Chair of Spacecraft Systems
TUM School of Engineering and Design
Technical University of Munich

Deep Reinforcement Learning for Decentralized Autonomous Decision-Making in Federated Satellite Systems

Enhancing Space Operations with AI

Clemente Javier Juan Oliver

Thesis for the attainment of the academic degree

Master of Science (M.Sc.)

at the TUM School of Engineering and Design of the Technical University of Munich.

Examiner:

Prof. Dr. Alessandro Golkar

Supervisor:

M.Sc. Vincenzo Messina

Submitted:

Munich, 27.08.2024

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

Munich, 27.08.2024 Clemente Javier Juan Oliver

Abstract

The exponential growth in the number of satellites orbiting Earth is in need of the development of more efficient, autonomous decision-making frameworks for managing large satellite constellations. Traditional centralized methods of satellite operation are increasingly inadequate in dealing with the dynamic and unpredictable nature of space environments. To address these challenges, this thesis investigates the application of Deep Reinforcement Learning (DRL) for decentralized autonomous decision-making in Federated Satellite Systems (FSS), a paradigm that enables the collaborative use of satellite resources across different owners and missions.

This research presents a comprehensive framework that integrates DRL into satellite operations, with the objective of enhancing real-time decision-making capabilities. A modular simulation environment was developed to model the interactions between multiple satellites within an FSS, simulating various operational scenarios including resource sharing and communication management. The simulation framework supports diverse satellite types and coordination models, ranging from fully centralized to fully decentralized configurations, allowing for a thorough evaluation of different operational strategies.

Three state-of-the-art DRL algorithms—Deep Q-Networks (DQN), Soft Actor-Critic (SAC), and Proximal Policy Optimization (PPO)—were implemented and trained within this simulation environment. These algorithms were evaluated based on their performance in optimizing satellite operations, particularly in tasks such as target observation, data sharing, and resource management.

The scenario considered for this project consists of 20 observer satellites and 20 target objects orbiting Earth. The main goal for the observers is to detect and observe the targets while in orbit, and meanwhile spread the obtained data between all participants while leveraging energy and storage resources. The comparative analysis of these AI agents in different coordination models highlights the strengths and weaknesses of each algorithm in different operational contexts, providing valuable insights into the trade-offs between computational efficiency, scalability, and mission success.

Furthermore, the practical feasibility of deploying these DRL algorithms in real satellite systems was assessed by implementing them on an NVIDIA Jetson, a hardware platform representative of onboard satellite computers. The performance benchmarks indicate that DRL-based approaches can be effectively integrated into existing satellite architectures, paving the way for more autonomous and resilient space missions.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to Professor Alessandro Golkar for his trust and for his invaluable guidance throughout my time at the Space Systems Chair. Your insightful feedback and suggestions, have greatly enhanced the value of this thesis, and I am immensely grateful for the opportunity to learn under your supervision.

I am also deeply thankful to my supervisor, M. Sc. Vincenzo Messina, for his continuous support, insightful feedback, and the knowledge he has imparted to me over the course of this challenging project and my time at the chair. Your patience, teaching ability, and unwavering help have been incredible in my development as a researcher. I deeply respect your commitment to both teaching and research, and I am profoundly grateful for the opportunity to learn under your guidance. The lessons I have learned from you extend far beyond satellites.

Additionally, I would like to extend my sincere thanks to M. Sc. Ramón García, whose expertise, advice and help were instrumental throughout this journey. Ramón, your support during critical times was invaluable, and I am deeply grateful for your dedication and mentorship throughout my time at the Chair. You have been a mentor in your own right, and I truly admire the way you approach research.

To the entire team at the Space Systems Chair: Lara, Rafael, Jaspar and Sherina, thank you for the camaraderie, support, and countless hours spent together over the last months. You have made this journey not only intellectually rewarding but also personally fulfilling. I have also learned a lot from you outside of the professional field, and I am grateful to have shared this time with such an outstanding group of people.

I cannot forget about my other companions at the chair during this adventure: Leo, Kian, Tinucci, Sidhant, Leire, Rober, Eva... You made the everyday moments special, turning challenges into opportunities for laughter and lasting memories. Whether working late nights or playing darts, the experiences we created together will stay with me. Thank you guys.

Tampoco puedo olvidarme de Marina, Bea y Alonso, sin los que no creo que hubiese sobrevivido tanto tiempo en Munich, gracias por todo y por lo que nos queda.

Y finalmente, para Ana, mamá y papá, por estar siempre ahí aunque estéis a miles de kilómetros, por enseñarme que uno debe perseguir lo que quiere y por darme la oportunidad de hacerlo. Todo esfuerzo tiene su recompensa, y esta también es vuestra. Gracias de corazón.

Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence. 1–4, 6, 27

ANN Artificial Neural Network. 1, 18

BER Bit Error Rate. 11

ConOps Concept of Operations. 3, 8

CTDE Centralized Training with Decentralized Execution. 26

Dec-POMDP Decentralized Partially Observable Markov Decision Process. 3, 19, 20, 28, 29, 32, 33

DNN Deep Neural Network. 17, 18, 20

DQN Deep Q-Networks. 3, 18, 20, 22, 35, 36, 38, 39, 41, 44–48, 50–54, 56, 58, 71

DRL Deep Reinforcement Learning. 1–7, 18, 20–22, 26, 33, 35, 38, 57–59

DSS Distributed Satellite Systems. 7

FSL Free Space Loss. 11, 12

FSS Federated Satellite System. 1–4, 7, 8, 12, 26, 28, 32, 34, 55

ISL Inter-Satellite Link. 4, 6, 7

LEO Low Earth Orbits. 8

LOS Line-of-Sight. 11

MADDPG Multi-agent Deep Deterministic Policy Gradient. 25, 26

MARL Multi-agent Reinforcement Learning. 3, 6, 25–27, 58

MDP Markov Decision Process. 1, 18, 19

MIP Mixed-Integer Programming. 18

ML Machine Learning. 6

POMDP Partially-Observable Markov Decision Process. 18–20

PPO Proximal Policy Optimization. 3, 18, 20, 21, 24, 35, 36, 40, 41, 44–54, 56–58, 71

QPSK Quadrature Phase Shift Keying. 11

RAAN Right Ascension of Ascending Node. 8

RL Reinforcement Learning. 1–3, 6, 17, 18, 20, 27, 28, 32, 33, 35, 38, 56, 57, 71

SAC Soft Actor-Critic. 3, 18, 20, 22, 23, 35, 36, 39, 41, 44–48, 53, 55, 56, 58, 71, 72

SAN Satellite Access Network. 6

SB3 Stable Baselines 3. 2, 27, 38

SNR Signal-to-Noise Ratio. 11

SSO Sun-Synchronous Orbits. 8

TRPO Trust Region Policy Optimization. 24

UHF Ultra High Frequency. 11

VHF Very High Frequency. 11

1 Introduction

1.1 Motivation

As the number of satellites in orbit continues to grow and the demand for their services increases, managing large constellations and interactions between satellites is becoming increasingly challenging. Currently, there are approximately 10,000 satellites in orbit, and this number is projected to rise to 30,000 by the end of the decade [1], [2]. Under these conditions, efficient scheduling and rapid decision-making for satellites will become critical issues. This project aims to enhance the autonomy of satellite operations by implementing an Artificial Intelligence (AI) Deep Reinforcement Learning (DRL) approach for real-time, on-board, collaborative decision-making.

The presentation of Federated Satellite System (FSS) marks a significant shift in the space industry, introducing a decentralized approach to satellite operations. This paradigm, as discussed by Golkar [3], emphasizes the collaborative use of satellite assets across different owners, offering a more efficient and scalable utilization of space resources. The concept not only proposes an innovative business model but also challenges the traditional centralized methods of satellite management.

The integration of DRL into the management of FSS represents a groundbreaking step towards autonomous decision-making. Reinforcement Learning (RL) for satellite operations, as explored in the doctoral works of Harris [4] and Herrmann [5], offers a robust framework for spacecraft planning and scheduling, addressing the complex and dynamic nature of space environments. The work further underscores the potential of RL in enhancing the autonomy of spacecraft, paving the way for more sophisticated DRL applications.

Harris' investigation into the autonomous management and control of multi-spacecraft operations showcases the intricate balance between autonomy and environmental interaction. This research highlights the critical role of advanced computational methods, like DRL, in navigating the unpredictable dynamics of space and enhancing the coordination within federated systems.

Recent advancements in satellite's networks, as outlined by Messina [6], introduce a decentralized optimization framework for satellite network operations. Their work on formulating a dynamic graph optimization framework highlight the need for adaptive, real-time decision-making mechanisms that can cope with the evolving nature of satellite networks. This is where DRL can make a substantial impact, offering a pathway to optimize operations while accounting for temporal and spatial variations.

Moreover, the exploration of AI techniques for next-generation massive satellite networks by Al Homssi et al. [7] illustrates the growing importance of AI in the domain of satellite communication and management. Their research into AI applications, including DRL, for enhancing the efficiency and reliability of satellite networks further validates the potential of DRL in revolutionizing networks of federated satellite systems.

To address these challenges in satellite's efficient, decentralized and uncertain task scheduling in dynamic environments, this work proposes a series of AI algorithms that are trained based on a simulated environment with the objective of maximizing rewards depending on the actions satellites choose, such as observing a target or successfully sharing information with participants of the federation. The choice of RL and Artificial Neural Network (ANN) for on-board autonomous decision-making poses useful advantages such as minimal restrictions on environment and problem representation thanks to the Markov Decision Process (MDP) framework, optimal learning from experience and fast execution times [8].

Our use case's main goal is to observe and spread information about a series of target satellites orbiting earth. This mission is carried out by a set of observer satellites that work inside a FSS, sharing a pool of resources. Each of the observers is equipped with an AI algorithm in order to choose which action to perform in each situation, always aiming at enhancing the mission outcome and leverage the limited resources on-board and on the FSS network.

These algorithms are trained on a FSS simulator which has been specifically developed for this project. This simulator offers different capabilities such as different communication types (ranging from fully decentralized to fully centralized), customizable satellite properties, variable number of satellites, and more settings that help shape the federation to provide an heterogeneous and realistic environment to train and test the AI policies.

Thanks to the adaption of the simulator to RL frameworks, we can train different AI agents to improve our mission outcomes. These agents are then compared between them in different scenarios and communication configurations to see the differences in performance and efficiency of each one of them. In addition, the deployment of these algorithms on an NVIDIA Jetson serves as benchmark for real satellites application, providing a broader testing environment for the thesis.

By leveraging the power of DRL in satellite operations, we seek to explore new frontiers in satellite network management, aiming for a future where satellite systems can autonomously adapt and optimize their operations in real-time. This endeavor not only contributes to the technical advancement of satellite operations but also aligns with the strategic goals of space exploration and utilization, promoting a more interconnected and autonomous space infrastructure.

1.2 Summary of Objectives

The primary objectives of this research can be divided into research questions that have be addressed and a set of objectives that help us answer these questions while providing useful additional outcomes from this project.

1.2.1 Research Questions

Answering the research questions stated for this thesis:

- **RQ1:** How does the application of reinforcement learning differ between centralized and decentralized coordination models in satellite federations, and what are the respective impacts on scalability, resilience, and adaptability?
- **RQ2:** How do various reinforcement learning algorithms compare in terms of computational efficiency and performance when applied to decentralized versus centralized coordination in satellite networks?
- **RQ3:** How do observation tasks impact the performance of reinforcement learning algorithms, particularly in applications such as object detection, and what are the key considerations for optimizing performance in such scenarios?

1.2.2 Goals

1. **Scalable and Efficient Simulation Framework:**
    - Develop a modular simulation framework tailored for Federated Satellite Systems (FSS), enabling the integration of diverse satellite types and their interactions.
    - Ensure the simulation framework is compatible with existing RL frameworks such as Ray RLlib or Stable Baselines 3 (SB3) to facilitate seamless training of AI agents.
    - Incorporate realistic satellite dynamics and communication protocols to provide a robust testing environment for RL algorithms.

2. **Training AI Agents for Autonomous Execution:**
    - Implement and evaluate DRL algorithms to train AI agents capable of autonomously executing satellite tasks.
    - Focus on optimizing resource utilization (e.g., power, storage) and maximizing mission outcomes through intelligent decision-making.
    - Validate the performance of trained agents in simulated environments, ensuring their ability to adapt and respond to dynamic space conditions.
3. **Leveraging Common Resources in Satellite Operations:**
    - Design DRL policies that enable satellites to collaboratively share and leverage resources within the FSS.
    - Explore the potential of Multi-agent Reinforcement Learning (MARL) to enhance coordination and cooperation among multiple satellites.
    - Aim for a decentralized model where individual satellites can make informed decisions based on local observations and shared information.

By achieving these objectives, this research aims to contribute to the advancement of autonomous satellite operations, paving the way for more efficient, resilient, and capable space systems.

To provide a brief overview of the structure of this thesis:

Chapter 2 introduces the Concept of Operations (ConOps) and the main problem addressed in this research. It provides background on both traditional and modern approaches to task scheduling and mission planning in satellites. Additionally, it offers a formal definition of the FSS and highlights key variables and parameters within essential systems such as communication, energy, and storage.

In Chapter 3, the theoretical background of RL is summarized, including the algorithms used in this project. This chapter is the foundation for understanding how RL principles are applied to enhance satellite autonomy within the FSS framework.

Chapter 4 presents the simulation framework developed for this research. This includes details on the implementation methodology, the formulation of the Decentralized Partially Observable Markov Decision Process (Dec-POMDP), and the validation of the simulation framework.

In Chapter 5, a comparative analysis of the training of different DRL algorithms used in this project, namely Deep Q-Networks (DQN), Soft Actor-Critic (SAC), and Proximal Policy Optimization (PPO), is presented. This chapter discusses their hyperparameter search and further training within the FSS framework.

Chapter 6 provides the results obtained from the simulations, studying the effectiveness of the DRL approaches under different coordination setups, including fully decentralized, centralized, and partially decentralized coordination models.

Chapter 7 explores the real-world applications of the developed algorithms, focusing particularly on their potential deployment in actual satellite systems. This chapter aims to bridge the gap between theoretical research and practical implementation by deploying the trained agents in a Jetson AGX Orin.

Finally, the thesis concludes by summarizing the major contributions of the research, discussing its limitations, and suggesting potential directions for future work in autonomous satellite operations and the application of AI in space systems.

2 Problem Formulation

2.1 Introduction and ConOps

The mission involves a network of observer satellites, each equipped with varying communication and resource capabilities, such as different communication bands and battery levels, operating in different orbits. These satellites work together to detect, observe, and share information about a set of target satellites, which represent objects orbiting in space. The primary objective is to maximize the number of high-quality observations while minimizing the duplication of effort and resource consumption across the network, making sure the information is spread as efficiently as possible.

The mission is designed around several key components. Observer satellites are responsible for detecting and observing the target satellites, equipped with communication systems that allow them to share observation data with other observers in the network. Target satellites, on the other hand, are passive objects that the observer satellites must locate and monitor. The FSS serves as the framework that enables decentralized operation, resource sharing, and communication among the observer satellites, allowing the network to function autonomously with each satellite contributing to the collective mission objectives. Inter-satellite communication is facilitated through Inter-Satellite Link (ISL), enabling the exchange of observation data, orbital parameters, and other mission-critical information. Additionally, each observer satellite must manage its own resources, including energy, data storage, and communication bandwidth, to sustain long-term operations and achieve mission goals.

Operational scenarios within the mission simulate different aspects of the FSS in action. In the target detection and observation scenario, observer satellites autonomously detect and observe target satellites based on their individual capabilities and current operational status. The quality and quantity of observations depend on factors such as orbital alignment, pointing accuracy, and resource availability. Data sharing and propagation occur once a target satellite is detected or observed, with the data being shared across the network to ensure that all participants have access to the latest and most accurate information about the target satellites, even if they have not observed the targets themselves. Resource optimization is another crucial aspect, where observer satellites must optimize their use of resources to maximize the overall mission outcome, including managing energy consumption, data storage, and communication bandwidth. Finally, the dynamic mission adaptation feature of the FSS framework allows the network to adapt to changing conditions, enabling continuous operation in a complex and uncertain environment.

Before delving into more technical details, this chapter tries to clarify and present the main constrains and limitations that have to be addressed in order to get a clear view on the topic.

2.2 Related Work

One of the main motivations behind this thesis is the improvement of autonomous missions, therefore a brief description of traditional mission planning and scheduling is given, as well as some of the most recent DRL approaches that have arisen in the last years. This is essential to understand the main advantage of the application of AI to space systems, giving an extremely flexible, robust and immediate solution to dynamic and opportunistic environments.

2.2.1 Traditional Task Planning and Scheduling

Spacecraft planning and scheduling have traditionally been ground-based processes. During the early phases of mission development and periodically during operations, an iterative process occurs between science planning and mission planning to define the science objectives and spacecraft trajectories at different levels of fidelity. These inputs are then fed into an activity planner, which generates an activity plan detailing the tasks the spacecraft must complete to meet the science objectives. This plan may also incorporate inputs from navigation and flight dynamics teams, depending on the mission. Once generated, the activity plan is sequenced into commands, which are then uplinked to the spacecraft and executed open-loop on board. This process is illustrated in Figure 2.1.

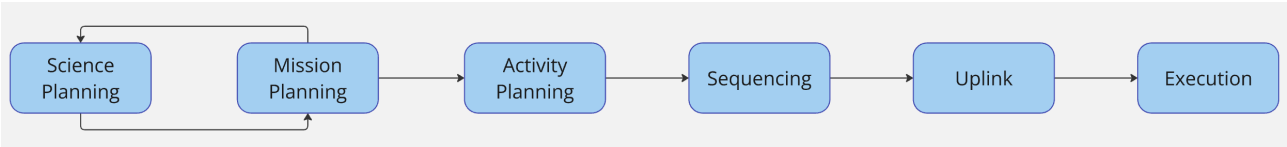


Figure 2.1 Traditional task planning and scheduling process.

This traditional approach introduces several challenges. First and foremost, executing the plan open-loop means the spacecraft cannot react to changes in its environment. If the spacecraft cannot complete a task or meet a constraint, the plan must be regenerated on the ground and then uplinked again, a process that can take several minutes to several hours. This method is also not extensible to opportunistic science events, so if an interesting event is detected, the spacecraft must wait for the next planning cycle to respond.

Additionally, this approach lacks robustness to environmental changes, requiring ground-based regeneration and uplinking of new plans. While this is manageable for missions with high predictability, such as Earth-observing satellites, it poses significant problems for missions with high uncertainty, such as deep space or event detection missions. As a result, robustness must be built into the plan at the cost of science return, efficiency, or quality. Thus, autonomous on-board planning and scheduling algorithms are desirable to reduce reaction times, enable opportunistic science, and increase plan robustness.

