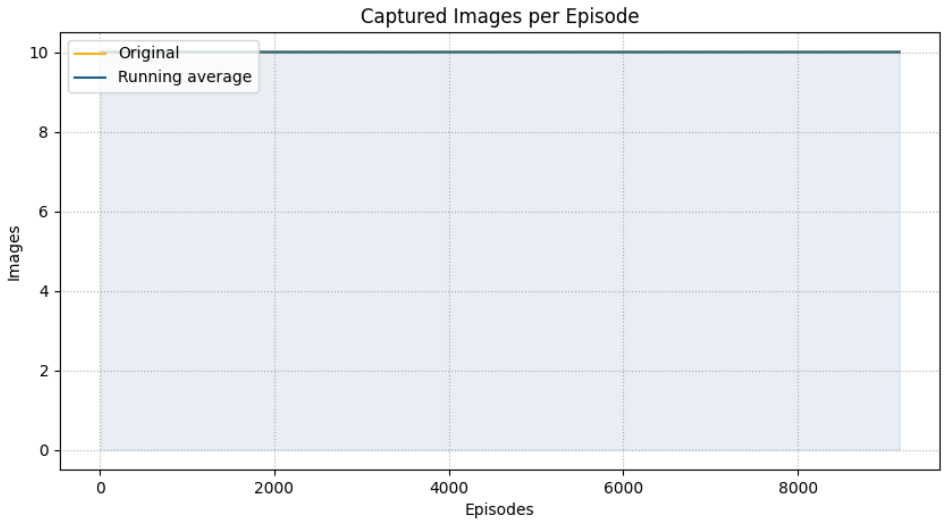

Figure 4.4: A2C single agent environment with fixed number of images (75%)

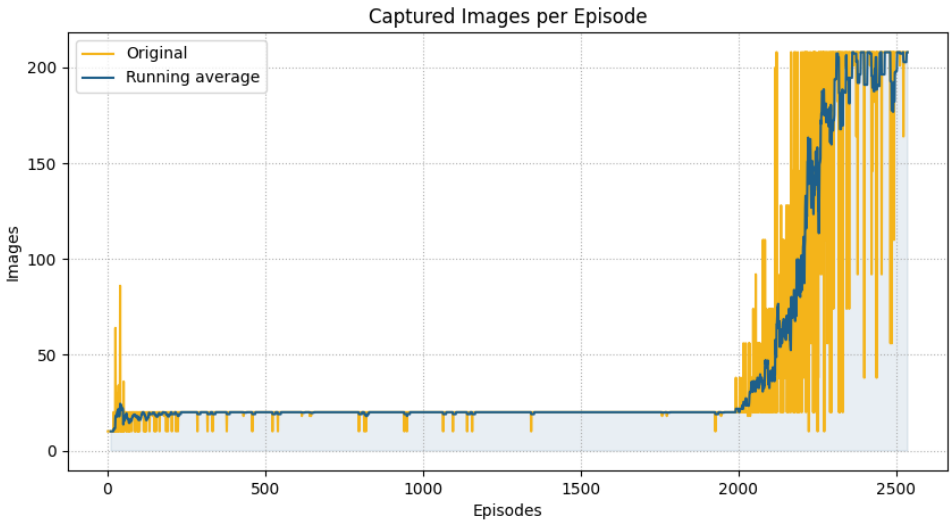

Figure 4.5: A2C single agent environment with varying number of images – from 12.5% to 25%

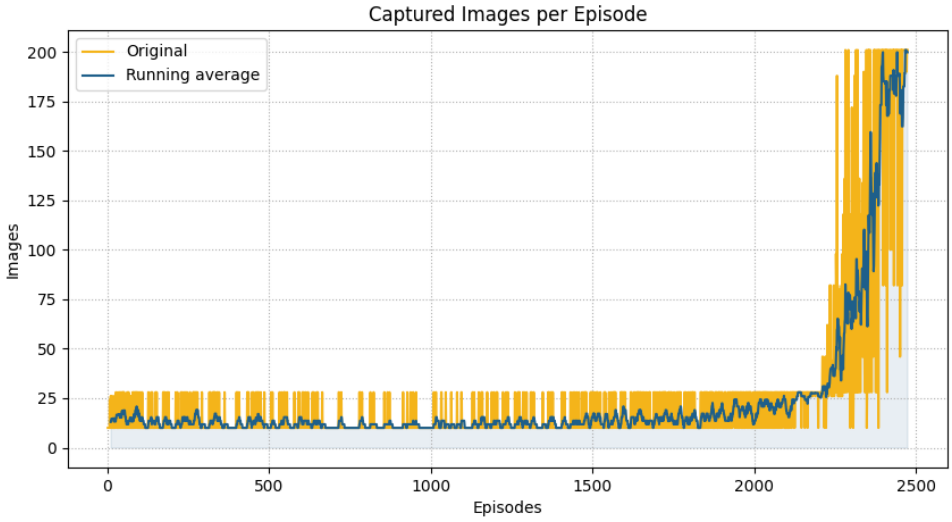

Figure 4.6: A2C single agent environment with varying number of images – from 25% to 50%

4.2 Indirect Cooperation Settings

The results obtained in the previous stage directly impacted the choices made in the rest of the study. For instance, since deriving the policy from the approximated value-function appeared to be less productive than approximating an ideal policy and the value-function simultaneously. The suggested scenarios were approached using the A2C algorithm, maintaining 12.5% to 25% of the orbit covered by images. Like previously established, the multi-agent scenarios were run using the three reward functions (R_t^1, R_t^2, and R_t^3). Once again, it is relevant to point out that since the

total available memory in the system increased to 20 images, the episodes are now limited to 400 image captures.

4.2.1 Independent Learners

When looking at the results graphs for the independent learners, one can see a clear difference in the learning process between the first two reward functions and R_t^3. Using R_t^1 the agent starts learning around 350 episodes (4.7), while R_t^2 allows the agents' performance to start rising sooner, before reaching 200 episodes (4.8). For the independent learners, the R_t^3 function resulted in an unstable learning, that never actually converged while exhibiting poor returns (4.9).