Significant efforts have been made to mitigate these challenges by enabling spacecraft to modify ground-based plans in the event of contingencies. One notable solution is CASPER (Continuous Activity Scheduling Planning Execution and Replanning), an on-board planning and scheduling software developed by NASA [9]. CASPER utilizes iterative repair techniques to continually check existing plans for resource constraint violations and modify them as necessary on board the spacecraft. This software, alongside the automated ground-based planning tool ASPEN [10], has been applied to several missions, such as Earth Observing-1 (EO-1) [11] and the Intelligent Payload Experiment (IPEX) [12], demonstrating the capability for on-board schedule modification.

Other tools, such as MEXEC [13] and Aerie [14] developed for the Europa Clipper mission, adjust activities based on resource variations and task execution. The Scheduling Planning Routing Inter-satellite Network Tool (SPRINT) [15], developed by MIT, addresses replanning for Earth-observing satellite constellations using a global planner for scheduling and on-board planners for handling unexpected opportunities.

While these on-board replanning tools address many of the challenges of ground-based planning, they still require an a priori plan generated on the ground and periodically uplinked to the spacecraft. This process remains time-consuming during regular operations. Providing spacecraft with greater control over their operational decisions can reduce the burden on operators, saving time and resources.

Strasser's work [16] focuses on the optimization of scheduling for a heterogeneous Earth observation satellite constellation. His approach involves a centralized model that leverages mathematical optimization techniques to generate efficient schedules. This method emphasizes the use of a global planner to coordinate tasks across multiple satellites, ensuring optimal resource utilization and mission success. In contrast, our approach aims to decentralize decision-making and scheduling, leveraging the capabilities of DRL within the Federated Satellite System paradigm. This decentralized approach eliminates the need for a centralized model, allowing individual

satellites to autonomously adapt and optimize their operations in real-time. By utilizing DRL, each satellite in the network can learn and make decisions based on local observations and interactions, leading to a more flexible and resilient system.

This shift from a centralized to a decentralized model aligns with the strategic goals of enhancing autonomy and efficiency in satellite operations, paving the way for more advanced and adaptive space missions. By comparing these two approaches, it is evident that decentralization through RL offers a promising pathway to overcoming the limitations of traditional and centralized planning methods in satellite scheduling.

2.2.2 Reinforcement Learning Approaches

Reinforcement learning (RL) has emerged as a transformative approach in the field of satellite operations, offering significant advantages in planning and scheduling through its ability to learn and adapt to dynamic environments. RL, a subset of Machine Learning (ML), involves training agents to make a series of decisions by interacting with their environment to maximize cumulative rewards [8], [17]. This method has shown remarkable success in various domains, including robotics [18], game playing [19], and, more recently, space applications [20].

Autonomous decision-making is a critical capability for future satellite missions, particularly those involving Earth observation, scientific research, and communication. Traditional rule-based systems often struggle to handle the complexity and variability of space environments. DRL provides a powerful framework for developing autonomous systems capable of handling these challenges.

DRL techniques have demonstrated state-of-the-art performance in complex decision-making tasks. For instance, Harris et al. [21] describe efforts at Ball Aerospace to develop a DRL-driven solution for the single-satellite, arbitrary-target, single-ground-station planning problem. By leveraging high-fidelity simulations, DRL agents can explore subtle dynamics that might be overlooked by approximation-driven techniques, leading to improved performance in tasking and planning. The process for RL approach on task planning and dynamic scheduling can be seen in Figure 2.2. Here the initial stages of science and mission planning are exactly the same as in the traditional planning approach, however, once the trained agent has been uplinked, it can react to the challenges faced by traditional planning thanks to the closed environment-agent loop.

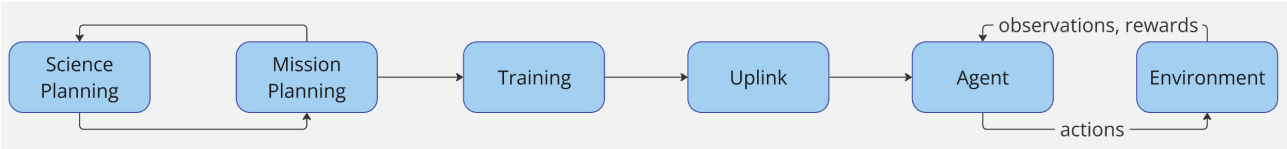


Figure 2.2 Reinforcement Learning approach for task planning and scheduling process.

The application of DRL in multi-agent systems, such as satellite federations, is particularly promising. Ramezani et al. [22] explore the use of MARL for planning and task coordination in a swarm of CubeSats. Their approach addresses the processing limitations of individual CubeSats by allowing them to pool their resources, thereby enabling the execution of complex computations collaboratively. This methodology not only enhances the computational capabilities of CubeSat constellations but also improves their robustness and adaptability in dynamic environments.

Other AI techniques are also increasingly being integrated into satellite networks to address various challenges, like dynamic task scheduling, energy management, and communication optimization. Al Homssi et al. [23] provide an overview of diverse AI applications for next-generation massive satellite networks, highlighting the role of AI in optimizing ISL, Satellite Access Network (SAN), and network security. The proactive and adaptive nature of AI-driven solutions makes them well-suited for managing the complexities of large-scale satellite constellations.

2.3 Federated Satellite System Paradigm and Definition

2.3.1 The Federated Satellite System Paradigm

The Federated Satellite System (FSS) paradigm [3] introduces a revolutionary approach to Distributed Satellite Systems (DSS). This novel concept defines a common network for diverse spacecraft, enhancing their capabilities by providing a pool of resources accessible to any participant of the FSS. This solution enables leveraging the unique abilities of other participants to improve mission outcomes.

In traditional satellite systems, each satellite operates independently with its resources and capabilities. However, the FSS paradigm shifts this approach by creating an interconnected network where satellites can share resources such as downlink bandwidth, processing power, and storage capacity. Other studies such as the Techsat 21 project [24] have expored the use of microsatellites formations to accomplish missions typically performed by larger single satellites. These cooperative models promotes efficiency and flexibility, allowing satellites to perform tasks beyond their individual capabilities by utilizing shared resources within the federations.

These concepts, where satellites autonomously share resources and capabilities, has gathered increasing attention in academia over the past decade. This attention is largely due to advancements in small satellite technologies, such as CubeSats, which make satellite federations more feasible both technically and economically [25]. The FSS paradigm enhances cost-effectiveness, performance, and reliability of space missions by creating a virtual combination of resources that can be dynamically allocated based on mission needs.

A key advantage of the FSS paradigm is its potential to form opportunistic and temporary collaborations among satellites, addressing various technological challenges when implemented in orbit. These collaborations enable the formation of dynamic, reconfigurable networks that can adapt to changing mission requirements and environmental conditions. For instance, satellites in a federation can share their idle resources, such as unused bandwidth or processing power, thereby optimizing the overall performance of the network and improving the mission outcome.

This thesis is built on the foundational concepts of FSS and integrates the decentralized optimization framework developed by Messina [6]. His research on formulating a dynamic graph optimization underscores the need for adaptive, real-time decision-making mechanisms within satellite networks. By applying Deep Reinforcement Learning (DRL) to this framework, our project aims to enhance the autonomy and efficiency of satellite operations. This integration is expected to address the challenges of temporal and spatial variations in satellite networks, enabling more robust and resilient mission planning and execution. Our exploration into DRL for decentralized autonomous decision-making aligns with Messina's vision of adaptive satellite networks and contributes to the broader goal of advancing space exploration and satellite management technologies.

2.3.2 Federated Satellite System Definition

Once the FSS paradigm has been presented, it serves as the base framework for our project. The decentralized nature of the FSS contributes the dynamic and uncertain behaviour of the whole mission. In order to improve the mission outcome, the satellites must learn to interact between them and with the changing FSS environment.

First of all, we must state that there are two types of satellites participating in the simulation: observers and targets. Their specific characteristics and details are presented in the next sections, but it is important to know that they have heterogeneous communication and imaging capabilities as well as different orbits and parameters such as battery or storage.

Working inside the FSS framework, the network is considered as the virtual infrastructure to trade resources among satellites through an inter-satellite link (ISL). Each of the observer satellites is one of the federation network's nodes. These nodes detect and observe target objects in space and then share and propagate information between them. This information, meaning the knowledge communicated or received concerning a particular fact or circumstance [6], can in this case be incomplete, not the current one and can also be obtained indirectly

from another observer satellite. This helps understand the impacts of dynamic partial observable environments in the decision-making or resource optimization processes.

The targets are considered in this project as objects (satellites) in space that have to be discovered and recorded by observers. Each of the observer satellites can share observation data or information on the contacted targets, spreading the information across the satellite network in a real-time manner. The main goal is to optimize the resources so that in the end all observer satellites have information on every target without the need to perform the observations by themselves, but rather in a collaborative mission fashion, making use of the pool of resources. Moreover, here the observers are both customers and suppliers of the FSS.

2.4 Satellite Subsystems, Parameters and Variables

In order to create a realistic environment founded on the FSS, a simulator was developed to model the main mission. As stated in the ConOps, the mission goals mainly include space objects detection and observation and communication and coordination operations. Therefore, in order to correctly model all the interactions and capabilities of the participants, a formal definition of the framework's satellites subsystems has to be given. While in this section the satellites' parts are described, a more detailed description of the higher architectural components of the whole simulation framework is presented in Section 4.

All the values presented in this section were adjusted as necessary for our specific simulation scenarios, but they are fully customizable. Moreover, validation tests are performed for all systems in Section 4.3.

2.4.1 Orbital and Attitude Parameters

In our simulation framework, we model the orbital and attitude dynamics of both observer and target satellites. The following Keplerian parameters are used to ensure realistic and reliable simulation outcomes. A visual description of the parameters is found in Figure 2.3

Orbital Parameters

- **Semimajor Axis**: represents the size of the satellite's orbit, being half the distance between the apoapsis and periapsis, in our case simulating typical Low Earth Orbits (LEO) such as Sun-Synchronous Orbits (SSO). Specific values are varied depending on the scenario to simulate different altitudes and mission profiles.
- **Eccentricity**: describes the shape of the ellipsis, generally kept low to simulate nearly circular orbits, with the exact values adjusted for scenarios involving slight orbital variations.
- **Inclination**: set based on the desired orbital plane, such as SSOs for consistent sunlight exposure or equatorial orbits for different mission needs. The inclination is calculated or selected to match the mission requirements.
- **Right Ascension of Ascending Node (RAAN)**: determines the orbital plane's orientation relative to the Earth's equatorial plane, adjustable to simulate different timing and coordination requirements within the satellite network.
- **Argument of Perigee and True Anomaly:** define the orbit's orientation and the satellite's position within the orbit, varied to test different initial conditions and orbital phases.
- **Mean Motion**: the rate at which the satellite orbits the Earth, calculated based on the semimajor axis and gravitational parameter.

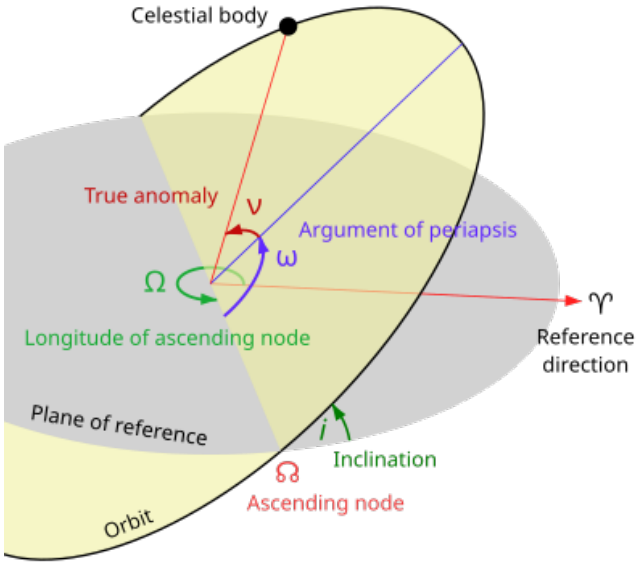


Figure 2.3 Diagram of orbital parameters.

Attitude Parameters

- **Quaternion**: represents the satellite's orientation in three-dimensional space. The initial quaternion can be set to represent various starting orientations, with adjustments made as needed for specific simulation requirements.
- **Angular Velocity**: defines the rotation rate of the satellite about its axes, adjustable to simulate different rotational states and stabilization requirements.

Orbital Propagation

The satellite's orbit is propagated using Keplerian mechanics, which is an essential method for simulating the motion of celestial bodies under the influence of gravity. This method relies on solving Kepler's equation to compute the position and velocity of the satellite in its orbit over time. The propagation begins by defining the orbital elements, including the semimajor axis a, eccentricity e, inclination i, argument of perigee ω, right ascension of the ascending node Ω, and the true anomaly θ.

The mean motion n of the satellite, which represents the angular speed at which the satellite orbits the Earth, is calculated using the formula:

$$n = \sqrt{\frac{\mu}{a^3}} \tag{2.1}$$

where $\mu = 3.986004418 \times 10^{14}$ m^3/s^2 is the standard gravitational parameter for Earth.

Kepler's equation is then solved to find the eccentric anomaly E, which is related to the true anomaly θ and the orbital eccentricity e by:

$$E = \arctan\left(\frac{\sqrt{1-e^2}\sin\theta}{e+\cos\theta}\right) \tag{2.2}$$

The mean anomaly M is determined from:

$$M = E - e \sin E \tag{2.3}$$

The time evolution of the orbit is computed by incrementing the true anomaly θ using the rate of change $\dot{\theta}$, which is given by:

$$\dot{\theta} = \sqrt{1 - e^2} \cdot \frac{n}{1 - e \cos E} \tag{2.4}$$

The new radius vector r, which represents the distance from the center of the Earth to the satellite, is updated using:

$$r = \frac{a(1 - e^2)}{1 + e \cos \theta} \tag{2.5}$$

Finally, the position (x, y, z) and velocity $\vec{v} = (v_x, v_y, v_z)$ vectors of the satellite are calculated in the inertial reference frame by combining the orbital elements and trigonometric functions, ensuring the accurate propagation of the satellite's orbit over time.

Attitude Propagation

Attitude propagation is performed using quaternion kinematics, which provides a robust method for simulating the orientation of the satellite in three-dimensional space. The satellite's orientation is represented by a quaternion, a four-dimensional vector that encodes rotation in a way that avoids the singularities associated with Euler angles.

The quaternion's time evolution is governed by the satellite's angular velocity $\vec{\omega} = (\omega_x, \omega_y, \omega_z)$. The rate of change of the quaternion, $\dot{q}$, is calculated using the following differential equation:

$$\dot{q} = \frac{1}{2} \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix} q \tag{2.6}$$

This equation is integrated using an Euler integration step, where the quaternion is updated as:

$$q_{\text{new}} = q_{\text{old}} + \dot{q} \cdot \Delta t \tag{2.7}$$

where Δt is the time step for propagation.

To maintain the quaternion's unit norm (ensuring it remains a valid rotation), the quaternion is normalized after each update:

$$q_{\text{new}} = \frac{q_{\text{new}}}{\|q_{\text{new}}\|} \tag{2.8}$$

This normalization process is crucial as it prevents numerical errors from accumulating over time, which could otherwise lead to drift in the satellite's simulated orientation. By applying these quaternion kinematics, the satellite's attitude is accurately propagated, allowing for precise control and stabilization of the satellite's orientation during the mission.

2.4.2 Communication Subsystem

Observer satellites are the only ones with communication capabilities. They are equipped with an omni-directional antenna, based on Line-of-Sight (LOS). The simulation assumes negligible latency for simplicity but includes provisions to simulate realistic delays as needed.

The communication subsystem of the satellites is crucial for data exchange and coordination among federation participants. Each satellite is equipped with a communication system based on Scrocciolani's work [26] with further adjustements made by Messina [6], and it is characterized by the following features and values depicted in Table 2.1:

Parameter	Value	Unit
Frequency	437	MHz
Bandwidth	9600	Hz
Transmit Power	2	W
Transmit Gain	1	dB
Receive Gain	1	dB
System Losses	3.5	dB
Receiver Sensitivity	-151	dBW
Modulation Scheme	QPSK	-

Table 2.1 Link Budget for Communication Subsystem

- **Communication Band**: satellites are additionally equipped with different communication bands, randomly selected from the four options of Ultra High Frequency (UHF), Very High Frequency (VHF), S-band, or X-band, with the possibility of having all bands available for complete communication capabilities (for instance for the central node of the network).
- **Power and Gain**: the communication system operates with a transmission power of 2 W, and both transmission and reception gains are accounted for, with gains set to 1 dB, and corresponding losses included.
- **Frequency and Bandwidth**: the system operates at a frequency of 437 MHz, with a bandwidth of 9600 Hz, supporting various modulation schemes, including Quadrature Phase Shift Keying (QPSK).
- **Modulation and Sensitivity**: the system supports a modulation order of 4 (QPSK) and has a sensitivity of -151 dBW, ensuring robust communication under various conditions.

The communication subsystem includes calculations for the following variables:

- **Free Space Loss (FSL)**: determined by the distance between communicating satellites and their operating frequency.
- **Signal-to-Noise Ratio (SNR)**: the system calculates SNR based on the power, gains, losses, and FSL, which is critical for determining data transmission quality.
- **Data Rates and Bit Error Rate (BER)**: the system computes both ideal and effective data rates, considering modulation efficiency and channel conditions. BER calculations assess communication reliability.

The maximum distance to communicate taking into account our specifications and conditions is around 2,315 km, limited by the effective data rate calculation.

2.4.3 Information Sharing

Communication between observers is conducted in both directions. This means that after two satellites have successfully communicated, regardless of which one initiated the contact, both receive the same information on other observers and targets they have contacted.

Successful data transmission is achieved when the total amount of data transmitted exceeds the total amount of data to be transmitted. This is determined by estimating the data that needs to be transmitted, which depends on the amount of new information collected by each observer. Additionally, the transmission rate used in these calculations is estimated based on FSL.

Target data is only considered valid if the target has been contacted in recent timesteps, ensuring that information on target satellites remains accurate and timely. This approach helps maintain the federation's activity in detecting and observing targets.