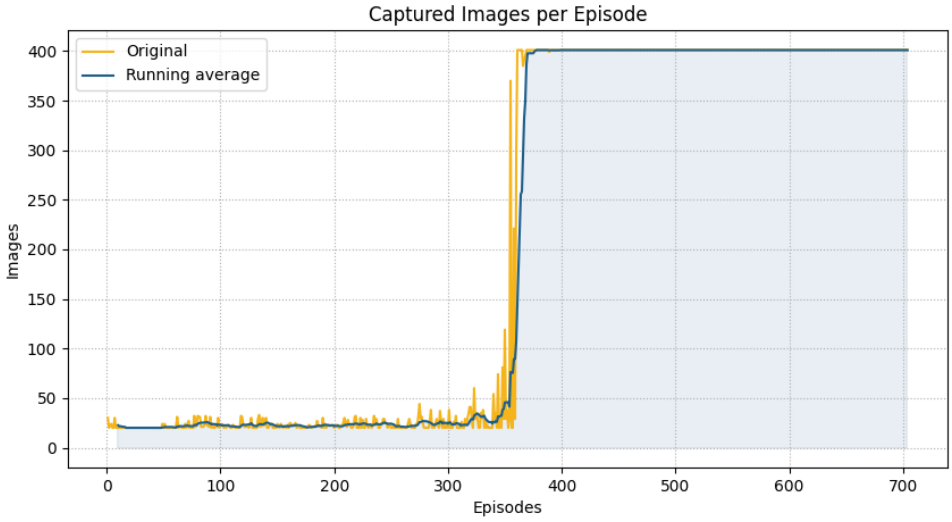

Figure 4.7: Images captured by two A2C independent learners rewarded according to R_t^1

4.2.2 Partial Communication

In the case of the communicating agents, the R_t^2 reward signal (4.11) was the one displaying the most promising results, as the learning process had a consistent improvement since the beginning of training before eventually stabilizing around 400 episodes. These advances only came later for R_t^1 (4.10) and R_t^3 (4.12) – once showing signs of progress after 3000 episodes. The R_t^3 feedback, however, has the advantage when compared to R_t^1, since it reaches the upper limit imposed before 5000 episodes, while the latter does not.

4.2.3 Full Communication

As it is possible to conclude from the figures 4.13, 4.14 and 4.15 the learning process was not as successful when compared to the partial communication scenario. Firstly, the R_t^1 function results in a noisy performance that seems to evolve around 8000 episodes. On the other hand, the agents

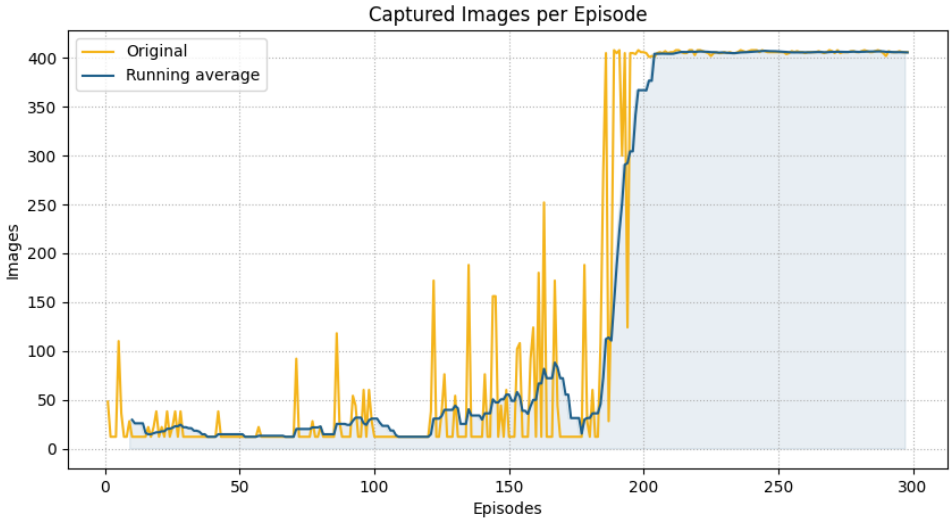

Figure 4.8: Images captured by two A2C independent learners rewarded according to R_t^2

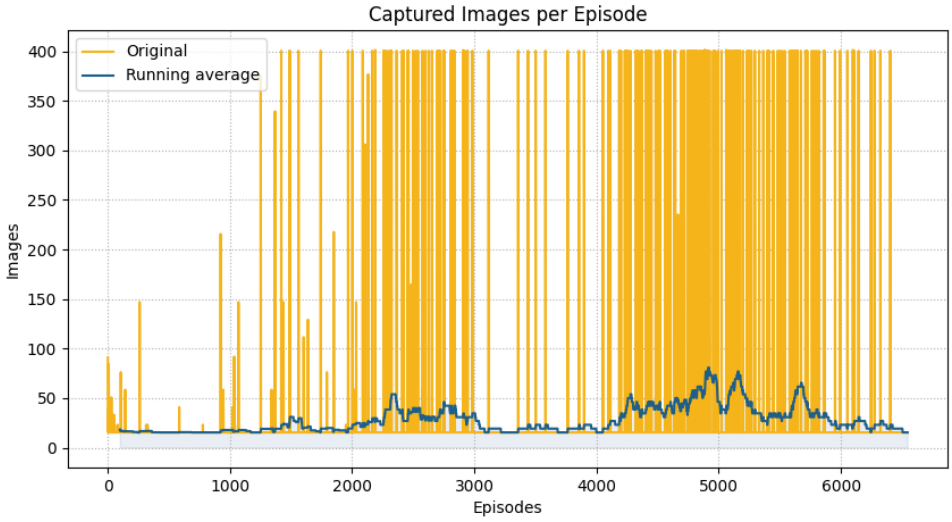

Figure 4.9: Images captured by two A2C independent learners rewarded according to R_t^3

reinforced according to R_t^2 seems to start improving before their 150^{th} episode. Finally, R_t^3, the agents appear to start learning how to exploit the environment around 4000 episodes, achieving an unreliable performance that is characterized by a lot of variability. It is possible to see that after the initial upward slope, it becomes difficult to predict the number of captured images per episode.

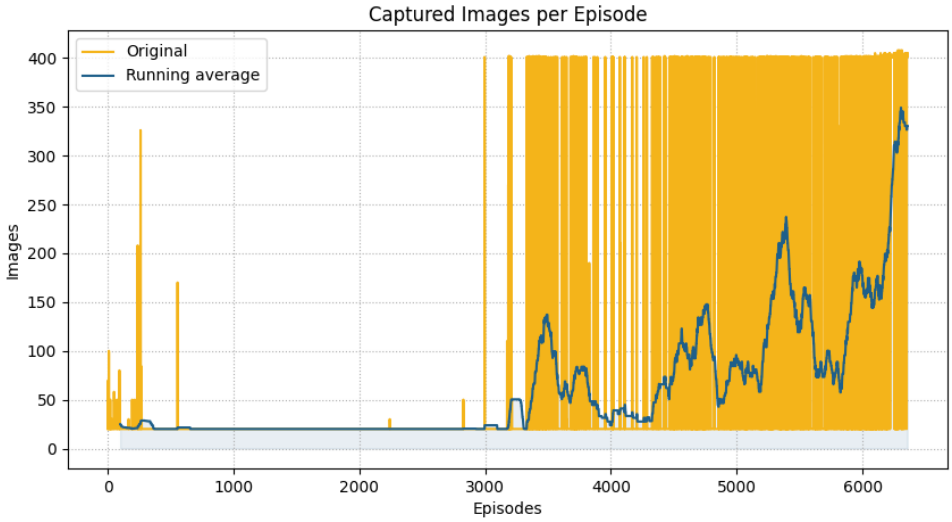

Figure 4.10: Images captured by two A2C communicating learners, rewarded according to R_t^1

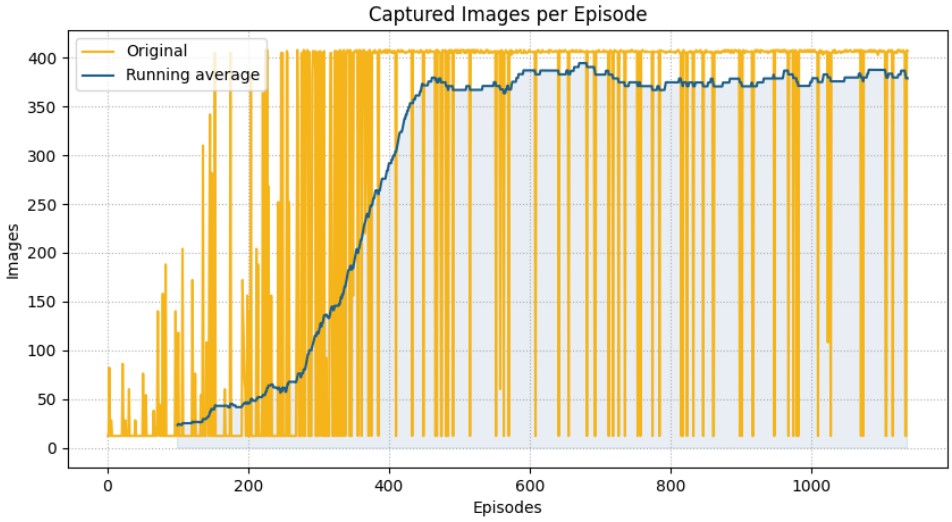

Figure 4.11: Images captured by two A2C communicating learners, rewarded according to R_t^2

4.3 Direct Cooperation Settings

4.3.1 Two Heterogenous Interacting Agents

When given the chance to interact with each other, the agents learned how to cooperate to a small degree, as can be seen by 4.16. Although the agents are able to renew their resources multiple times, their performance stabilized around 300 episodes.

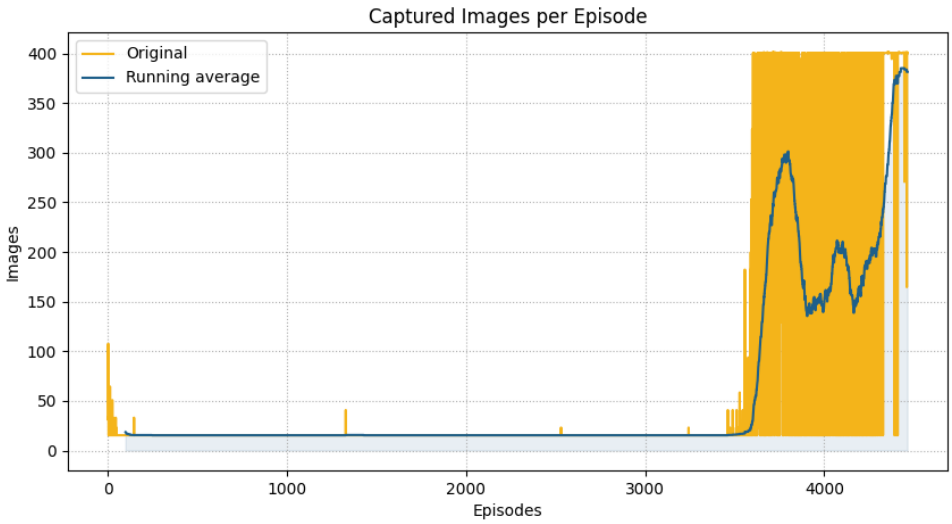

Figure 4.12: Images captured by two A2C communicating learners, rewarded according to R_t^3