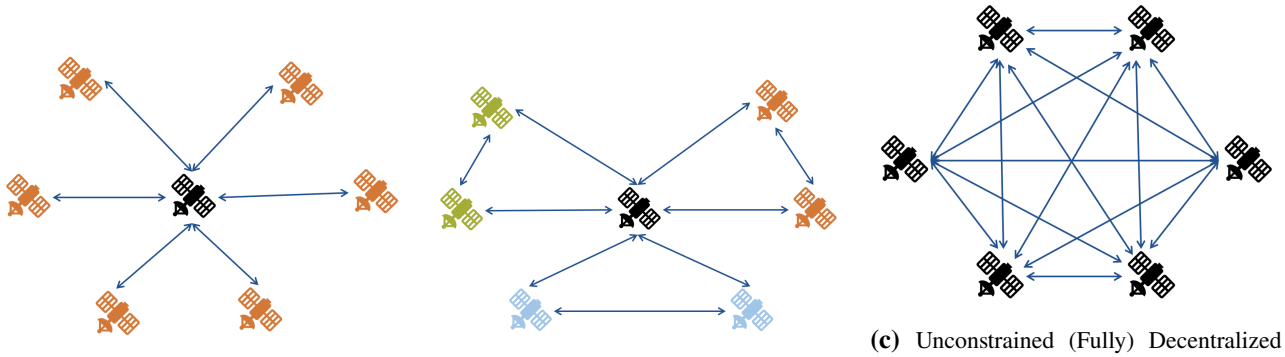
The following information can be shared and gathered by each observer participant:

- **Orbital Parameters**: both self and from contacted observer satellites, providing current positioning and motion details.
- **Communication Ability**: towards other observer satellites, determining the potential for successful data exchange.
- **Battery Level**: information on the power status of both the observer and recently contacted observers, important for energy management and planning.
- **Storage Level**: data storage status for both the observer and recently contacted observers, critical for data management and prioritization.
- **Pointing Accuracy**: equivalent to the quality of data collected, shared among the network to provide a measure of observation quality.
- **Target Orbital Parameters**: data about the orbits of recently contacted target satellites, crucial for tracking and further observations.
- **Observation Status**: indicates whether targets have been detected or observed, helping prioritize future observation tasks.
- **Communication Status**: status of communication links between observers, used for managing network connectivity and data flow.

2.4.4 Coordination Models

As stated, communication bands may vary among observers, allowing for the development of different, custom coordination types for the environment:

- **Centralized Communication**: only one node of the FSS network has a universal communication band. The rest of the participants must transmit information to this central node for further propagation.
- **Decentralized Communication with Band Constraints**: each participant can only communicate with others sharing the same communication band. Central nodes (equipped with all bands) can be introduced to increase the complexity, providing a more challenging setup.
- **Unconstrained Decentralized Communication**: every node can transmit information to any other node within the maximum distance limits specified by the communication system's technical components. This mode is exactly the same as the decentralized one, but without the requirement of having the same band as the other node.



(a) Centralized Coordination

(b) Partially Decentralized Coordination

(c) Unconstrained (Fully) Decentralized Coordination

Figure 2.4 Comparison of Coordination Models

These models support various coordination scenarios, from centralized to fully decentralized systems, as can be seen in Figure 2.4, allowing flexibility in the simulation of different network architectures and operational strategies.

2.4.5 Imaging and Observation Capabilities

The optical payload on the observer satellites is designed to capture data on space events and objects. The following details describe the imaging system and its operational parameters.

Each observer satellite is equipped with an optical payload characterized by the following [27]:

- **Aperture Diameter:** 9 mm, allowing sufficient light gathering for imaging in various conditions.
- **Wavelength**: 700 nm, representing the upper limit of the visible light spectrum, typically used for optical observations.
- **Object Size Resolution**: the system is calibrated to resolve objects of specified dimensions (5 meters in our case, to mimic Starlink size satellites detection).

The observation process is a critical aspect of the satellite's operation, encompassing several essential steps. Initially, the satellite evaluates both the distance to the target and the pointing accuracy. The maximum distance for effective observation is governed by the diffraction limit of the optical system.

To assess pointing accuracy, the satellite calculates its pointing direction and the vector from the observer to the target satellite. The cosine of the angle between these vectors is then computed to determine the angular distance. If this distance falls within a total 20-degree field of view and the target is within the maximum detectable distance (see Figure 2.5), the pointing accuracy is normalized on a scale from 0 to 1, with higher values indicating better alignment.

The diffraction limit is calculated using the formula:

$$\text{Diffraction Limit} = 2.44 \cdot \frac{\lambda \cdot d}{D} \tag{2.9}$$

where λ is the wavelength of light (700 nm in this case), d is the distance to the target, and D is the aperture diameter of the optical system (0.09 m).

To determine the maximum detectable distance, adjustments to the diffraction limit equation are made:

$$\text{Max Detectable Distance} = \frac{\text{Object Size} \cdot D}{2.44 \cdot \lambda} \tag{2.10}$$

This calculation ensures that only targets within the detectable range and with sufficient pointing accuracy are observed. For instance, assuming a Starlink-sized object, the limit distance is just over 263 km. A visualization of the field of view and estimated limit distance is depicted in Figure 2.5.

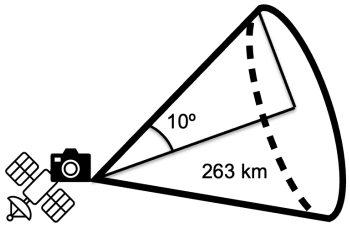


Figure 2.5 Satellites' imaging field of view and limit distance.

Upon a successful observation, the satellite updates the observation status matrix and records the cumulative pointing accuracy. Additionally, the observer's available storage is adjusted based on the data collected during the observation process.

The optical payload's performance metrics, such as pointing accuracy and the number of observations, are meticulously recorded. These metrics are then used to refine observation strategies and optimize resource utilization, ensuring efficient and accurate data collection in a dynamic and challenging space environment.

2.4.6 Energy Management

The energy management system on each observer satellite ensures continuous operation by balancing energy consumption and generation. The following components and processes describe how energy is managed:

Each satellite is equipped with a battery system and solar panels for energy storage and generation:

- **Energy Storage Capacity**: the energy storage system uses a battery pack with a capacity of 84 Wh, sufficient for maintaining satellite operations during eclipse periods when solar energy is unavailable.
- **Solar Panels**: the solar panels have a total surface area of 0.12 m^2 (0.4 m $\times$ 0.3 m), which are deployable and capable of harnessing solar energy. The efficiency of these panels is set at 30%, with the solar constant assumed to be 1,370 W/m^2.
- **Energy Available**: at the start, each satellite is assigned an amount of energy available, which is randomly generated between 20 and 70 Wh.

The charging of the satellite's battery is governed by the exposure to sunlight. Sunlight exposure is determined by the satellite's position relative to the Sun and whether the satellite is in the Earth's shadow (eclipse). This exposure is further modulated by the angle between the satellite's pointing direction and the Sun vector.

The energy produced by the solar panels is calculated using Equation 2.11:

$$\text{Energy Produced} = \text{Solar Panel Size} \times \text{Efficiency} \times \text{Solar Constant} \times \text{Sunlight Exposure} \times \text{Time Step} \quad (2.11)$$

This produced energy is then added to the battery's energy reserves, ensuring the total does not exceed the maximum capacity or falls below zero.

The power consumption of the satellites varies depending on the operational mode [28]:

- **Standby Mode**: the baseline power consumption for maintaining satellite systems operational is 7.5 W. This amount includes 0.2 W that correspond to the communication subsystem, which in total consumes 2 W.

- **Communication Mode**: when the satellite is actively communicating, the power consumption increases to 9.3 W, accounting for the additional energy required for data transmission and reception.
- **Observation Mode**: during active observation, the power consumption is highest, at 18.806 W, reflecting the energy-intensive nature of imaging and data processing.

The power consumption is subtracted from the available energy in the battery according to the current operational mode, ensuring that the satellite's energy levels are continually monitored and managed. This management includes handling situations where energy reserves are depleted or when the satellite needs to enter low-power modes to conserve energy.

This energy management system ensures that the satellites can sustain operations, handle dynamic power requirements, and maximize the utilization of available solar energy.

2.4.7 Data Storage and Management

The data storage system onboard each observer satellite is designed to handle the data collected during observations and communications. The following components and processes describe how data is managed and stored:

Each satellite is equipped with a storage system characterized by the following:

- **Total Data Storage Capacity**: the maximum storage capacity is 64 GB, not sufficient for storing extensive observation data and other mission-related information, increasing the complexity of the simulations.
- **Initial Storage Available**: at the start of the mission, each satellite is assigned an amount of storage available, which can be randomly generated.
- **Data Package Size**: the size of each data package per observer satellite is standardized at 52 bytes, facilitating consistent data handling and transmission, while the amount of target data transmitted depends on the state of the simulation. The size of 52 bytes takes into account the orbital parameters together with the availability boolean, battery, storage, pointing accuracy data and communication ability data.

The rate at which data storage is consumed depends on the satellite's current activities:

- **Observation Mode**: during active observation, the data collected is stored at a fixed rate of 1 Mbits/s ($1024 \times 1024 \times 8$ bits per second), which accounts for high-resolution data and metadata.
- **Communication Mode**: during the communication process, storage is not directly consumed by the act of transmitting data. Instead, storage consumption occurs after the data has been successfully transmitted. At that point, the amount of transmitted data is deducted from the available memory storage. Once transmission is finished and the data is accounted for at the receiving end, the storage is not freed up in the sender's end. As a result, the restrictive availability of storage may impact the satellite's ability to collect and store new data.

Data handling during operations involves several crucial processes. As data is collected during observations, it is stored in the satellite's onboard memory. The amount of storage available is continuously updated by subtracting the data stored per time unit (time step) from the available storage.

During communication, data is transmitted to other satellites. The storage available on the sending satellite is decremented by the amount of data transmitted. This process ensures that storage capacity is accurately reflected, and any data received by other satellites is accounted for in their respective storage systems.

The system ensures data integrity by preventing storage overflows. When the available storage reaches its maximum capacity, new data collection or reception is stopped.

The storage management system is critical for maintaining data integrity and ensuring that the satellites can continue to collect and transmit data efficiently throughout the mission. Proper handling of data storage also

allows for effective mission planning and real-time decision-making, leveraging the full capabilities of the satellite's onboard systems.

3 Reinforcement Learning in Satellite Operations

In the context of federated satellite systems, the complexity of coordinating multiple satellites to achieve mission objectives necessitates advanced decision-making frameworks. Traditional methods, as seen in Section 2.2, while effective in certain scenarios, struggle to adapt to the dynamic and unpredictable nature of space environments. As the demand for real-time, autonomous operations increases, it becomes imperative to explore more sophisticated approaches that can enhance both the efficiency and resilience of satellite networks. This is where Reinforcement Learning (RL) becomes invaluable. By providing a structured approach to learning optimal strategies from the developed environment, RL offers a powerful tool for addressing the challenges inherent in decentralized satellite operations. The following section delves into the principles of RL and how it can be applied to develop autonomous systems capable of making real-time decisions in complex environments.

3.1 Introduction to Reinforcement Learning

In the realm of satellite operations, RL provides a framework for developing autonomous systems capable of making real-time decisions. Reinforcement learning has its origin from using machine learning to optimally control an agent in an environment.

The goal of reinforcement learning is to learn an optimal policy, a policy that achieves the maximum expected reward from the environment when acting. The reward is a single dimensionless value that is returned by the environment immediately after an action. The whole process can be visualized in Figure 3.1.

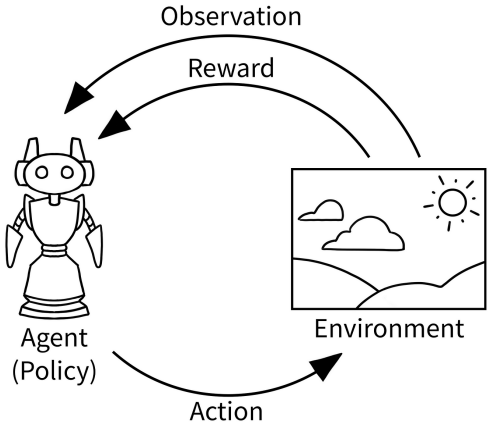


Figure 3.1 Reinforcement learning process [29].

The agent learns a policy, a function that maps an observation obtained from its environment to an action. Policy functions are typically a Deep Neural Network (DNN) (with several hidden layers, see Figure 3.2), therefore giving name to "deep" reinforcement learning [29].

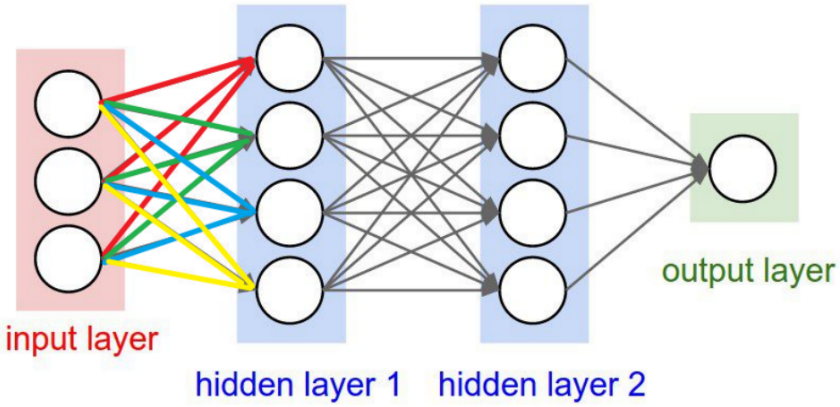


Figure 3.2 Neural network example [30].

Traditional methods like Mixed-Integer Programming (MIP) often face limitations in terms of computational feasibility and adaptability to changing conditions, especially in dynamic environments like satellite task scheduling. RL, on the other hand, excels in such settings by learning optimal policies through continuous interaction with the environment [31].

The foundational concept in RL is the Markov Decision Process (MDP), which formalizes and describes the problem of decision-making in stochastic environments [32]. An MDP is defined by a set of states, actions, transition probabilities, and rewards. In the context of satellite operations, an MDP can model the decision-making process for tasks such as imaging, communication, and maneuvering. When full state observability is not possible, the problem extends to Partially-Observable Markov Decision Process (POMDP), where decisions are made based on incomplete information [17].

In DRL, contrary to RL, state and action spaces can be mapped with a function or value, let it be $Q(s, a)$, that returns a value (reward). In DRL, where problems become more complex, this Q-value is approximated with an Artificial Neural Network (ANN). The ANN may need to be "deep", meaning a few hidden layers may not suffice to capture all the intricate details of that knowledge, hence the use of DNN.

Finally, in this chapter, three of the most extended and used DRL algorithms are presented: PPO, DQN and SAC. These are the algorithms applied to our case and each of them is a state-of-the-art tool to obtain DNNs for complex problems.

3.2 Markov Decision Processes and Reward-Driven Behavior

Markov Decision Processes (MDPs) [32], [33] are fundamental to the field of Reinforcement Learning and provide a formal framework for modeling decision-making in situations where outcomes are partly under the control of a decision-maker and partly random. MDPs are defined by a tuple (S, A, P, R, γ) where:

- S represents the set of all possible states the system can be in.
- A is the set of actions available to the decision-maker.
- R is the reward function, $R(s, a)$, giving the immediate reward received after transitioning from state s to state s' after action a.
- R is the reward function, which assigns a numerical value to each state-action pair, guiding the agent's learning by indicating the desirability of different outcomes.
- γ is the discount factor, which reflects the importance of future rewards compared to immediate ones.

In satellite operations, MDPs can model various scenarios, such as scheduling tasks, optimizing fuel usage, or managing communications. For example, the state might represent the satellite's current position, battery level, and task queue, while actions could involve choosing the next task, adjusting orbit, or entering a power-saving mode. The reward function in this context is designed to maximize the operational efficiency and mission objectives of the satellite.

The process is as follows: a decision-making agent takes an action a_i while in state s_i. The decision-making agent transitions to a new state s_{i+1} and receives a reward r_i [20] (see Figure 3.3).

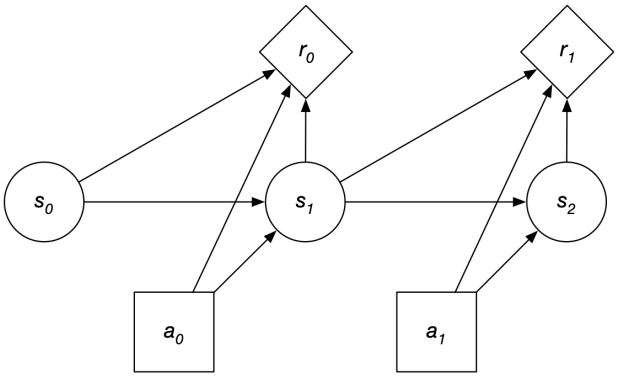


Figure 3.3 MDP process.

MDPs assume full observability, meaning the agent has access to the complete state of the environment. However, in many practical scenarios, including satellite operations, agents often operate under partial observability, where they cannot access all relevant state information. This leads to the use of Partially Observable Markov Decision Processes (POMDPs), which are extensions of MDPs designed to handle situations where the agent must make decisions based on incomplete or uncertain information [21]. A POMDP is defined by a tuple $(S, A, P, R, \gamma, O, Z)$ where:

- O is the set of observations.
- Z is the observation probability matrix, $Z(o|s', a)$, representing the probability of observing o given the next state s' and action a.

In a POMDP, the agent maintains a belief state, $b(s)$, which is a probability distribution over possible states based on the history of actions and observations. The objective is to find a policy $\pi : B \rightarrow A$ that maximizes the expected cumulative reward, where B is the set of all possible belief states.

In more complex systems involving multiple agents like ours, the problem becomes even more challenging. In these cases, decentralized partially observable Markov Decision Process (Dec-POMDP) is employed. Dec-POMDPs account for the fact that each agent has only partial information and must make decisions that consider the potential actions of other agents. This adds layers of complexity due to the need for coordination and the increased computational challenges, as Dec-POMDPs are known to be of high computational complexity ($2^{n^{O(1)}}$) [34]. A Dec-POMDP is defined by a tuple $(N, S, \{A_i\}, P, \{R_i\}, \gamma, \{O_i\}, Z)$ where:

- N is the number of agents.
- S is the set of states.
- $\{A_i\}$ is the set of actions for each agent i.
- P is the transition probability matrix.
- $\{R_i\}$ is the reward function for each agent i.
- γ is the discount factor.

- $\{O_i\}$ is the set of observations for each agent i.
- Z is the observation probability matrix.

The objective in a Dec-POMDP is to find a set of policies $\{\pi_i\}$ (Equation 3.1) that maximize the expected cumulative reward for all agents:

$$\{\pi_i^*\} = \arg\max_{\{\pi_i\}} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \sum_{i=1}^{N} R_i(s_t, a_{i,t}) \mid \{\pi_i\}\right] \tag{3.1}$$

In the context of satellite operations, observations are dynamic and partial. An agent (satellite) may or may not have information on another observer (satellite) or target at any given time. This scenario fits well within the POMDP and Dec-POMDP frameworks, where decision-making must account for incomplete and dynamically changing information.

3.3 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) integrates deep neural networks into the RL framework, allowing for the handling of high-dimensional state and action spaces. This integration is particularly crucial for satellite systems, where the complexity of tasks and the variability in the environment are significant. DRL algorithms, such as PPO, DQN, and SAC, provide robust mechanisms for policy optimization by leveraging the powerful function approximation capabilities of deep neural networks. These algorithms enable agents to make informed decisions based on a richer understanding of the environment, accommodating the dynamic and uncertain nature of satellite operations [31].