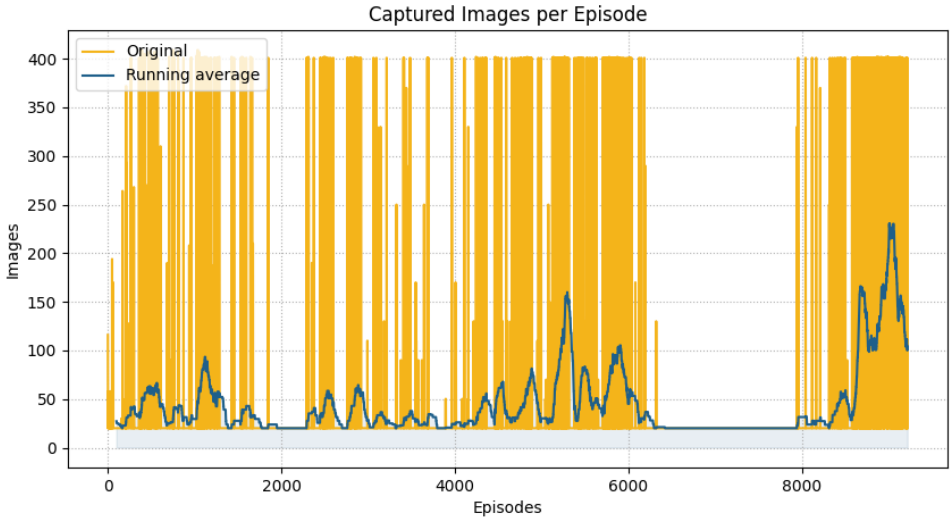

Figure 4.13: Images captured by two A2C fully observable learners, rewarded according to R_t^1

4.3.2 Three Homogenous Interacting Agents

Although the learning process seems to start giving promising results very early (before 100 episodes), the number of captures was underwhelming (Figure 4.17). Even though introducing a new agent actually makes it easier for agents to sustain longer episodes, and consequently capture a larger number of images, this number never reached its true potential of 600 images per episode.

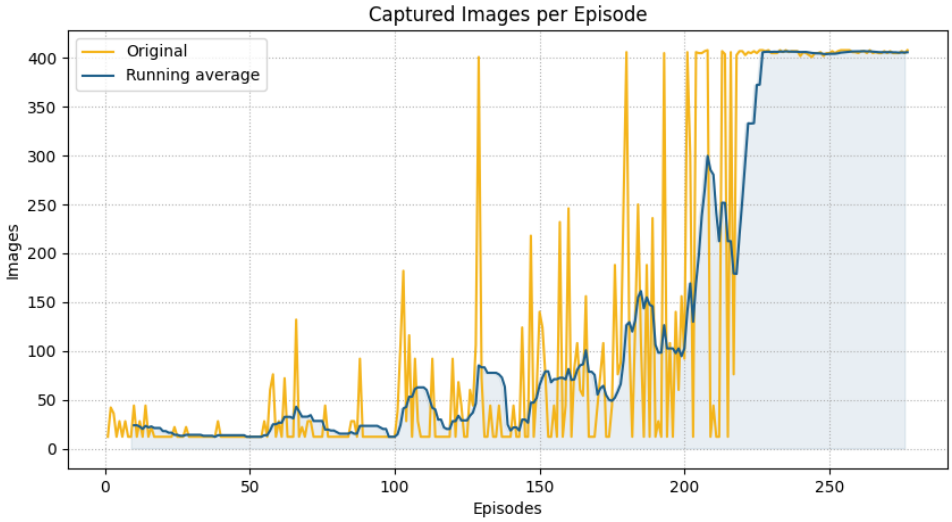

Figure 4.14: Images captured by two A2C fully observable learners, rewarded according to R_t^2

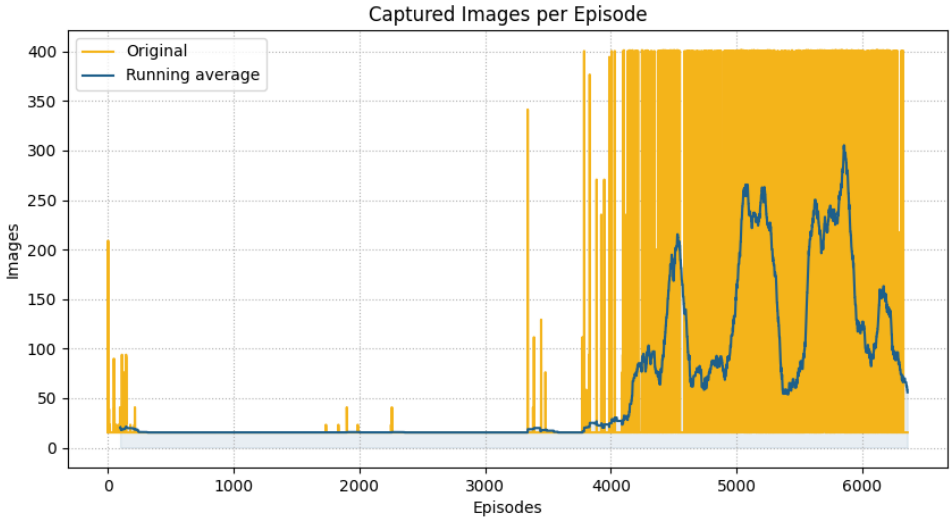

Figure 4.15: Images captured by two A2C fully observable learners, rewarded according to R_t^3

Moreover, the results were in accordance to the prediction made. The agents limited themselves to capturing images and downlinking them when they had a chance. The fact that there is one more agent that needs to waste their resources before the episode comes to an end, also might buy the other agents some time to reach the ground station. This lack of image transfers among satellites can be seen in the graph 4.18. As shown, the initial exploration lets the satellites transfer over 80 images in the start of the training process. However, the agents quickly discard transferring images as a viable option – the line starts to drop since the very beginning of training, until it reaches 0.