The use of frameworks like Ray RLlib, which offers scalability and flexibility, is quite useful in deploying DRL algorithms in practical applications. This framework supports various RL algorithms and facilitates efficient training across distributed systems, making it an ideal choice for large-scale problems like those encountered in satellite operations [35] [36].

3.3.1 Q-Values and Deep Reinforcement Learning

The transition from traditional RL to DRL involves the use of Deep Neural Networks (DNNs) to approximate complex functions such as the Q-value in Q-learning. In DRL, neural networks are used to represent policies or value functions, enabling the handling of high-dimensional state and action spaces that are infeasible with traditional tabular methods. This is particularly relevant in satellite operations, where the state space can include continuous variables like position, velocity, and resource levels [21].

The Q-value function, or Q-function, is essential to many RL algorithms, including those used in DRL. The Q-function $Q(s, a)$ represents the expected cumulative reward of taking action a in state s. In deep Q-learning, a neural network, often called a Q-network, approximates this function. The network takes the state as input and outputs Q-values for all possible actions. This allows the agent to select actions that maximize the expected reward [30].

The use of DNNs in approximating Q-values and policies is essential for scaling RL to real-world problems with complex and continuous state and action spaces. For instance, in satellite operations, the state space might include variables such as orbital parameters, battery levels, and communication status, while the action space could involve decisions about maneuvers, communication scheduling, and data collection tasks [21], [20].

3.3.2 Hyperparameter Tuning

Hyperparameter tuning is a crucial stage in the deep reinforcement learning pipeline. The goal is to find the optimal set of hyperparameters that enable the model to converge to a good minimum and generalize well on unseen data. Common hyperparameters include learning rate, batch size, number of epochs, optimizer type, size of hidden layers, and activation functions.

While tuning these parameters can be daunting due to the large search space and the time-consuming nature of training deep learning models, there are well-established strategies to streamline this process.

For the learning rate, utilizing a scheduler can help mitigate issues related to selecting a suboptimal initial learning rate. These schedulers adjust the learning rate during training to maintain optimal convergence. The choice of activation functions can significantly impact the performance and convergence of neural networks, with ReLU (Rectified Linear Unit) and its variants being commonly used due to their simplicity and effectiveness. When selecting an optimizer, Adam and its variants are often a good starting point due to their robustness and ability to handle sparse gradients. For the size of hidden layers, it's advisable to start with a smaller number of hidden layers and units, scaling up as needed based on model performance and computational resources [30], [37].

Given the vastness of the hyperparameter space, manual tuning can be inefficient. To address this, several automated methods are commonly employed [30], [37]:

- **Grid Search**: this method involves systematically exploring a predefined grid of hyperparameter values. While comprehensive, grid search is often impractical for large hyperparameter spaces due to its computational intensity.
- **Random Search**: a more efficient alternative, random search samples hyperparameters from specified distributions. This method typically explores the hyperparameter space more effectively than grid search, as it does not require exhaustive evaluation of all possible combinations.
- **Bayesian Optimization**: this advanced technique balances exploration and exploitation of the hyperparameter space. It models the function to be optimized and iteratively selects hyperparameters to evaluate, using prior results to make informed decisions. Despite its sophistication, Bayesian optimization can be computationally demanding and may not always provide significant improvements over random search.

The choice of hyperparameter tuning method depends on the specific requirements and constraints of the project, including computational resources and time availability. For instance, in environments where training is expensive or time-consuming, methods like Bayesian or ASHA optimization might offer a better trade-off between search efficiency and computational cost.

3.3.3 Training

In the context of Deep Reinforcement Learning, training refers to the iterative process through which an agent learns to optimize its policy or value function by interacting with an environment. The training phase is crucial as it determines the effectiveness of the agent in achieving its goals, such as optimizing resource usage or maximizing mission outcomes in satellite operations.

The training process in DRL involves several key steps [8], [37]:

- **Experience Collection**: the agent interacts with the environment to collect experiences, which are tuples of the form (s, a, r, s'). Here, s represents the state, a the action taken, r the reward received, and s' the subsequent state. These experiences are stored in a replay buffer, which helps in breaking the correlation between consecutive experiences and stabilizes training.
- **Policy Update**: using the collected experiences, the agent updates its policy or value function. In policy-based methods like Proximal Policy Optimization (PPO), the policy is updated directly to maximize

expected rewards. In value-based methods like Deep Q-Networks (DQN), the value function $Q(s,a)$ is updated to minimize the difference between predicted and actual rewards.

- **Exploration vs. Exploitation**: a crucial aspect of training in DRL is balancing exploration (trying new actions to discover their effects) and exploitation (choosing actions that are known to yield high rewards). Strategies like epsilon-greedy in DQN or entropy regularization in Soft Actor-Critic (SAC) help maintain this balance.
- **Parallelization and Scalability**: DRL training can be computationally intensive, especially with high-dimensional state and action spaces. Techniques such as parallelization are employed to speed up the training process. This involves distributing the workload across multiple processing units or machines. For example, multiple environments can be simulated in parallel, with each environment running independently to collect experiences simultaneously.

The training process in DRL is iterative and involves continuous refinement of the agent's policy. This process is computationally demanding but crucial for developing robust and effective agents capable of performing complex tasks in dynamic environments, such as satellite constellation management.

3.3.4 Deep Q-Networks

Deep Q-Networks (DQNs) represent a significant advancement in the application of neural networks within reinforcement learning, specifically for tasks involving high-dimensional state spaces. DQNs approximate the action-value function $Q(s,a)$, which estimates the expected return of taking an action a in state s under a certain policy. This approach allows the agent to learn which actions yield the highest rewards over time [38].

The core idea behind DQNs is to use a deep neural network to generalize the Q-value function across a wide range of states, rather than relying on a simple lookup table. This is crucial for handling large and continuous state spaces where traditional methods fall short.

Key components of DQN include [39]:

- **Experience Replay**: to break the correlation between consecutive samples, DQNs use a mechanism called experience replay. This involves storing the agent's experiences (s,a,r,s') in a replay buffer and randomly sampling from this buffer during the training process. This random sampling helps in stabilizing the training process and reduces the risk of oscillations or divergence. The buffer is periodically updated to ensure it doesn't grow indefinitely, usually by removing the oldest experiences.
- **Target Network**: DQNs employ a target network, a copy of the original Q-network, to compute the target Q values. The weights of the target network are updated less frequently than those of the primary network, providing a stable target for learning and helping to mitigate the instability that arises from frequently changing targets.
- **Double DQN**: an improvement over the basic DQN, Double DQN addresses the issue of overestimation bias in the action-value function. It decouples the selection and evaluation of actions by using the primary network to select the action and the target network to evaluate it. This helps in obtaining more accurate value estimates.

DQNs are classified as off-policy algorithms, meaning they can learn the optimal policy independently of the agent's current actions. This is advantageous for environments where exploration and exploitation must be balanced effectively.

In the context of satellite operations, DQNs are particularly useful for making decisions in environments where the state space is large and continuous, such as managing energy resources, scheduling tasks, and optimizing communication links [20]. Algorithm 1 provides the full pseudo-code for DQN [39]

Algorithm 1: Deep Q-Learning [39]

Initialize replay buffer $\mathcal{D}$
Initialize state-action value function $Q_\theta(s,a)$ with random weights θ
Initialize target state-action value function $\hat{Q}_{\theta^-}(s,a)$ with weights $\theta^- \leftarrow \theta$
for iteration = 1 **to** N **do**
  for $i = 1$ **to** $|I|$ **do**
    for actor = 1 **to** M **do**
      $$a_i \leftarrow \begin{cases} \arg\max_a Q_\theta(s_i,a) & \text{with probability } 1-\epsilon \\ \text{random action} & \text{otherwise} \end{cases}$$
      $s_{i+1}, r_i \sim G(s_i, a_i)$
      Store transition (s_i, a_i, r_i, s_{i+1}) in $\mathcal{D}$
      Sample random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from $\mathcal{D}$
      $$y_j = \begin{cases} r_j & \text{if } s_{j+1} \text{ is terminal} \\ r_j + \gamma \max_{a'} \hat{Q}_{\theta^-}(s_{j+1}, a') & \text{otherwise} \end{cases}$$
      Perform gradient descent step on $(y_j - Q_\theta(s_j, a_j))^2$ **with respect to** θ
      Periodically reset $\hat{Q}_{\theta^-} \leftarrow Q_\theta$
    end for
  end for
end for

3.3.5 Soft Actor-Critic

Soft Actor-Critic (SAC) is an advanced off-policy actor-critic algorithm that has gained attention for its effectiveness in continuous action spaces, although it can be used in discrete ones too [38]. SAC integrates entropy regularization into the reinforcement learning framework, encouraging exploration by maximizing both the expected reward and the entropy of the policy. This balance between exploration and exploitation helps the agent avoid suboptimal deterministic policies [39], [40].

The key components of SAC include [39]:

- **Policy Network (Actor)**: generates a stochastic policy $\pi(a|s)$, which outputs actions given the state.
- **Q-Value Networks (Critics)**: two Q-value networks, Q_1 and Q_2, are used to evaluate the expected return for state-action pairs, reducing overestimation bias.
- **Entropy Regularization**: the inclusion of an entropy term $\alpha\mathcal{H}(\pi(\cdot|s))$ in the objective function, where α is a temperature parameter, encourages a diverse set of actions, enhancing exploration. The temperature parameter α can be automatically adjusted during training to maintain a desired level of exploration.

The SAC training process involves updating the policy and Q-functions iteratively. The objective for the policy update incorporates the expected Q-value and an entropy term, as shown in Equation 3.2:

$$J(\pi) = \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[\min_{i=1,2} Q_i(s,a) - \alpha \log \pi(a|s)\right] \tag{3.2}$$

SAC's off-policy nature allows it to use data from outside the current policy distribution, improving sample efficiency and robustness. The Q-functions are updated to minimize the Bellman residual, and the policy is updated using a gradient method to maximize the expected reward while considering the entropy bonus. SAC's use of both Q-functions and entropy regularization enhances policy robustness and sample efficiency, making it suitable for complex environments like satellite operations. Algorithm 2 shows the logic of this method [39].

Algorithm 2: Soft Actor-Critic [39]

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
repeat
  Observe state s and select action $a \sim \pi_\theta(\cdot|s)$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  if s' is terminal **then**
    Reset environment state
  end if
  if it's time to update **then**
    for j in range(however many updates) **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets for the Q functions:

$$y(r, s', d) = r + \gamma(1 - d)\left(\min_{i=1,2} Q_{\text{targ},i}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right), \quad \tilde{a}' \sim \pi_\theta(\cdot|s')$$

      Update Q-functions by one step of gradient descent:

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2 \quad \text{for } i = 1, 2$$

      Update policy by one step of gradient ascent:

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s)|s)\right),$$

      where $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot|s)$ which is differentiable w.r.t. θ via the reparameterization trick.
      Update target networks:

$$\phi_{\text{targ},i} \leftarrow \rho\phi_{\text{targ},i} + (1 - \rho)\phi_i \quad \text{for } i = 1, 2$$

    end for
  end if
until convergence

3.3.6 Proximal Policy Optimization

Proximal Policy Optimization (PPO) is an on-policy reinforcement learning algorithm that aims to improve the stability and reliability of policy updates. PPO uses a surrogate objective function to restrict the extent to which the new policy can deviate from the old policy, preventing large updates that could lead to divergence [38], [37]. This makes PPO particularly robust for environments with highly variable rewards, such as satellite operations.

PPO combines the benefits of Trust Region Policy Optimization (TRPO) and simpler optimization methods, achieving a balance between performance and computational efficiency. It uses a clipped surrogate objective to limit the policy updates, which can be seen in Algorithm 3 [39], [41].

PPO is classified as an on-policy algorithm, meaning it uses the data generated by the current policy to update the policy itself. This requires careful handling of exploration, as the data collection and policy improvement steps are closely tied, ensuring that updates are based on the most recent policy interactions.

Algorithm 3: Proximal Policy Optimization [39]

1: **Input:** initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} \hat{A}^{\pi_{\theta_k}}(s_t, a_t), g(\epsilon, \hat{A}^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.
7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.
8: **end for**

3.3.7 Multi-Agent Problems

In multi-agent reinforcement learning (MARL), multiple agents interact within a shared environment, often with overlapping goals or resources (Figure 3.4). This approach is especially useful in scenarios like our multi-satellite environment, where coordination between satellites can enhance system performance.

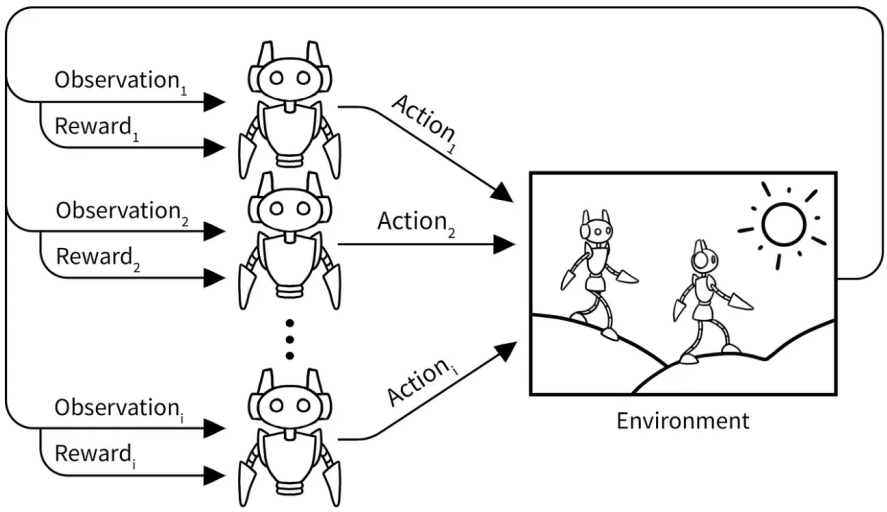


Figure 3.4 Multi-agent reinforcement learning process [29].

Cui et al. [42] utilized double Deep Q-Networks for communication scheduling in a constellation of Earth-orbiting satellites, demonstrating superior performance compared to genetic algorithms in terms of computation time and efficiency. Dalin et al. [43] applied the Multi-agent Deep Deterministic Policy Gradient (MADDPG)

algorithm to address multi-satellite tasking, achieving results comparable to other solvers. However, both studies faced limitations in handling resource constraints and scaling algorithms beyond fixed constellation parameters during training.

MARL encompasses a variety of problem formulations and algorithms tailored to multi-agent decision-making challenges. Key aspects of MARL include different reward structures—fully cooperative, competitive, and mixed cooperative/competitive—and training methodologies. For instance, one can model cooperative training by enhancing global rewards when certain objectives are achieved. In the case of competitive environments, a negative reward can be applied to agents that did not accomplish the goals or were beaten by other agents. These methodologies range from independent learning, where each agent learns its own policy, to centralized learning, where a joint policy is learned for the entire action space, and Centralized Training with Decentralized Execution (CTDE), which balances the strengths of centralized and decentralized approaches.

Independent learning methods like Independent Q-learning (IQL), Independent Advantage Actor-Critic (IA2C), and Independent Proximal Policy Optimization (IPPO) face challenges such as non-stationarity and partial observability, yet they remain practical solutions. In contrast, centralized approaches can be more computationally intensive due to the exponential growth in joint action spaces but do not suffer from non-stationarity [5].

CTDE methods, such as MADDPG, Counterfactual Multi-Agent (COMA) policy gradient, central-V, Value Decomposition Networks (VDN), and QMIX, offer a compromise by leveraging full observability during training while allowing decentralized execution. This hybrid approach can be particularly beneficial in environments like satellite constellations, where decentralized decision-making is essential during operation but centralized strategies can optimize learning [44], [21], [20].

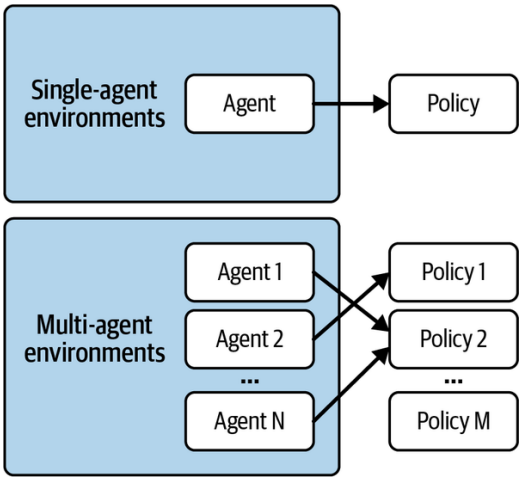


Figure 3.5 Mapping agents to policies in multi-agent reinforcement learning problems [35].

As illustrated in Figures 3.4 and 3.5, the multi-agent setup and different policy mappings in DRL allows for a diverse range of interactions among agents, including collaborative, competitive, and mixed strategies. This heterogeneity provides a more realistic approach to tackling complex problems like the FSS, where multiple satellites may need to coordinate or compete for limited resources.

4 Simulation and Training Framework

4.1 Background

In order to obtain a set of policies that can achieve a good performance when working with a group of federated satellites, giving the lack of real data, the development of a proper simulation framework is one of the main and most important tasks in this project.

An essential characteristic and requirement of this system is scalability, therefore, a modular simulator is the chosen approach. It needs at least three main components: the satellites, the federation simulator and the RL environment that serves as connection with the Python RL libraries and AI frameworks.

The steps to integrate all three components starts by including the participant satellites (both observers and targets) in the simulator and then wrap everything using the Gymnasium [45], [46] and Ray RLlib [36] APIs, which makes sure all the interfaces with RL libraries are correct. Figure 4.1 illustrates the main higher architecture of the framework. A sequence diagram of the whole framework can also be found in the Annex (Figure A.5).

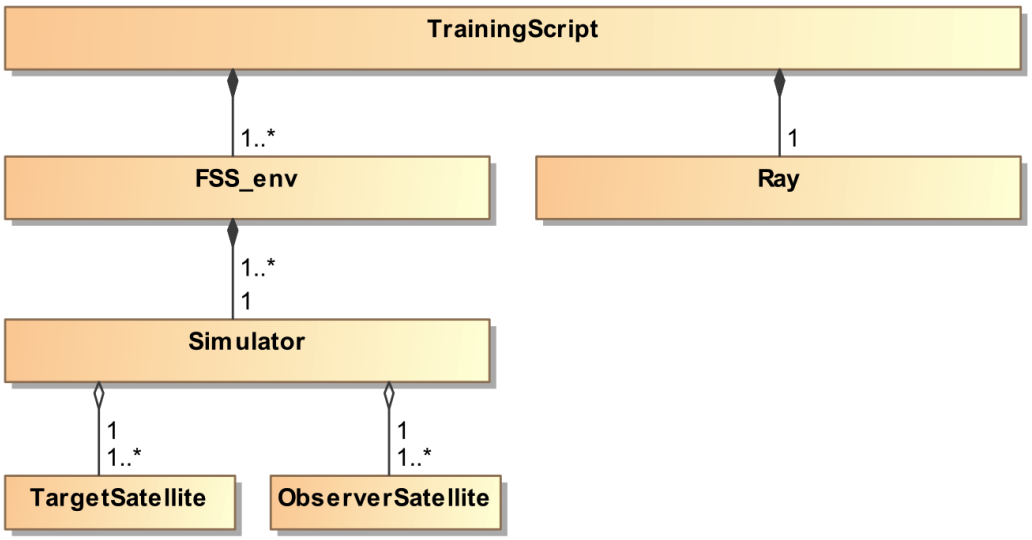


Figure 4.1 Training and simulation framework architecture.