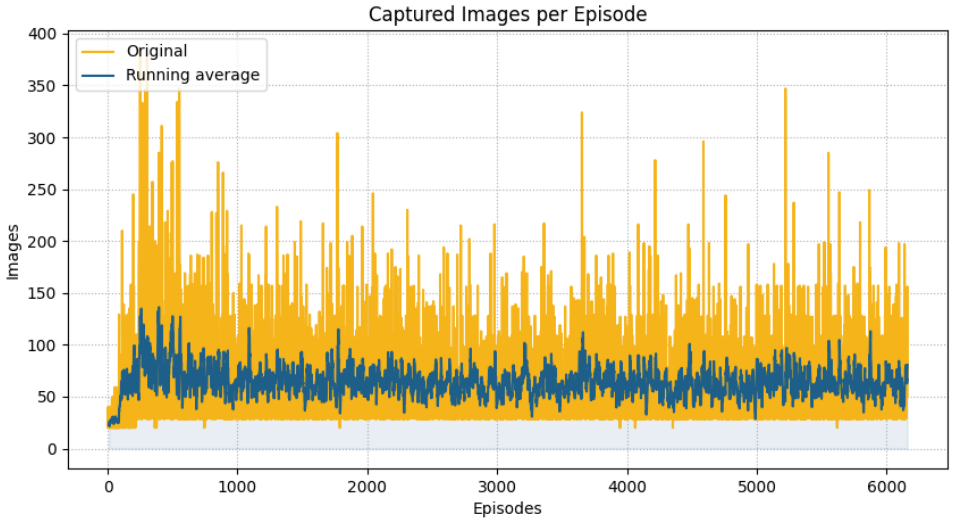

Figure 4.16: Images captured by two A2C heterogenous and partially communicating learners, rewarded according to R_t^2

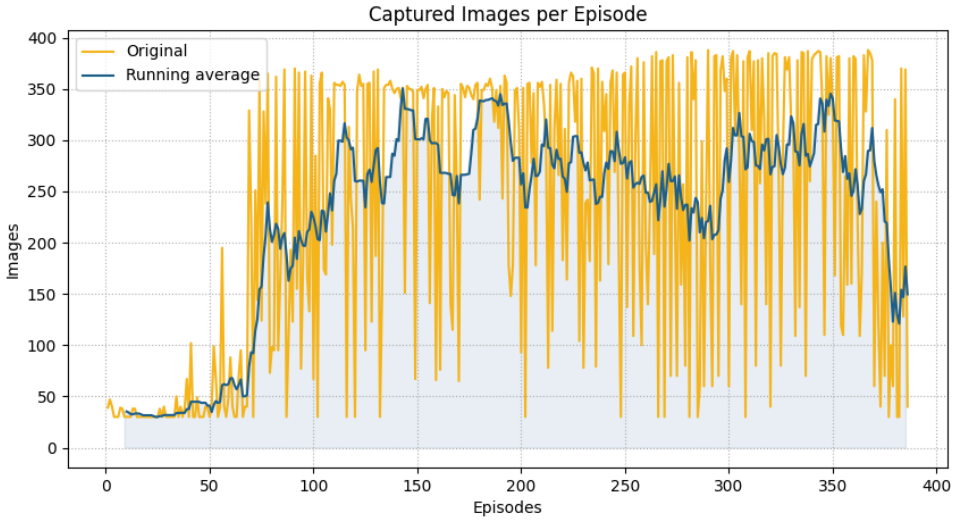

Figure 4.17: Images captured by three A2C homogenous and partially communicating learners, rewarded according to R_t^2

4.3.3 Three Heterogeneous Interacting Agents

Restraining the movement of the agents, and therefore their access to the ground station, had a clear impact on the results (see Figure 4.20). The number of images starts to rise from 30 captures – the bare minimum before an episode ends, given the memory of each satellite – and, even though it continues to be above 30, the number of images per episode averages around 50 to 60 captures with a slight increase by the end of training.

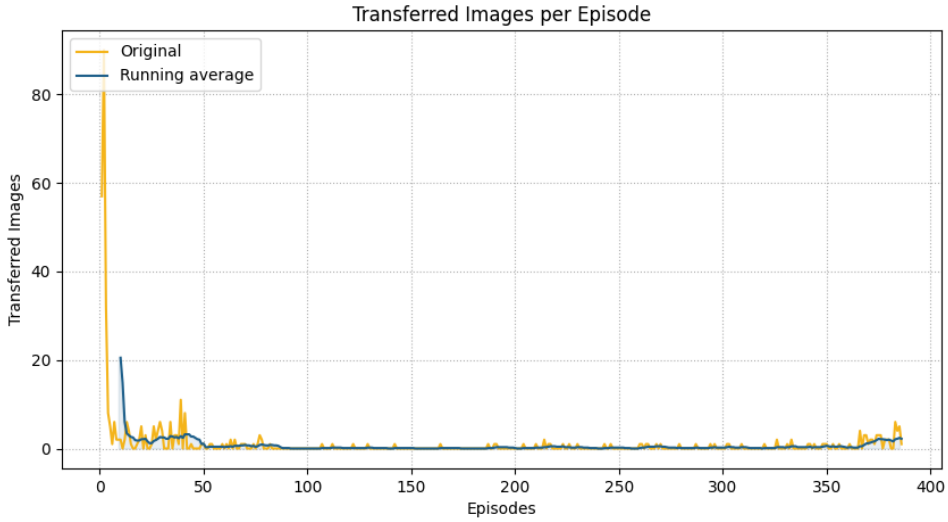

Figure 4.18: Images transferred between three A2C homogenous and partially communicating learners, rewarded according to R_t^2

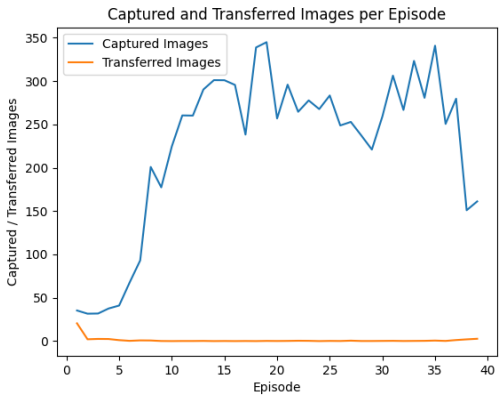

Figure 4.19: Relation between the number of images captured and transferred by three A2C homogenous and partially communicating learners, rewarded according to R_t^2

More importantly, these results suggest a change in behaviour as the number of transferred images takes a different shape compared to 3.2.5.2. Although, its values are still very low, this line presents some local extremes that seem highly correlated with the peaks in the number of captured images (Figure 4.22). These results suggest that the success of the agents was greater when they chose to transfer images between them, releasing some resources. At the same time, the line shows that the optimized policies led to a suboptimal solution – minimal transfers and low number of image captures. Finally, since the number of images that the satellites share is so low in comparison to the actual number of target observations made, these values might indicate the presence of lazy agents. In other words, not all satellites adopt a behaviour that positively impacts

the constellation. A plausible explanation is that the satellites that do not have access to the ground station simply stopped contributing to the common goal when they reached their memory limits.