The simulator and satellites parts used for the framework are developed based on the work by Messina [6]. While Messina's work is primarily focused on information propagation among different communication structures (centralized, decentralized, etc.), observation tasks and other features such as charging and partial observations and communications were also implemented for this project. This allows for further capabilities and tests, enhancing the federation simulation.

Regarding the RL libraries, the two main available options are SB3 [47] and Ray RLlib [36]. Although there is not much literature on the comparison between these two tools, a common consensus among users is that RLlib is the best for managing large-scale projects, while SB3 provides an easier and more user friendly interface for speed-up smaller ones. Therefore, RLlib was the chosen option, since it allows for distributed training and it has great MARL support. The access to Leibniz Supercomputing Centre (LRZ) AI Cluster provides

powerful machines equipped with state-of-the-art hardware like NVIDIA A100s (available hardware and more information can be seen in their documentation webpage [48]). In order to leverage these resources, RLlib, despite its steeper learning curve, has all the necessary capabilities to scale up the project and obtain a better overall performance. Table 4.1 displays the algorithm catalog available in Ray RLlib. Moreover PyTorch was the chosen option for the RL framework, since it works with all the algorithms we use, even with multi-GPU.

Algorithm	Frameworks	Discrete Actions	Continuous Actions	Multi-Agent	Model Support	Multi-GPU
APPO	tf + torch	**Yes** +parametric	**Yes**	**Yes**	+RNN, +LSTM auto-wrapping, +Attention, +autoreg	tf + torch
BC	tf + torch	**Yes** +parametric	**Yes**	**Yes**	+RNN	torch
CQL	tf + torch	No	**Yes**	No		tf + torch
DreamerV3	tf	**Yes**	**Yes**	No	+RNN (GRU-based by default)	tf
DQN, Rainbow	tf + torch	**Yes** +parametric	No	**Yes**		tf + torch
IMPALA	tf + torch	**Yes** +parametric	**Yes**	**Yes**	+RNN, +LSTM auto-wrapping, +Attention, +autoreg	tf + torch
MARWIL	tf + torch	**Yes** +parametric	**Yes**	**Yes**	+RNN	torch
PPO	tf + torch	**Yes** +parametric	**Yes**	**Yes**	+RNN, +LSTM auto-wrapping, +Attention, +autoreg	tf + torch
SAC	tf + torch	**Yes**	**Yes**	**Yes**		torch

Table 4.1 Overview of available algorithms in RLlib (v.2.10) [36]

4.2 Implementation Methodology

In the next sections, the main parts of the code are presented to provide an specific definition of Section 2. The implementation methodology outlines the steps and techniques used to integrate the various components of the FSS simulation framework. This framework is designed to support the development and evaluation of reinforcement learning algorithms in a simulated satellite constellation environment. By carefully defining the problem space and the interactions between different system components, we ensure that the simulation environment accurately reflects the complexities and challenges inherent in real-world satellite operations.

4.2.1 Decentralized Partially Observable Markov Decision Process Formulation

In our FSS simulation, we utilize the Decentralized Partially Observable Markov Decision Process (Dec-POMDP) framework. This framework is essential for modeling systems with multiple agents, where each agent has only partial information about the global state and must make decisions considering the potential actions of other agents. The inherent complexity of Dec-POMDPs arises from the need for coordination among agents and the computational challenges associated with high-dimensional state and action spaces.

A Dec-POMDP is formally defined by the tuple $(N, S, \{A_i\}, P, \{R_i\}, \gamma, \{O_i\}, Z)$, as outlined in Section 2, to which we include targets (T) for completeness. Below is a brief summary of each parameter, with detailed explanations provided in the following subsections. Notably, while the discount factor (γ), the transition probability matrix (P), and the observation probability matrix (Z) are key theoretical components, they are not explicitly defined in the simulation setup. Instead, the RL algorithms implicitly handle these elements during training. The discount factor is a predefined parameter that influences the weighting of future rewards in the cumulative reward calculation, encouraging long-term planning. The transition probabilities, although not directly modeled, are effectively learned by the agent through interactions with the environment, as the agent updates its policy based on observed state transitions and rewards.

- N represents the number of agents in the system, corresponding to the observer satellites in our FSS. For our use case, we used 20 agents.
- T denotes the number of target satellites present in the simulation. In our specific case, there are 20 targets.
- S is the set of possible states of the environment, including all relevant variables such as satellite positions, velocities, and communication statuses.

- $\{A_i\}$ denotes the set of actions available to each agent i. In our case there are $2 + T$ actions per agent, making a total of 22 possible actions for our use case. The first action accounts for stand-by, the second one tries to propagate the information throughout the network and the rest account for observing Target$[A_i - 2]$.
- P is the transition probability matrix, describing the probability of moving from one state to another given the current state and the actions of all agents. This encapsulates the physical dynamics of the satellites and their interactions.
- $\{R_i\}$ is the reward function for each agent i, providing feedback based on the actions taken and their outcomes. The reward functions are designed to align with mission objectives, giving positive rewards when successfully communicating with other observer or when obtaining a high-quality observation of a target.
- γ is the discount factor, reflecting the importance of future rewards compared to immediate ones. A higher value emphasizes long-term planning.
- $\{O_i\}$ represents the set of observations available to each agent i, which may include partial information about the state of other satellites and targets. A more detailed view on this component is given in Section 4.2.4.
- Z is the observation probability matrix, detailing the likelihood of receiving a particular observation given the state and actions. This accounts for the partial and uncertain nature of information in the system.

Again, the primary objective in a Dec-POMDP framework is to find a set of policies $\{\pi_i\}$ for the agents that maximize the expected cumulative reward shown in Equation 3.1

In the context of satellite operations, each satellite (agent) operates with partial and dynamic information about the environment and other satellites. For instance, an agent may not always know the exact position or status of another satellite due to limited communication windows or sensor constraints. This scenario fits well within the Dec-POMDP framework, as it inherently handles the challenges of partial observability and the need for decentralized decision-making.

This Dec-POMDP formulation allows us to model the complexities of coordinating multiple satellites, each making decisions based on limited information, to achieve shared objectives. This framework is crucial for developing and testing advanced reinforcement learning algorithms tailored for multi-agent systems in dynamic and uncertain environments.

4.2.2 Satellites

This part is fundamental for accurately simulating the physical and functional attributes of satellites, encompassing two primary types: observers, responsible for gathering data and observations, and targets, which serve as objects of observation.

While the satellite's capabilites and main characteristics were reviewed in Section 2.4, in this part a more architectural and code-oriented view is described.

The core structure of the code includes the `Satellite`, `ObserverSatellite`, and `TargetSatellite` classes. The `Satellite` class serves as the base, providing essential attributes and methods shared across all satellites, such as those for orbital mechanics and power systems explained in Section 2.4. The `ObserverSatellite` class builds upon this foundation by incorporating specific functionalities required for observation and communication missions, including calculating pointing accuracy and managing observational data received from others.

The `ObserverSatellite` and `TargetSatellite` classes inherit all the properties of the base `Satellite` class, as depicted in Figure 4.2. The `ObserverSatellite` class, in particular, extends these basic functionalities to include advanced methods for conducting observation missions, processing data, and coordinating with other satellites.

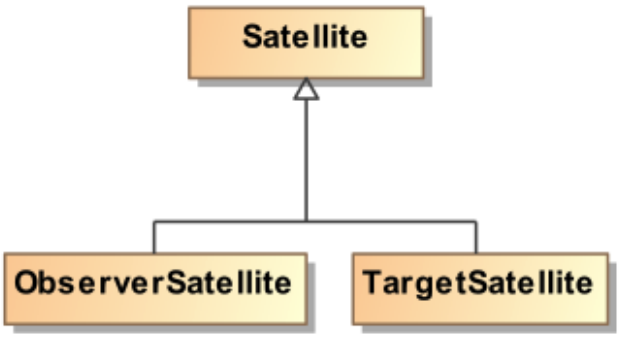


Figure 4.2 Instantiation of satellite class.

Key components of the `Satellite` class are based on Messina's work [6] and include:

- **Propagate Orbit**: simulates the satellite's orbit using Keplerian elements and updates the satellite's position and velocity.
- **Propagate Attitude**: updates the satellite's orientation over time, using angular velocity and quaternion representation.
- **Distance Calculation**: computes the distance between satellites, crucial for assessing communication and observational capabilities.
- **Pointing Direction Calculation**: determines the satellite's pointing direction, essential for observation missions.
- **Battery Management**: manages the satellite's battery charge, accounting for sunlight exposure and solar panel efficiency.
- **Sunlight Exposure Calculation**: calculates the amount of sunlight received by the satellite's solar panels, considering eclipse conditions and orientation.
- **Communication and Data Handling**: oversees data storage and communication capabilities, assessing the satellite's ability to communicate based on its state and distance to other satellites.

The complete attributes and methods of the `Satellite` class can be found in Figure A.1 in the Annex A section. Figure A.2 of the Annex details the complete attributes and methods of the `ObserverSatellite` class, highlighting its enhanced capabilities for observational tasks.

Key components of the `ObserverSatellite` class include the following:

- **Observation Techniques**: methods for evaluating pointing accuracy, observing targets, and updating data and contact matrices, which are critical for the coordination and data exchange among satellites.
- **Observation Management**: functions for assessing the status and quality of observations, tracking observation counts, and managing observation-related data.
- **Communication Management**: methods to determine communication capabilities with other satellites, based on the type of coordination network (centralized, decentralized, etc.).
- **Data Processing**: manages the processing state and availability of new data for communication with other satellites.
- **Energy and Storage Management**: detailed tracking of energy consumption rates for various activities (standby, communication, observation) and management of data storage capacities.

Observation, communication, data processing and energy and storage management capabilities are based on Messina's work [6].

This class structure supports the simulation's objective of accurately modeling the behaviors and interactions within a satellite constellation, thereby facilitating the development and refinement of advanced reinforcement learning algorithms in a controlled and realistic environment.

4.2.3 Simulator

The simulator file is the core component that models the dynamics and interactions within the satellite federation. It encompasses the mechanisms for information propagation and observation tasks, ensuring realistic and computationally efficient processing of agent actions. While detailed orbital mechanics are managed within the satellite classes, the simulator focuses on the interactions and coordination among the satellites.

The simulator framework is built around the `Simulator` class, with specific subclasses as displayed in Figure 4.3 to handle different communication structures within the federated satellite system. These subclasses define the rules for how satellites communicate and coordinate, reflecting different real-world operational scenarios.

The simulator inherits and extends functionalities originally developed by Messina [6], integrating additional capabilities such as battery management, object detection, and observation coordination.

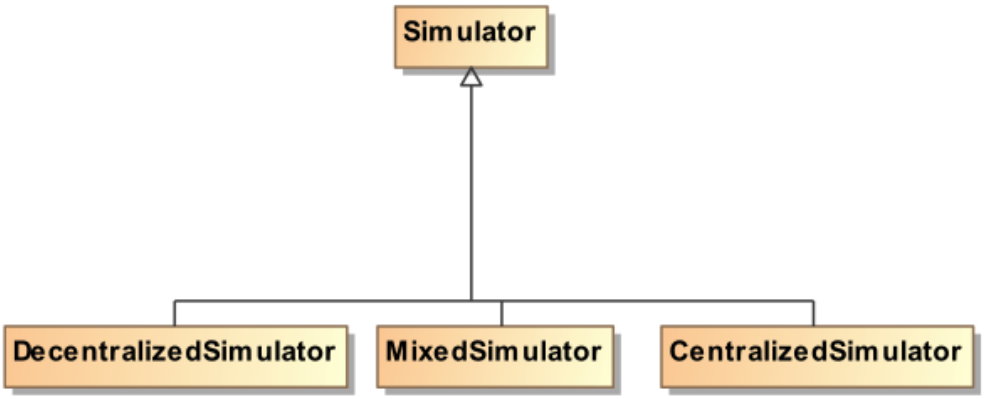


Figure 4.3 Instantiation of simulator class.

Key components of the `Simulator` class include:

- **Time Step Management**: manages the simulation time steps, ensuring consistent progression and synchronization across all simulation elements.
- **Action Processing**: interprets and processes actions taken by the satellites, updating the internal state of the simulation accordingly. This component is crucial for evaluating the outcomes of different strategies implemented by the RL agents.
- **Reward Calculation**: computes a reward signal to guide RL training, based on the outcomes of the agents' actions. This mechanism helps the RL model learn optimal behaviors over time.
- **Termination Condition**: defines conditions under which the simulation episode ends, such as reaching a maximum number of time steps or achieving specific mission objectives.
- **Communication and Data Exchange**: manages the data communication between observer satellites, including updating adjacency and contact matrices. This feature is particularly important for simulating networked operations and decentralized decision-making processes.
- **Energy and Storage Management**: tracks energy consumption and storage utilization, critical for maintaining operational realism, especially when considering the limited resources available on satellite platforms.

The comprehensive attributes and methods of the `Simulator` class are illustrated in Figure A.3 in the Annex. This structure supports a modular approach, allowing for easy adaptation and integration with various RL libraries and APIs.

4.2.4 Federated Satellite System Environment

The environment file (`FSS_env`) establishes a comprehensive environment for simulating, training, and deploying reinforcement learning algorithms within the Federated Satellite System (FSS). This environment is built upon Gymnasium's Multi-Agent environment framework [46], providing a standard interface for interacting with RL algorithms. Gymnasium, originally developed by OpenAI and now maintained by the Farama Foundation, offers a robust platform for developing and comparing RL algorithms through a standardized API and a suite of environments [45].

The `FSS_env` class inherits from Ray RLlib's `MultiAgentEnv` class, setting up a simulation with multiple satellite agents, targets and a simulator. This environment enables seamless interaction with algorithms by standardizing inputs and outputs, making it compatible with various RL frameworks. The key components that can be extracted from the class are:

- **Action Space**: the action space is defined as a discrete set of actions, encompassing the primary operations a satellite can perform. This includes stand-by, communicating with other satellites and observing targets. The exact number of actions is determined by the number of targets like defined in the Dec-POMDP, forming a space of $2 + T$ possible actions.
- **Observation Space**: the observation space is a complex, continuous space represented by a matrix that encapsulates the state of each satellite. Under includes details such as orbital parameters, battery levels, storage capacity, communication status, and observation status of targets.
- **Reset Function**: this function initializes the simulator and satellites, setting up the initial conditions for the simulation. It returns the initial state of the environment, preparing it for the RL agent's interaction.
- **Step Function**: the step function updates the state of the simulator based on the actions taken by the agents. It calls the `Simulator` class step function and runs it until a special event is detected.
- **Special Event Detection**: although not explicitly included in the class methods, this feature identifies critical events that might significantly alter the state of the environment, such as events in which two satellites become close enough to communicate or observe. These events trigger agent responses in the simulation, making it more efficient to train by only asking the agents to compute actions whenever they can obtain a positive rewards and avoiding "empty" steps in which the best possible action is to stand-by. Nevertheless, most agents in the special event steps also have to learn to stand-by and not waste resources if they are not the ones involved in the event, making sure the policy is trained in all kind of situations.

The complete list of methods and attributes of the environment class can be seen in Figure A.4 in the Annex. However, more detailed description of the spaces and reward function is given here.

State Space

The state space in the `FSS_env` class encompasses all possible states that the system can occupy. This comprehensive space includes both observable and underlying unobservable states, capturing the full complexity of the environment.

In detail, the state space consists of:

- **State Variables**: These include complete orbital parameters (such as semimajor axis, inclination, eccentricity, etc.) for each satellite, battery levels, data storage capacities, communication statuses, and observation statuses for each satellite.

- **Dimensionality**: Given the number of satellites and the complexity of the data (positions, velocities, system statuses), the state space is high-dimensional. For example, the state of each satellite may be represented by a feature vector, which includes various aspects like position coordinates, velocity components, energy levels, and data storage statuses.

Observation Space

The observation space of the DRL problem is dynamically structured to capture the state of each satellite and the system as a whole. This includes the following parameters:

Parameter	Description	Data Type
`observer_satellites`	Orbital parameters of observer satellites	Continuous
`band`	Own communication band identifier (ranging from 1 to 5)	Discrete
`availability`	Own availability status	Binary
`target_satellites`	Target satellites' position and velocity	Continuous
`battery`	Battery level of observers	Continuous
`storage`	Storage level of observers	Continuous
`observation_status`	Observation status of target satellites	Discrete
`pointing_accuracy`	Pointing accuracy of observer satellites towards target satellites	Continuous
`communication_status`	Communication status between observers	Binary
`communication_ability`	Communication ability with other observer satellites	Binary

Table 4.2 Information shared or/and available for each Observer Satellite

Bear in mind that these attributes are dynamic and partial, depending on each observer communications and observations as has been established in the Dec-POMDP formulation.

Action Space

The action space in the `FSS_env` environment defines the set of all possible actions that each agent (satellite) can take. This space is critical for the RL agents as it dictates their potential decisions at any given time.

Key features of the action space include:

- **Discrete Actions**: each agent can select from a discrete set of actions, which include operations like observing a target, communicating with other satellites, or engaging in standby modes. The number of actions is determined by the number of targets, resulting in an action space of size $2 + T$.
- **Action Mapping**: actions are encoded as integers, with each integer corresponding to a specific action such as selecting a target to observe or initiating a communication sequence. This encoding simplifies the integration with RL algorithms that require discrete action spaces.

Rewards

The reward is a critical component of the `FSS_env` environment, guiding the RL agents towards achieving the desired objectives. It defines the rewards or penalties that agents receive based on their actions and the resulting state transitions.

The rewards are designed to:

- **Encourage Effective Observations**: agents receive positive rewards for successfully observing targets, particularly when such observations improve the overall knowledge state of the system (to avoid repetitions) or are taken with a high pointing accuracy.

- **Promote Efficient Communication**: rewards are also granted for effective communication between satellites.
- **Penalize Inefficiency**: negative rewards or penalties are imposed for actions that waste resources or deplete them, such as not completing a data transmission, running out of battery or filling up the storage.
- **Incorporate Long-Term Goals**: the reward incorporates factors that promote long-term collaborative strategic goals, giving points when the main mission (all the participants have observation data on all the targets) has been accomplished, and ensuring continuous system-wide situational awareness.

4.3 Validation of Simulation Framework

To ensure the accuracy and reliability of the developed FSS simulation framework, a comprehensive validation process was conducted. This process involved several tests designed to verify key functionalities, including orbit propagation, attitude propagation, battery management, communication effectiveness, and observation accuracy. These tests are crucial for confirming that the simulation behaves as expected under various conditions and scenarios.

The validation tests were implemented using the `unittest` framework in Python, with a consistent set of initial conditions applied across all tests to maintain reproducibility and control.

In the orbit propagation test, the custom Keplerian orbit propagation results were compared against the outputs from the `poliastro` library [49], which serves as a reference standard. The comparison involved calculating position and velocity errors between the custom propagation and `poliastro`'s results, ensuring that the error margins for position and velocity remained within acceptable limits—specifically less than 1000 meters (approximately 0.01% of the total orbit magnitude) for position and 100 m/s (roughly 1% of the magnitude) for velocity.

The attitude propagation test validated the quaternion-based attitude propagation mechanism by comparing it against the results obtained using `scipy`'s `solve_ivp` function. The accuracy of quaternion propagation was assessed by measuring the maximum quaternion error, which was maintained below a threshold of 0.1.

For the battery test, energy usage over time was simulated, accounting for various operational modes such as standby, communication, and observation. The test evaluated the trajectory of energy consumption and availability over a simulated period, ensuring that energy calculations were accurate and that energy levels did not exceed storage capacity or fall below zero.

The distance and effective data rate test verified the calculation of distances between satellites and the corresponding effective data rates. It checked the correct calculation of distances based on satellite positions and ensured that data rate determinations were accurate, producing non-negative values aligned with the expected communication capabilities.

The observation accuracy test assessed the ability of satellites to correctly identify and observe targets within their field of view and maximum distance. Various scenarios with different distances and attitudes were tested, ensuring that observations only occurred when the target was within the field of view and the maximum detection distance. Additionally, the test verified that the observation status matrix, cumulative pointing accuracy, and observation counts were accurately updated based on the satellite's observational capabilities and configurations.

The validation results indicated that all tests were executed successfully, confirming the correctness and robustness of the simulation framework. Detailed test results and error margins are documented in the Annex A.3 for reference. The validation process provided confidence in the simulation's ability to accurately model satellite behaviors and interactions, laying a solid foundation for subsequent research and development efforts utilizing this framework.

5 Comparative Analysis of Training Process of Reinforcement Learning Algorithms

The training and evaluation of reinforcement learning (RL) algorithms were conducted using an HPC cluster provided by the Leibniz Supercomputing Center (LRZ) [48], specifically utilizing the HGX A100 Architecture. Due to the intensive computational requirements and the extensive data necessary for training, this setup was chosen after initial failed attempts to train with a MacBook Pro and a Jetson AGX Orin.

For reference, each training iteration in the MacBook and Jetson takes approximately between 90 and 120 minutes, while in the HGX one of this iterations is computed in not more than 10 or 12 minutes.

In addition, Ray RLlib was selected for its scalability and support for distributed training, allowing for efficient management of complex, large-scale simulations. The choice of hardware significantly impacted the training efficiency due to the enhanced computation speed provided by GPUs, which is vital for the matrix multiplications in DRL. The training utilized the HGX A100 Architecture, part of the LRZ cluster:

Specification	**Details**
Slurm Partition	`lrz-hgx-a100-80x4`
Number of nodes	5
CPU cores per node	96
Memory per node	1 TB
GPUs per node	4 NVIDIA A100
Memory per GPU	80 GB

Table 5.1 HGX A100 Architecture specifications used in the training [48].

The environment was configured with the parameters `num_targets`, `num_observers`, `communication_type`, `time_step`, and `duration`. The resource allocation was handled through settings like `num_rollout_workers` (parallel actors for simulating environment interactions), `num_envs_per_worker`, `num_cpus_per_worker`, and `num_gpus_per_worker`.

5.1 Hyperparameter Search

For hyperparameter optimization, a grid search with the advanced ASHA Scheduler and Optimizer [50] was employed. A major assumption of SHA and ASHA is that if a trial performs well over an initial short time interval it will perform well at longer time intervals, combining random search with principled early stopping in an asynchronous way. This optimization included 20 samples per algorithm, with the possibility of early stopping if results were significantly below expectations. The analysis focused on three RL algorithms: Deep Q-Network (DQN), Soft Actor-Critic (SAC) and Proximal Policy Optimization (PPO).

The hyperparameter search and training process were implemented using Python with Ray and Torch libraries. Key configurations included environment settings, resource settings, and search space specifications.

5.1.1 Deep Q-Network

For DQN, the hyperparameter search space included:

- Target network update frequency (`target_network_update_freq`): choice between 500, 1000, 2000 or 3600.
- Learning rate schedule (`lr_schedule`): a selection from [[0, 1e-4], [1000000, 1e-5]] or [[0, 1e-3], [1000000, 1e-4]].

Target Network Update Frequency	Learning Rate Schedule	Reward
2000	[[0, 1e-4], [1000000, 1e-5]]	1344.07
1000	[[0, 1e-3], [1000000, 1e-4]]	-626.465
2000	[[0, 1e-3], [1000000, 1e-4]]	-1000.35
500	[[0, 1e-4], [1000000, 1e-5]]	-972.159
2000	[[0, 1e-3], [1000000, 1e-4]]	-975.798
2000	[[0, 1e-3], [1000000, 1e-4]]	-942.652
1000	[[0, 1e-4], [1000000, 1e-5]]	1068.09
2000	[[0, 1e-3], [1000000, 1e-4]]	-1024.6
500	[[0, 1e-4], [1000000, 1e-5]]	-743.036
3600	[[0, 1e-3], [1000000, 1e-4]]	-1032.45
1000	[[0, 1e-3], [1000000, 1e-4]]	-1115.76
1000	[[0, 1e-3], [1000000, 1e-4]]	-1106.2
1000	[[0, 1e-4], [1000000, 1e-5]]	-980.545
3600	[[0, 1e-3], [1000000, 1e-4]]	-2348.8
3600	[[0, 1e-3], [1000000, 1e-4]]	-1023.31
500	[[0, 1e-4], [1000000, 1e-5]]	277.001
500	[[0, 1e-4], [1000000, 1e-5]]	-1033.24
3600	[[0, 1e-4], [1000000, 1e-5]]	-987.896
3600	[[0, 1e-4], [1000000, 1e-5]]	-996.772
3600	[[0, 1e-4], [1000000, 1e-5]]	-1025.75

Table 5.2 DQN Hyperparameter Search Results

The ASHA Scheduler managed the trials based on the episode reward mean, optimizing the parameters to enhance training efficiency. Results of the search are displayed in Table 5.2, with the highlighted value meaning the chosen configuration.

5.1.2 Soft Actor-Critic

For SAC, the hyperparameter search space included:

- Learning rate (`lr`): chosen from a uniform distribution ranging from 1e-7 up to 1e-4.
- Discount factor (`gamma`): random sample from a random distribution between 0.9 and 0.99.
- GAE lambda (`lambda`): a random selection from 0.9 to 1.0.

The ASHA Scheduler managed the trials just as in the previous case, with the results shown in Table 5.3, highlighting the chosen configuration.

5.1.3 Proximal Policy Optimization

For PPO, the hyperparameter search space included:

- Learning rate (`lr`): ranging from 1e-7 to 1e-4.
- Discount factor (`gamma`): from 0.9 up to 0.99.
- GAE lambda (`lambda`): uniformly distributed choice from 0.9 to 1.0.

Gamma	Learning Rate	Reward
0.979053	1.22718e-06	-1746.93
0.942453	8.10283e-05	-2713.53
0.930610	2.78385e-05	-1733.62
0.958065	1.31135e-05	-1681.76
0.910371	1.07042e-07	-1699.19
0.984420	1.49338e-06	-29537.8
0.964084	1.88099e-05	-28790.8
0.970649	4.00097e-05	-1785.26
0.963136	6.31194e-05	-1773.75
0.946066	5.52082e-07	-1721.47
0.978031	3.08891e-06	-413.898
0.967298	3.38082e-05	-1750.58
0.927746	7.10176e-07	-1722.01
0.950303	8.55035e-05	-2104.61
0.948195	2.11867e-07	-1719.45
0.925953	6.96477e-06	-1850.80
0.917211	1.61142e-07	-1712.76
0.930684	3.62808e-05	-1730.05
0.939489	3.59890e-07	-1741.40
0.905464	8.59721e-05	-1750.84

Table 5.3 SAC Hyperparameter Search Results

Gamma	Learning Rate	Lambda	Reward
0.923488	1.69668e-05	0.992815	-1160.51
0.92444	9.89899e-05	0.929464	-1280
0.977139	8.44094e-05	0.904515	-1122.54
0.954782	4.15919e-05	0.924528	-1131.59
0.907877	6.92507e-05	0.938835	-1273.36
0.984239	1.99378e-05	0.952727	-1281.85
0.931683	6.92763e-05	0.929491	-1113.12
0.932597	1.66651e-05	0.9202	-1080.1
0.923478	5.93653e-05	0.933646	-1130.07
0.970321	9.66406e-05	0.97461	-1287.03
0.943435	1.85936e-05	0.908595	-1272.82
0.914669	7.80153e-05	0.946209	-1110.51
0.927431	6.12855e-05	0.998174	-1264.96
0.978612	9.82495e-05	0.939177	-1276.17
0.919107	4.21289e-05	0.919598	-1109.66
0.932353	2.98588e-05	0.971295	-1117.3
0.930893	1.65387e-05	0.941362	-1303.55
0.928018	9.16354e-05	0.981015	-1283.2
0.957544	9.57006e-05	0.996926	-1165.02
0.946976	1.59942e-05	0.901437	-1103.02

Table 5.4 PPO Hyperparameter Search Results

Results of the tuning are summarized in Table 5.4, again having the best results highlighted in cyan.

5.2 Training

The training framework and infrastructure played a vital role in achieving efficient and scalable training for the RL algorithms. It must be stated that all trainings were carried out for nearly 3 days using the unconstrained decentralized coordination configuration. Nevertheless all three algorithms are tested in centralized and decentralized network configurations in the results part (Section 6).

Several frameworks facilitate the training of DRL agents by providing tools for managing the complexities of data collection, parallelization, and model updating. While frameworks like Ray RLlib offer robust support for parallelized training and multi-agent setups, others like SB3 provide reliable and fast implementations of common RL algorithms. For this project, Ray RLlib has been chosen due to its scalability and support for distributed training as was mentioned in previous sections. This Ray framework allows for the creation of multiple "workers," which can either be local (within the same process) or remote (distributed across multiple nodes), making this setup (see Figure 5.1) particularly advantageous for large-scale simulations and complex environments, such as those encountered in satellite operations.

Effective resource allocation is key to optimizing the training process. Parameters like `num_workers`, `num_gpus`, and `num_cpus_per_worker` allow fine-tuning of how resources are utilized. This ensures that both the policy update and experience collection phases are performed efficiently, making full use of available computational resources.

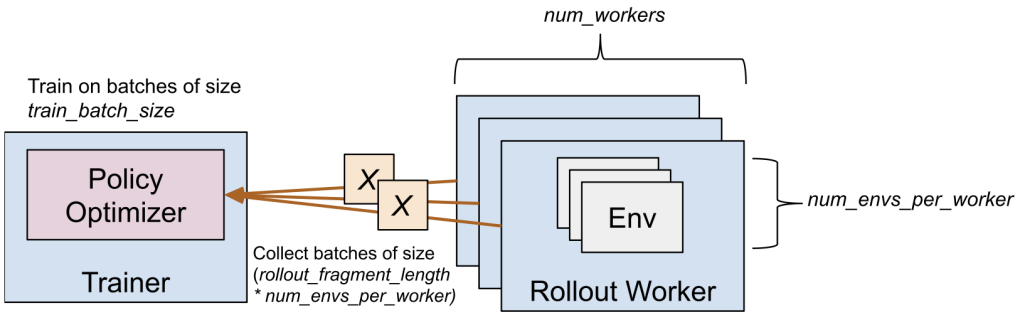


Figure 5.1 Training parallelization with Ray Rollout Workers.

The choice of hardware and infrastructure significantly impacts the efficiency and speed of the training process. The use of GPUs accelerates the computation of neural network operations, which is critical in DRL due to the large number of matrix multiplications involved. In distributed training setups, multiple GPUs or clusters can be utilized to further enhance training speed and handle larger models.

5.2.1 Deep Q-Network

The DQN algorithm's performance, as shown in Figure 5.2, reflects a gradual improvement over the course of the training iterations. Initially, there is a significant dip in performance, which could be indicative of the agent's exploration phase where it is actively trying to discover better strategies. This behavior is common in DRL as the DQN algorithm, through the process of exploration-exploitation, sometimes opts for suboptimal actions in the short term to gather more diverse experiences.

After this initial drop, the algorithm's performance begins to stabilize and improve steadily, indicating that the DQN agent is effectively learning from its environment and refining its policy. The relatively smooth upward trend post-exploration suggests that the DQN model is successfully leveraging its experience replay and the learned Q-values are converging towards an optimal policy. However, the occasional minor fluctuations hint at

the ongoing balance the agent is maintaining between exploration and exploitation, and the potential sensitivity of DQN to certain hyperparameters such as the variable learning rate. Despite these fluctuations, the final stable behaviour indicates a strong potential of the DQN model in this environment when appropriately tuned.

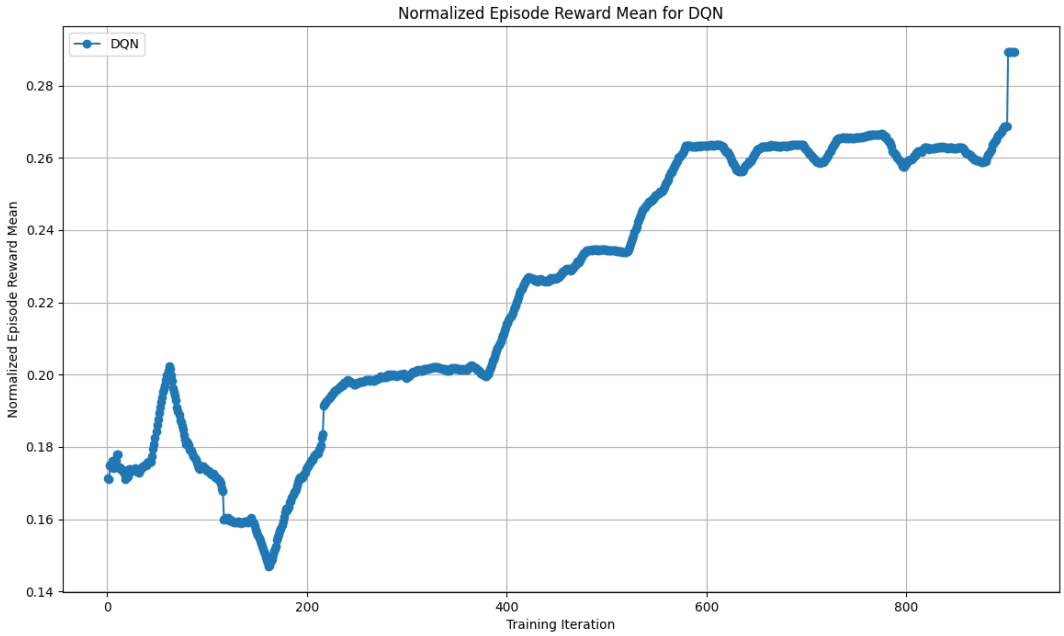


Figure 5.2 Normalized Episode Reward Mean against Training Iterations for DQN.

5.2.2 Soft Actor-Critic

The performance of the SAC algorithm, as seen in Figure 5.3, indicates a relatively flat progression in normalized episode reward mean across the training iterations. This flat curve suggests that SAC did not experience significant learning improvements during training, remaining mostly stagnant after an initial increase.

SAC is generally well-suited for continuous action spaces, where it excels by maintaining a balance between exploration and exploitation through its entropy regularization. However, in this discrete action space scenario, SAC may be struggling to apply its continuous action optimization principles effectively, leading to suboptimal policy learning. The algorithm's inability to improve further could also be due to suboptimal tuning of the entropy coefficient, which plays a crucial role in guiding the exploration process. If the coefficient is too high, the agent might focus too much on exploring, hindering its ability to exploit known good strategies; if too low, the exploration might not be sufficient, leading to premature convergence to suboptimal policies, which seems to be the case.

The relatively stable but low reward curve could also point to a mismatch between the algorithm's assumptions and the actual environment's dynamics, suggesting that alternative algorithms or modifications to the SAC approach might be necessary to achieve better performance in this specific task setup.

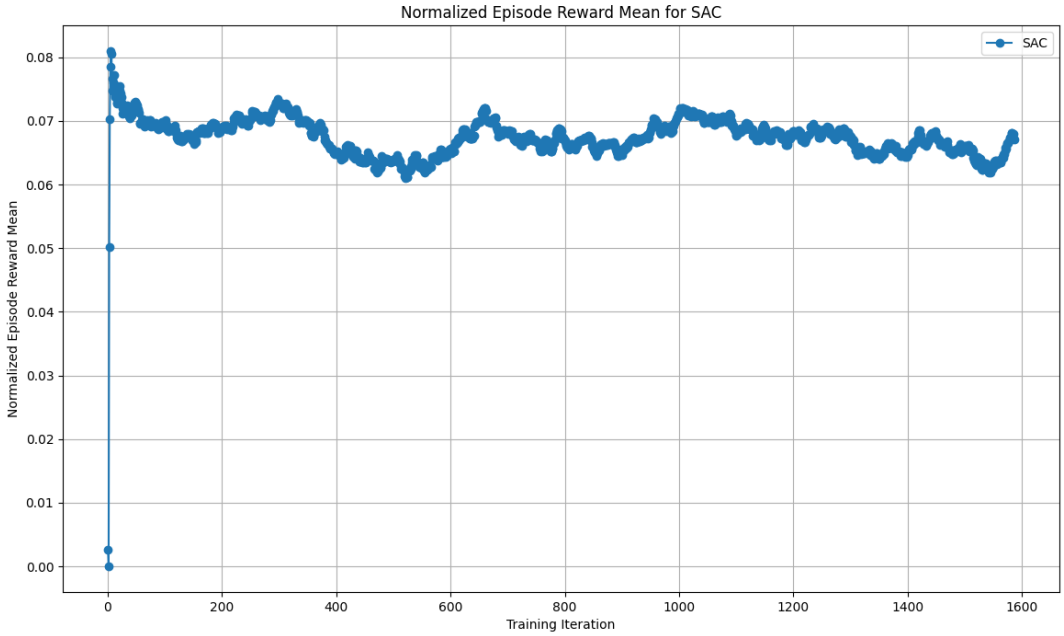


Figure 5.3 Normalized Episode Reward Mean against Training Iterations for SAC.

5.2.3 Proximal Policy Optimization

The PPO algorithm's performance, depicted in Figure 5.4, reveals a pattern of initial strong improvement followed by high instability. In the early stages of training, PPO demonstrates a rapid increase in normalized episode reward mean, which is characteristic of PPO's ability to quickly capitalize on advantageous policies due to its clipped objective function that stabilizes policy updates.