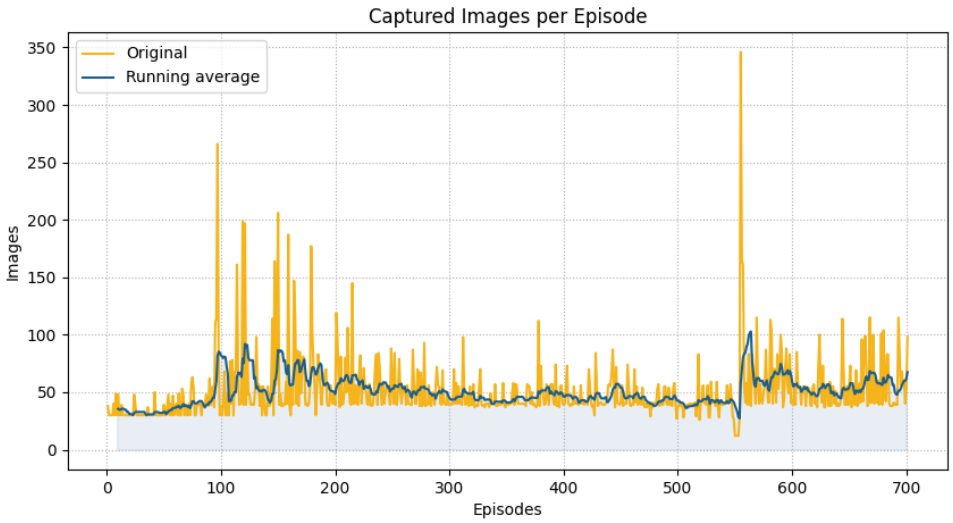

Figure 4.20: Images captured by three A2C heterogeneous and partially communicating learners, rewarded according to R_t^2

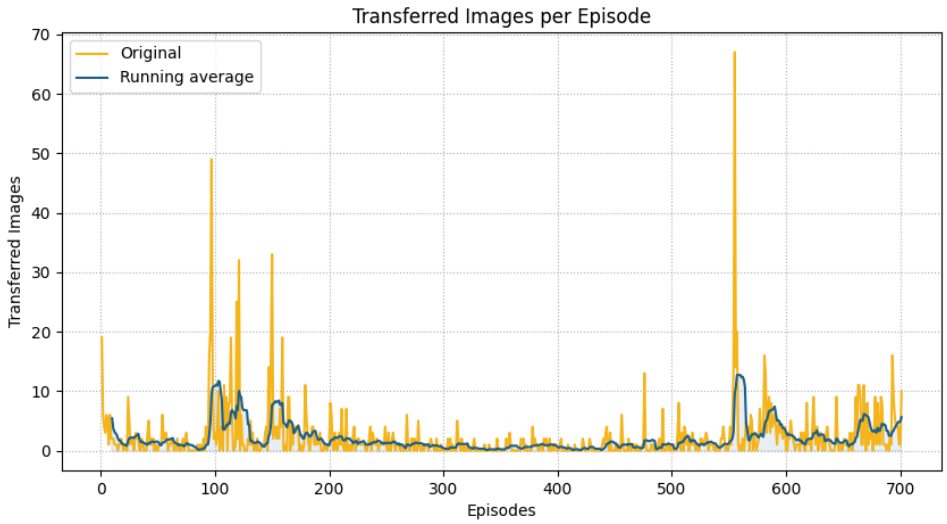

Figure 4.21: Images transferred by three A2C heterogeneous and partially communicating learners, rewarded according to R_t^2

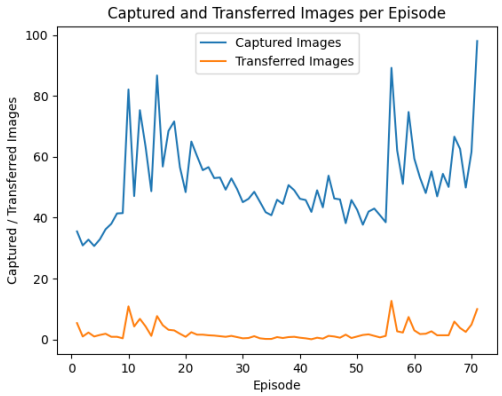

Figure 4.22: Relation between the number of images captured and transferred by three A2C heterogeneous and partially communicating learners, rewarded according to R_t^2

Chapter 5

Conclusions

With the increasing number of decentralised systems, the Multi-Agent System paradigm has gained a significant role in the study of the dynamics and interactions of the multiple entities comprised in such systems. Additionally, the use of Deep Reinforcement Learning in the Multi-Agent context (MADRL) has become particularly advantageous. It allows agents to develop innovative strategies without the need to define the dynamics and rules of their environment explicitly. For this reason, MADRL displays great applicability to solve problems requiring a lot of coordination, which can appear in the form of cooperation, competition, or a mixture of both. Given the great relevance and complexity of EOS systems, it is crucial to promote and invest in research that can improve the coordination of the satellite constellations. Hence, this setting can significantly benefit from MAS and DRL approaches.

In the literature, it is possible to verify that many challenges must be solved to guarantee the best possible usufruct of the services provided by EOS. Similarly, the state-of-the-art includes numerous solutions that resort to many different paradigms. With respect to collaborative resource management, it is possible to find implementations that rely on MADRL to propose a solution. However, as this field is still relatively recent, there is still much room for improvement.

The MADRL problem conceived attempts to adopt the sequential decision-making paradigm while addressing the necessary resource management aspects inherent to EOS systems. The proposed solution starts with an environment tuning stage. While relying on single-agent reinforcement learning, this phase was essential to select the algorithm and the background conditions used in the remaining experiments. With the goal of studying collaborative behaviours, the scenarios developed included two agents that were rewarded according to three distinct feedback signals. Moreover, these experiments introduce the notion of communication. The agents' perceptions vary from being entirely local to including some or all information contained in other observations.