However, after reaching a peak, the performance starts to decline and fluctuates significantly. This behavior suggests that while PPO initially found a promising policy, it struggled to maintain and refine it over time. The observed instability could be attributed to the aggressive updates made by PPO, especially if the clipping parameter or other hyperparameters such as learning rate were not finely tuned. Additionally, the volatility in the reward curve might indicate that the algorithm is experiencing difficulties in dealing with the non-stationary nature of the environment, which can occur in dynamic and complex tasks like those modeled in this simulation.

The subsequent decline in performance after the peak could also reflect an overfitting issue, where the agent's policy becomes overly specialized to particular scenarios encountered earlier in training, resulting in less generalization to new situations. This highlights a potential need for further exploration of the hyperparameter space or alternative strategies such as using a more conservative clipping threshold to promote a more balanced exploration-exploitation trade-off.

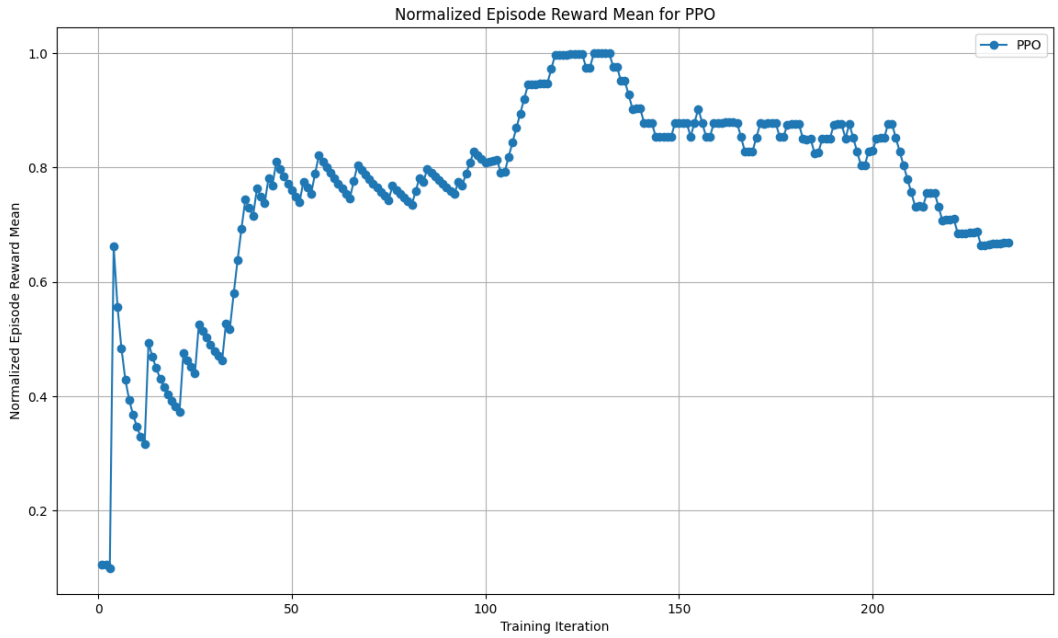


Figure 5.4 Normalized Episode Reward Mean against Training Iterations for PPO.

5.2.4 Comparative Analysis

The comparative analysis of the three algorithms' performance is depicted in Figures 5.5 and 5.6. These plots illustrate the normalized episode reward mean over training iterations and training time for all algorithms.

PPO initially outperformed the other algorithms by quickly achieving high rewards. However, its performance exhibited significant instability as training continued, leading to fluctuations and a decline in reward mean. This suggests that while PPO can rapidly identify strong policies, it struggles to maintain and improve them over time, possibly due to the algorithm's sensitivity to hyperparameter settings or the non-stationary nature of the task environment.

DQN, on the other hand, demonstrated a more gradual and consistent improvement, with fewer fluctuations after the initial exploration phase. This consistency indicates that DQN was able to steadily learn and refine its policies, ultimately leading to robust performance. The final surge in DQN performance further underscores its potential effectiveness in environments where stability and long-term policy optimization are critical.

SAC performed the weakest among the three algorithms, showing minimal improvement throughout the training period. This poor performance is likely due to SAC's inherent design, which is better suited for continuous action spaces rather than the discrete action space used in this scenario. The lack of significant reward increases suggests that SAC struggled to adapt to the specific challenges of this task, and may require further tuning or different configurations to be effective in similar environments.

Overall, DQN and PPO show potential for effective application in satellite constellation management tasks, with DQN being the most stable and reliable in this context. Future research might explore hybrid approaches that combine the rapid learning of PPO with the stability of DQN, or investigate modifications to SAC that better align with the discrete nature of the task.

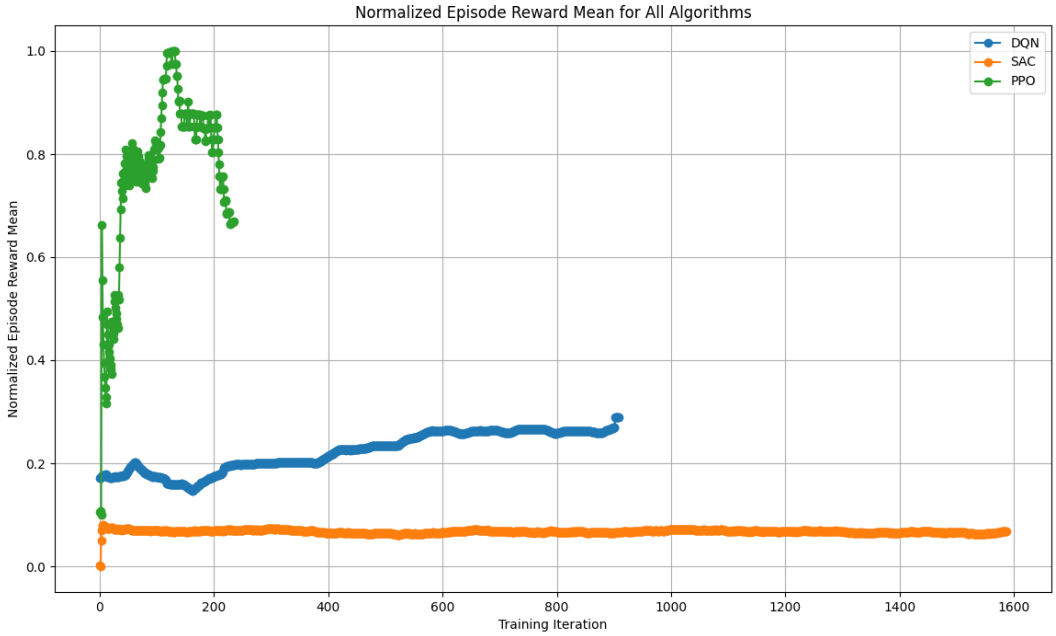


Figure 5.5 Normalized Episode Reward Mean over Training Iterations.

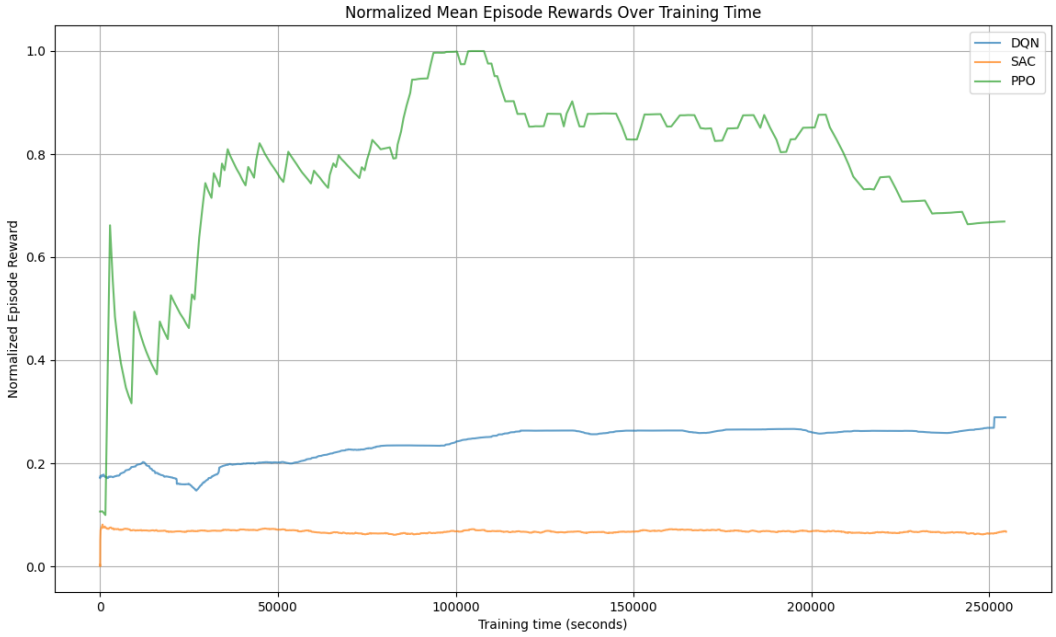


Figure 5.6 Normalized Mean Episode Rewards over Training Time.

6 Results

This section presents the results of the simulation experiments, where different coordination configurations were tested using the reinforcement learning algorithms presented. The focus is on three distinct coordination types: unconstrained (fully) decentralized coordination, centralized coordination with a single central node, and constrained (partially) decentralized coordination without central nodes.

In the fully decentralized coordination, all participants can talk to everyone, while in the constrained partially decentralized approach only the observers sharing the same communication band are allowed to exchange information between them.

Each scenario was simulated 100 times for each algorithm, and the results are discussed in this section. A random Monte Carlo experiment was also conducted to provide a baseline comparison, highlighting the performance differences between random actions and the trained policies. Moreover all data displayed in this chapter can be found in the form of a table in the Annex A.4.

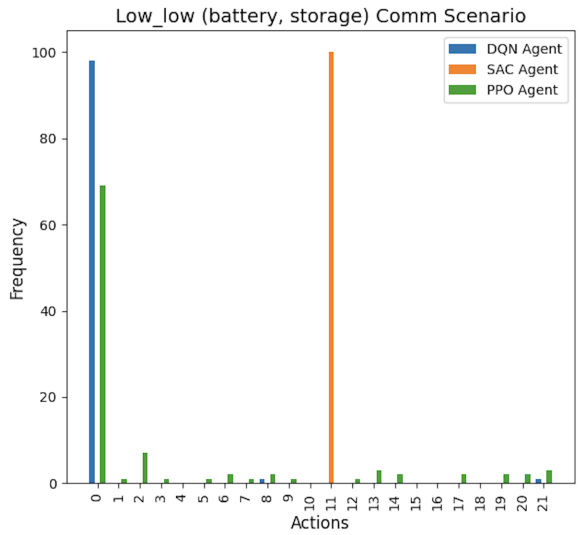
6.1 Validation of Policies

To further analyze the decision-making process of each agent, a series of 12 test scenarios was designed. These scenarios were structured around three specific test cases, each with a predefined optimal action that the agents were expected to select. The scenarios were tested under four different resource conditions, created by combining low and high battery and storage levels. The detailed logic and outcomes of the tests are summarized and can also be visualized in the Annex A.5.

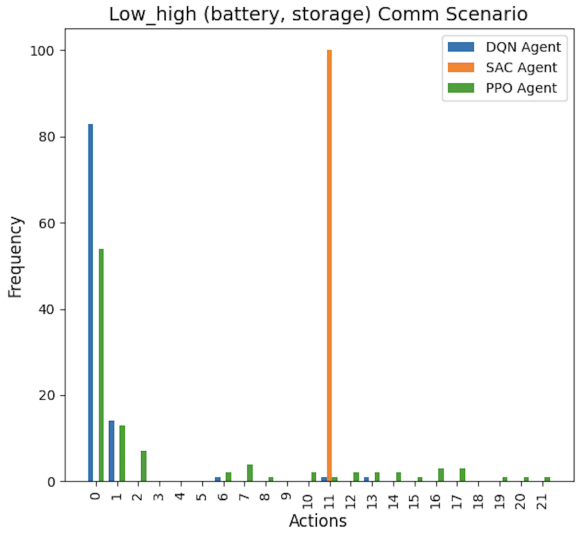
Each test scenario was carefully crafted to isolate particular aspects of the agents' decision-making processes. For example, in the communication test scenario, Observers 1 and 2 were placed within communication range of each other—specifically within 2325 km—to evaluate whether they could successfully identify and execute the correct communication action. These observers were intentionally positioned to be isolated from the observation test to prevent any overlap in test conditions.

For the observation test scenario, Observer 3 was positioned near Target 1, specifically within the observation range of 263 km. This scenario was designed to test the agents' ability to select the correct observation action. Similar to the communication test, the observation test was isolated from other scenarios to ensure that the agents' decision-making could be evaluated without interference from other actions.

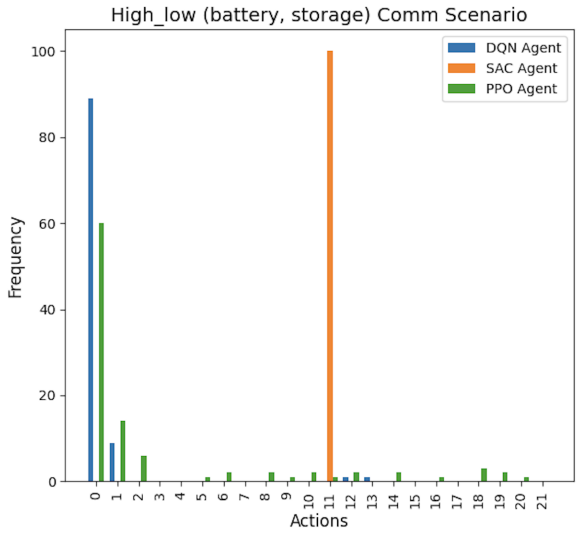
Additionally, an extra observer was included to test the stand-by scenario. This observer was positioned with a 90º offset in true anomaly relative to the other observers, ensuring that it was isolated from both the communication and observation tests. This configuration allowed the agent to evaluate the situation and correctly decide to take no action (i.e., stand-by) when no other meaningful action was appropriate.



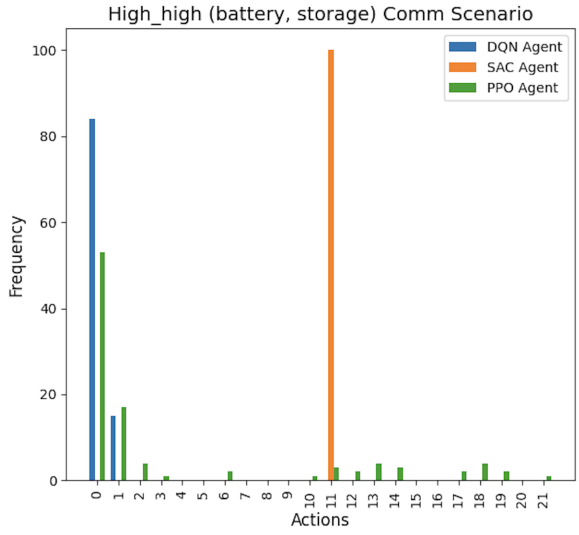
(a) Low-Low (Battery, Storage)



(b) Low-High (Battery, Storage)



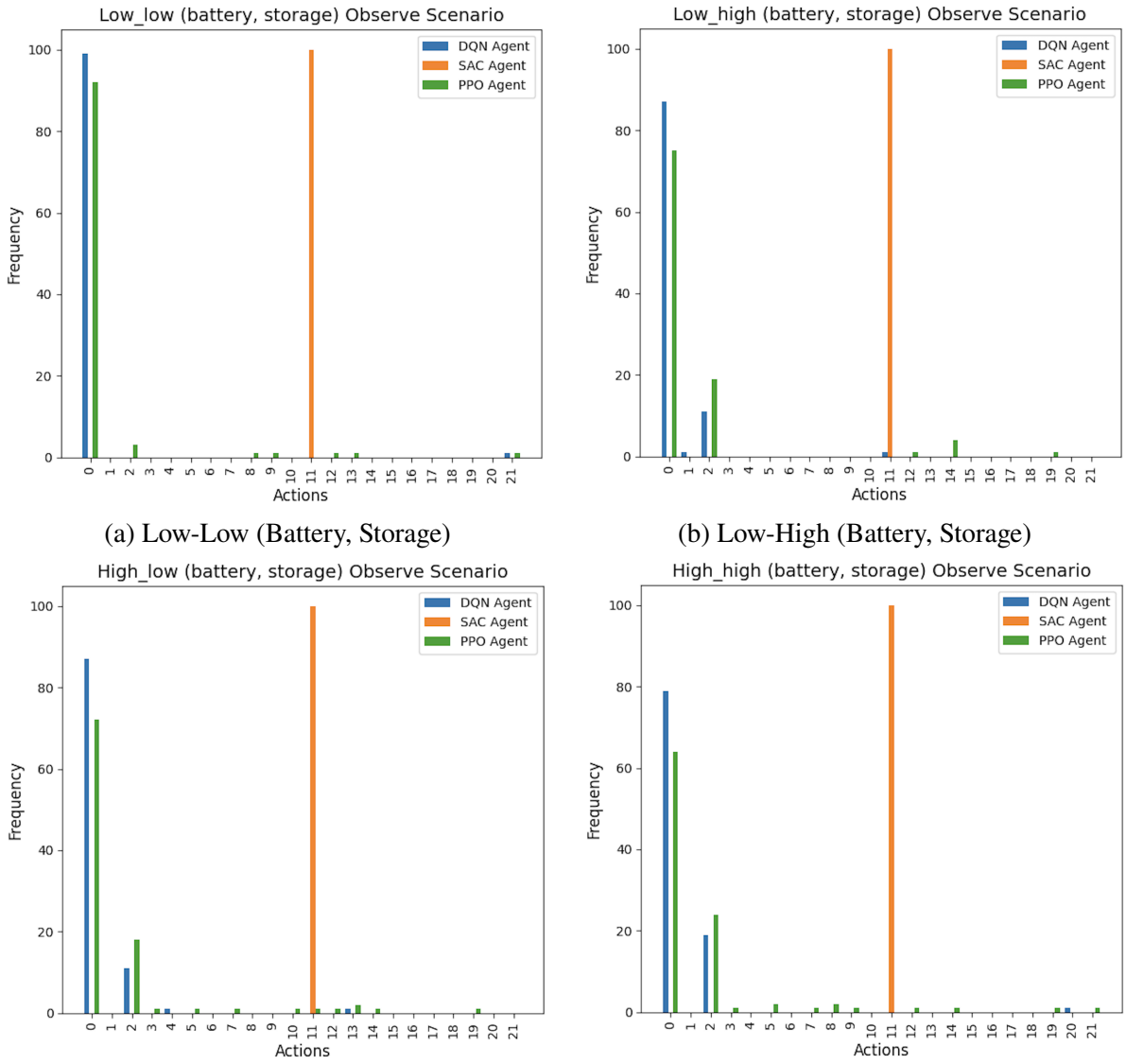
(c) High-Low (Battery, Storage)



(d) High-High (Battery, Storage)

Figure 6.1 Communication validation scenarios under different resource conditions. The correct action in these scenarios is Action 1.