Finally, the implementation allows the agents to interact with one another. This way, their actions directly impact the other satellites' states. In this setting, some conditions were imposed

to create three homogeneous or heterogeneous agents and study the changes in behaviour.

The results obtained demonstrated that the second reward function was the most successful. However, they do not imply any specific pattern for choosing a reward function for each observation type. Furthermore, the analysis of the three interacting agents suggests that the cooperation may have been harmed by the presence of lazy agents.

5.1 Main Contributions

This work aimed to contribute not only to the improvement of EOS cooperation, but primarily to the domain of Multi-Agent Deep Reinforcement Learning. The main contributions achieved are the following:

- Formalization of a resource management problem as a MADRL problem.
- Development and tuning of an environment that facilitates the training of multiple agents in the context of cooperative resource management.
- Exploratory analysis of different strategies for reward designs and study of cooperative policies that emerge from these reinforcement strategies.
- Exploratory analysis of the impact of different communication levels through information sharing in observations.
- Development of scenarios that compare the behaviour of homogenous and heterogeneous constellations of satellites.
- Empiric examination of experimental work.

5.2 Future Work

The proposed solution successfully addressed the identified issues in the problem definition. However, as expressed in Chapter 3, the implementation aspired to allow further work to address other aspects of EOS coordination. The goal of the modifications identified is to improve the solution with respect to the MADRL efficacy and efficiency. Additionally, they seek a more faithful modelling of the environment as a way to ensure the utility of the solution in real-life scenarios.

One of the aspects that were considered for the initial problem definition was to assign a priority to each target observation. This way, one could then study if the agents were able to be selective regarding their resource investments and prioritize some captures over others.

This priority could also take into account a time factor. For instance, the importance of an image could be increased due to it being requested sooner than others. Furthermore, prioritizing observations related to time-sensitive events could be of particular interest to watch over climacteric disasters or even regions with ongoing armed conflicts.

With respect to images, it would also be of interest to consider that each target observation might require a different set of sensors to be captured. This diversity would mean that in some situations, only a subset of the satellites would be apt to perform the acquisition, once again raising the demand for proper coordination.

Moreover, the current problem definition considers that the only resource spent is memory space and that each image requires the same investment. Firstly, observations may demand different amounts of information. What is more, multiple other factors must be considered before accepting new assignments, namely fuel and the time necessary to position the satellite and complete the task successfully.

A possible additional cooperation scenario would include tasks that were available to multiple agents simultaneously and, therefore, would require some use of negotiation to facilitate coordination.

One of the most substantial simplifications of the problem presented is the absence of noise in communications. However, it would be intriguing to include some uncertainty in data transmission between agents and to the ground stations. Similarly, it would be valuable to study the reaction of RL agents to unpredictable adverse weather conditions that impede observations.

Regarding the MADRL domain, there are also a few improvements and experiments that could be done in the future. Firstly, the DQN approach proposed could benefit from a better parameterization, as the results were unsatisfactory. Furthermore, by resorting to algorithms such as QMIX or QTRAN, one could try to mitigate the presence of lazy agents and guarantee contributions from every satellite involved. Finally, some adjustments to the problem's action space could be made to allow the use of state-of-the-art architectures such as MADDPG.

References

[1] E. Andrejczuk, J. A. Rodriguez-Aguilar, and C. Sierra. A concise review on multiagent teams: Contributions and research opportunities. In *Multi-Agent Systems and Agreement Technologies*, pages 31–39. Springer International Publishing, 2017.

[2] G. Bastianelli, D. Salamon, A. Schisano, and A. Iacobacci. Agent-based simulation of collaborative unmanned satellite vehicles. In *2012 IEEE First AESS European Conference on Satellite Telecommunications (ESTEL)*. IEEE, October 2012.

[3] R. Bellman. A markovian decision process. *Indiana University Mathematics Journal*, 6(4):679–684, 1957.

[4] G. Bonnet and C. Tessier. Collaboration among a satellite swarm. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '07, New York, NY, USA, 2007. Association for Computing Machinery.

[5] S. Bouveret, Y. Chevaleyre, N. Maudet, and H. Moulin. Fair allocation of indivisible goods. In Felix Brandt, Vincent Conitzer, Ulle Endriss, Jerome Lang, and Ariel D. Procaccia, editors, *Handbook of Computational Social Choice*, pages 284–310. Cambridge University Press, 2016.

[6] K. L. Cahoy and A. K. Kennedy. Initial results from access: An autonomous cubesat constellation scheduling system for earth observation. 77 Massachusetts Avenue, Cambridge, MA 02139, 2017.

[7] L. Chen, T. Guo, Y. Liu, and J. Yang. Survey of multi-agent strategy based on reinforcement learning. In *2020 Chinese Control And Decision Conference (CCDC)*, pages 604–609, August 2020.

[8] Y. Chevaleyre, P. E. Dunne, U. Endriss, J. Lang, M. Lemaître, N. Maudet, J. A. Padget, S. Phelps, J. A. Rodríguez-Aguilar, and P. Sousa. Issues in multiagentresource allocation. *Informatica (Slovenia) 30, 1 (2006), 3–31*, January.

[9] C. Dai, G. Zheng, and Q. Chen. Satellite constellation design with multi-objective genetic algorithm for regional terrestrial satellite network. *China Communications*, 15(8):1–10, August 2018.

[10] P. Feng, H. Chen, S. Peng, L. Chen, and L. Li. A method of distributed multi-satellite mission scheduling based on improved contract net protocol. In *2015 11th International Conference on Natural Computation (ICNC)*, pages 1062–1068. IEEE, August 2015.

[11] J. N. Foerster, F. Song, E. Hughes, N. Burch, I. Dunning, S. Whiteson, M. Botvinick, and M. Bowling. Bayesian action decoder for deep multi-agent reinforcement learning. *ArXiv*, November 2018.

[12] Cooperative Institute for Meteorological Satellite Studies. Spatial analysis. https://cimss.ssec.wisc.edu/sage/remote_sensing/lesson3/concepts.html. Accessed: 01-03-2022.

[13] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3-4):219–354, 2018.

[14] A. Globus, J. Crawford, J. Lohn, and A. Pryor. A comparison of techniques for scheduling earth observing satellites. In *Proceedings of the 16th Conference on Innovative Applications of Artifical Intelligence*, IAAI'04, page 836–843. AAAI Press, 2004.

[15] N. Gupta, G. Srinivasaraghavan, S. K. Mohalik, and M. E. Taylor. Hammer: Multi-level coordination of reinforcement learning agents via learned messaging. *ArXiv*, January 2021.

[16] H. Hu and J. N. Foerster. Simplified action decoder for deep multi-agent reinforcement learning. *ArXiv*, December 2019.

[17] C. Iacopino, P.L. Palmer, N. Policella, A. Donati, and A. Brewer. How ants can manage your satellites. August 2013.

[18] B. Jang, M. Kim, G. Harerimana, and J. W. Kim. Q-learning algorithms: A comprehensive classification and applications. *IEEE Access*, 7:133653–133667, 2019.

[19] C. L. Korb and A.R. Korb. Methods for optimizing the performance, cost and constellation design of satellites for full and partial earth coverage. *USPatent 10,664,782.*, 2020.

[20] W. L. Keng L. Graesser. Foundations of deep reinforcement learning: Theory and practice in python. https://www.informit.com/articles/article.aspx?p=2995356seqNum=5, December 2019.

[21] M. Lemaître, G. Verfaillie, H. Fargier, J. Lang, N. Bataille, and J. Lachiver. Equitable allocation of earth observing satellites resources. 2003.

[22] M. Lemaitre, G. Verfaillie, F. Jouhaud, J. Lachiver, and N. Bataille. Selecting and scheduling observations of agile satellites. *Aerospace Science and Technology*, 6(5):367–381, September 2002.

[23] D. Li, W. Haijiao, Y. Zhen, G. Yanfeng, and S. Shen. An online distributed satellite cooperative observation scheduling algorithm based on multiagent deep reinforcement learning. *IEEE Geoscience and Remote Sensing Letters*, 18(11):1901–1905, 2021.

[24] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *ArXiv*, September 2015.

[25] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. pages 6382–6393, 2017.

[26] L. Matignon, G. Laurent, and N. Fort-Piat. Independent reinforcement learners in cooperative markov games: A survey regarding coordination problems. *The Knowledge Engineering Review*, 27(11):1–31, March 2012.

[27] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *ICML 2016*, February 2016.

[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *ArXiv*, December 2013.

[29] A. Nedic, A. Ozdaglar, and P.A. Parrilo. Constrained consensus and optimization in multi-agent networks. *IEEE Transactions on Automatic Control*, 55(4):922–938, April 2010.

[30] A. OroojlooyJadid and D. Hajinezhad. A review of cooperative multi-agent deep reinforcement learning. *ArXiv*, abs/1908.03963, August 2019.

[31] F. G. Ortiz-Gomez, D. Tarchi, R. Martinez, A. Vanelli-Coralli, M. A. Salas-Natera, and S. Landeros-Ayala. Cooperative multi-agent deep reinforcement learning for resource management in full flexible VHTS systems. *IEEE Transactions on Cognitive Communications and Networking*, 8(1):335–349, March 2022.

[32] G. Picard, C. Caron, J. Farges, J. Guerra, C. Pralet, and S. Roussel. Autonomous agents and multiagent systems challenges in earth observation satellite constellations. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '21, pages 39–44, Richland, SC, 2021. International Foundation for Autonomous Agents and Multiagent Systems.

[33] M. Pîslar, D. Szepesvari, G. Ostrovski, D. Borsa, and T. Schaul. When should agents explore? *ArXiv*, August 2021.

[34] T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. Foerster, and S. Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. *ArXiv*, March 2018.

[35] A. H. Sanchez, T. Soares, and A. Wolahan. Reliability aspects of mega-constellation satellites and their impact on the space debris environment. In *2017 Annual Reliability and Maintainability Symposium (RAMS)*, pages 1–5. IEEE, 2017.

[36] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *ArXiv*, November 2015.

[37] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *ArXiv*, July 2017.

[38] P. O. Skobelev, E. V. Simonova, A. A. Zhilyaev, and V. S. Travin. Application of multi-agent technology in the scheduling system of swarm of earth remote sensing satellites. *Procedia Computer Science*, 103:396–402, 2017.

[39] K. Son, D. Kim, W. J. Kang, D. E. Hostallero, and Y. Yi. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. *ArXiv*, May 2019.

[40] A. Stooke and P. Abbeel. Accelerated methods for deep reinforcement learning. *ArXiv*, March 2018.

[41] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel. Value-decomposition networks for cooperative multi-agent learning. *ArXiv*, June 2017.

[42] R. S. Sutton and A. G. Barto. Reinforcement learning: An introduction. 2018.

[43] S. B. Thrun. Efficient exploration in reinforcement learning. Technical report, USA, 1992.

[44] A. E. Vasegaard, M. Picard, F. Hennart, P. N., and S. Saha. Multi criteria decision making for the multi-satellite image acquisition scheduling problem. *Sensors*, 20(5):1242, February 2020.

[45] J. Wang, X. Zhu, L. T. Yang, J. Zhu, and M. Ma. Towards dynamic real-time scheduling for multiple earth observation satellites. *Journal of Computer and System Sciences*, 81(1):110–124, February 2015.

[46] X. Wang, H. Zhang, S. Bai, and Y. Yue. Design of agile satellite constellation based on hybrid-resampling particle swarm optimization method. *Acta Astronautica*, 178:595–605, January 2021.

[47] G. Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, volume 1, page 648. July 2000.

[48] G. Weiss. *Multiagent Systems*. MIT Press, 2013.

[49] G. Yu, T. Vladimirova, and M. N. Sweeting. Image compression systems on board satellites. *Acta Astronautica*, 64(9-10):988–1005, May 2009.