Figure 6.1 illustrates the communication scenario across the four resource conditions. The correct action for this scenario is action 1. It can be observed that DQN and PPO agents frequently select the correct action, though not consistently. The SAC agent, however, fails to select the correct action, repeatedly choosing action 11 regardless of the resource conditions.



(a) Low-Low (Battery, Storage)

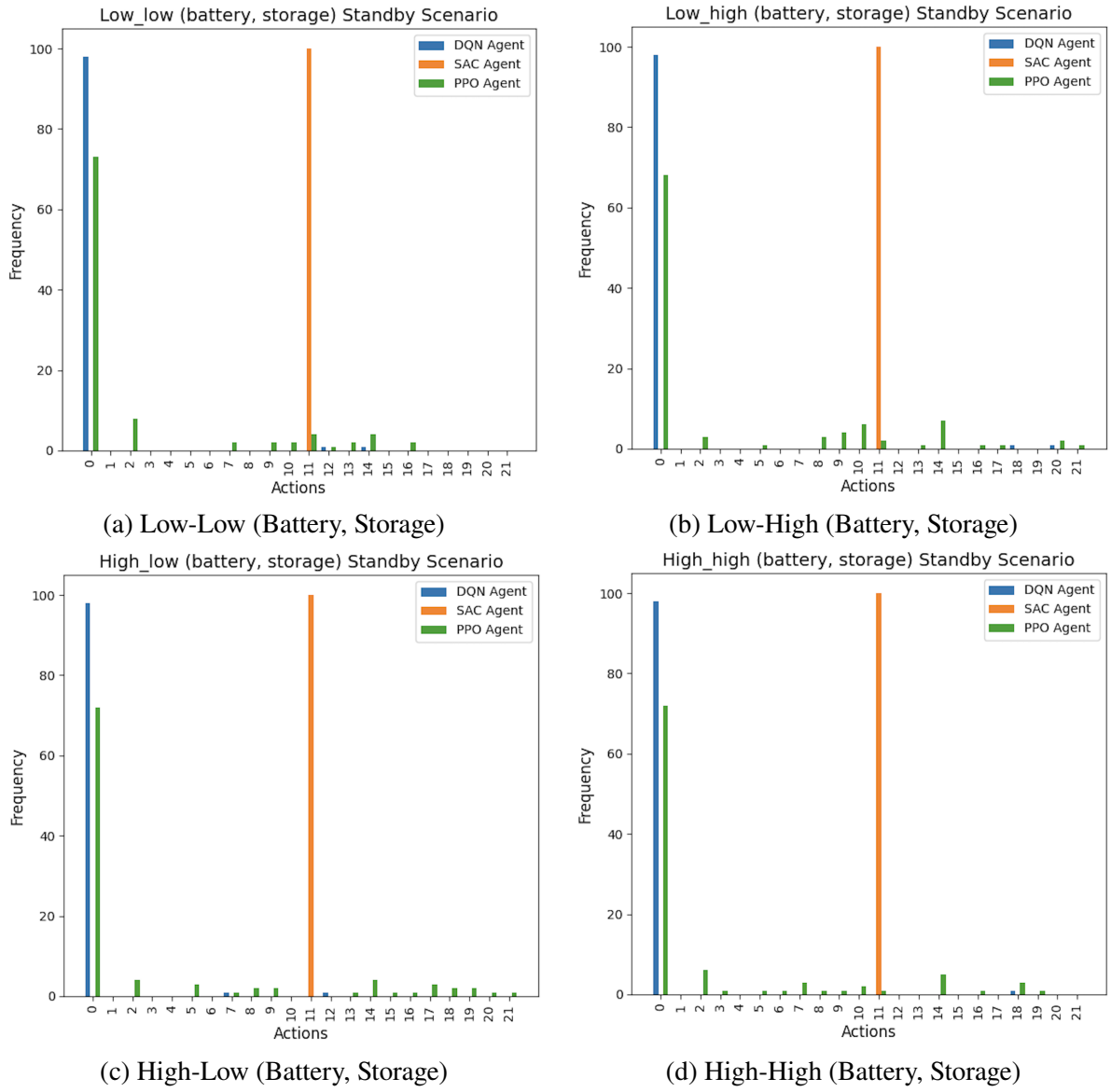
(b) Low-High (Battery, Storage)

(c) High-Low (Battery, Storage)

(d) High-High (Battery, Storage)

Figure 6.2 Observation validation scenarios under different resource conditions. The correct action in these scenarios is Action 2.

Figure 6.2 shows the results for the observation scenario, where action 2 is the correct choice. Both DQN and PPO agents show a tendency to select the correct action, with varying degrees of frequency depending on the resource conditions. Conversely, the SAC agent again consistently fails, choosing action 11 in all scenarios.



(a) Low-Low (Battery, Storage)

(b) Low-High (Battery, Storage)

(c) High-Low (Battery, Storage)

(d) High-High (Battery, Storage)

Figure 6.3 Stand-by validation scenarios under different resource conditions. The correct action in these scenarios is Action 0.

Finally, Figure 6.3 presents the stand-by scenario, where the optimal action is to take no action (action 0). Here, both DQN and PPO agents correctly choose the stand-by action more frequently, although their performance is still inconsistent. The SAC agent continues its pattern of selecting action 11, demonstrating a lack of adaptability to different situations.

In conclusion, while both PPO and DQN agents show potential for improvement and fine-tuning of their policies, the SAC agent's performance suggests that its policy is fundamentally flawed, as it consistently fails to adapt to the scenarios presented. Nevertheless, the results from the SAC agent are included in the following sections for comparative analysis.

6.2 Comparative Performance Analysis

In Figures 6.4, 6.5, and 6.6, we observe a visual comparison of key parameters across the three coordination models. These radar charts provide a comprehensive overview of how each algorithm performed under different

coordination scenarios by comparing metrics such as adjacency matrix percentage, global observation counts, global communication counts, total data transmitted, and battery available at the end of the simulation.

The parameters (all normalized between 0 and 1) being compared are critical in evaluating the effectiveness of each algorithm:

- **Adjacency Matrix Percentage**: this parameter represents the proportion of possible communication links between observers that were successfully established during the simulation. A value of 1 indicates that all observers were able to communicate with each other, while a value closer to 0 suggests that few or none of the communication links were successfully established. This metric is crucial for assessing the effectiveness of the coordination strategy in maintaining network connectivity among satellites.
- **Global Observation Counts**: this metric counts the total number of observations made by all observers of the various targets in the simulation. It indicates how effectively the agents were able to gather information about the environment, which is essential for mission success.
- **Global Communication Counts**: this parameter measures the total number of communications exchanged between observers. High communication counts suggest that the observers were actively sharing information, which is critical for coordinated decision-making in a distributed satellite network.
- **Total Data Transmitted**: this metric quantifies the total amount of data transmitted between observers during the simulation. Effective coordination strategies should maximize this value, as more data transmission typically translates to better coordination and information sharing.
- **Battery Available**: this parameter indicates the amount of battery power remaining at the end of the simulation. Efficient algorithms should aim to conserve battery life while still achieving high performance in other metrics, as energy management is vital in satellite operations.

In the unconstrained decentralized coordination scenario (Figure 6.4), the PPO algorithm demonstrated superior performance across most metrics. Notably, PPO excelled in maintaining battery life, indicating its efficiency in energy management while still achieving high values in adjacency and global observation counts. This suggests that PPO is able to balance effective communication and observation with energy conservation, making it particularly well-suited for unconstrained decentralized environments where resource management is crucial. Additionally, PPO showed strong performance in data transmission, which is vital for maximizing information sharing in such scenarios. The DQN algorithm also performed well, though it slightly lagged behind PPO in terms of observation counts, suggesting that while DQN is effective, it may prioritize different aspects of the mission compared to PPO. The Monte Carlo baseline and SAC both showed considerably lower performance, with SAC particularly struggling across all metrics, reinforcing its inadequacy in this context.

The centralized coordination model (Figure 6.5) presented a different set of challenges for the algorithms. Here, PPO continued to perform well, especially in data transmitted, though its performance in adjacency and battery conservation was somewhat reduced compared to the unconstrained decentralized coordination scenario. This reduction suggests that PPO may not adapt as effectively to environments where a central node dictates coordination, potentially due to the restrictive nature of centralized control. Interestingly, the Monte Carlo method showed competitive performance in global observation counts and adjacency, likely due to the stochastic nature of its actions occasionally resulting in favorable communication patterns. The DQN algorithm, however, struggled significantly in this configuration, particularly in terms of data transmission and global communication counts, indicating that it may not generalize well to centralized environments. As expected, SAC continued to underperform, confirming its unsuitability for such tasks.

In the constrained decentralized coordination scenario (Figure 6.6), where no central node exists, the PPO algorithm again demonstrated a balanced performance across all metrics, although it was not as dominant as in the unconstrained decentralized coordination scenario. This suggests that while PPO is robust, its effectiveness slightly diminishes in more complex and more restricted environments. The DQN algorithm showed some improvement in adjacency and global communication counts compared to its performance in the centralized scenario but still was short in data transmitted, indicating partial adaptability to decentralized structures. The

Monte Carlo approach remained competitive in global observation counts, which may be attributed to random variations that occasionally result in effective observation patterns. Again, SAC continued to show poor performance, highlighting its inability to establish effective policies.

Overall, PPO consistently demonstrated superior performance across different coordination models, particularly excelling in unconstrained decentralized environments (trained case) where its energy efficiency and balanced approach to communication and observation provided clear advantages. In contrast, DQN showed potential but struggled with generalization across different coordination structures, particularly in centralized environments, suggesting that further tuning or training in these kinds of environments may be needed to improve its robustness. The Monte Carlo baseline provided a useful comparison, occasionally performing competitively in less structured environments, but overall it lacked the targeted efficiency of trained algorithms. Finally, SAC consistently failed to perform across all coordination models, indicating that it may require significant modifications or alternative approaches to be viable in similar scenarios.

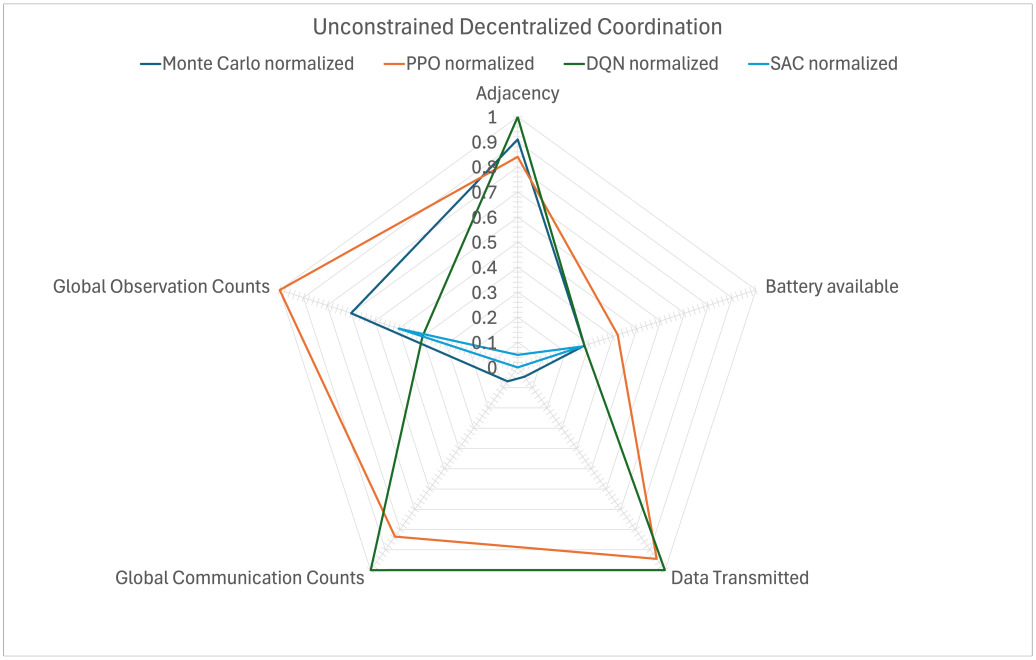


Figure 6.4 Comparison of key parameters in the unconstrained decentralized coordination model.

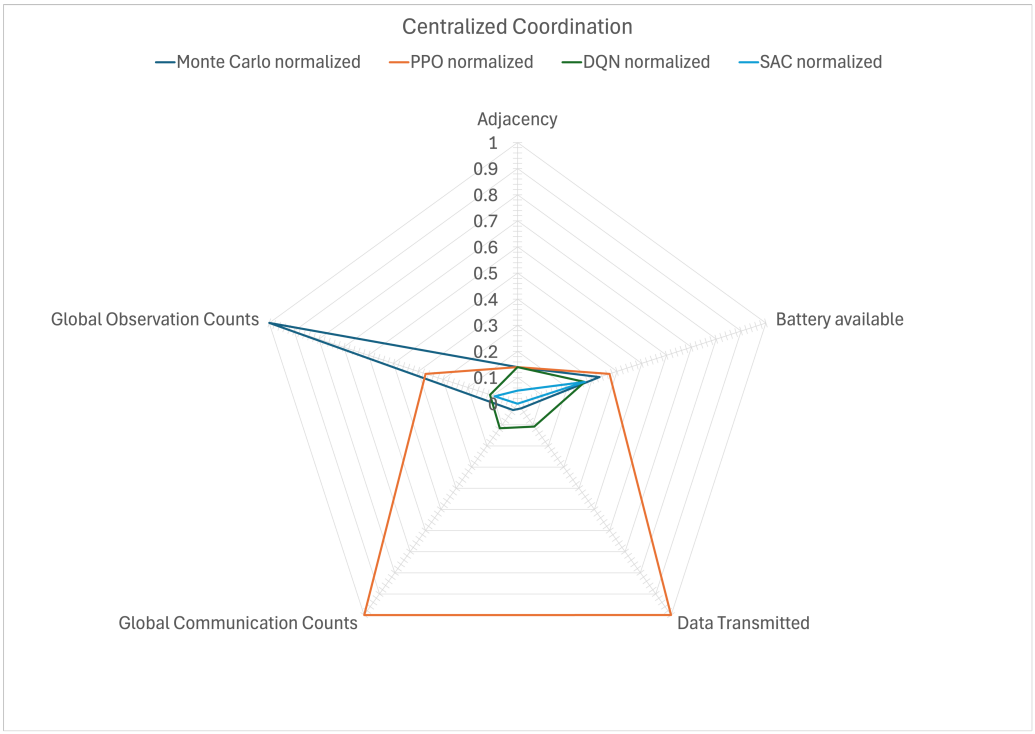


Figure 6.5 Comparison of key parameters in the centralized coordination model.

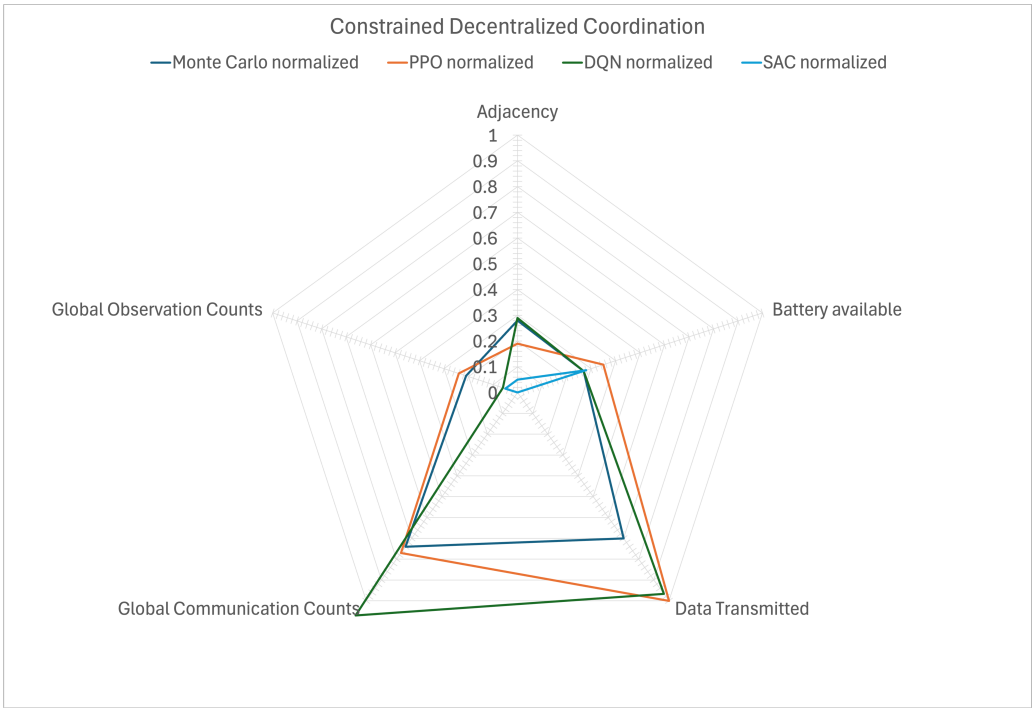


Figure 6.6 Comparison of key parameters in the constrained decentralized coordination model.

6.2.1 Resources Management

The PPO algorithm demonstrated a clear superiority in resource management, particularly in terms of extending battery life. Across the different scenarios, PPO achieved an impressive increase in battery longevity, ranging from 12% to as much as 55% compared to the other algorithms. This substantial enhancement in energy efficiency indicates that PPO is highly effective in optimizing power consumption, which is critical for prolonging

the operational lifespan of satellites in mission scenarios. However, it's important to note that storage utilization remained consistent across all tested algorithms and scenarios. This consistency suggests that while PPO excels in managing dynamic resources like battery power, more static resources such as storage did not benefit from similar optimization, possibly because the mission scenarios did not push storage capacity to its limits as we planned.

6.2.2 Communication Tasks

The performance of the DQN algorithm was strong in the unconstrained decentralized coordination scenario for which it was trained. However, its ability to generalize to other coordination configurations, such as centralized and constrained decentralized setups, was limited, resulting in a noticeable decline in performance. This drop highlights a potential issue with DQN which may require additional training or fine-tuning to perform optimally in environments that differ significantly from its training conditions.

In contrast, PPO displayed remarkable adaptability, outperforming all other algorithms in the centralized coordination configuration, even though it was not explicitly trained for this type of network. This adaptability suggests that PPO has a robust capacity for handling dynamic and restrictive coordination environments, making it a versatile choice for varying mission parameters.

Figures 6.7 and 6.8 further illustrate the differences in how these algorithms handle communication tasks. In the unconstrained decentralized coordination scenario, both PPO and DQN prioritize data transmission as a means of maximizing rewards. This is evident from the high total data transmitted, as seen in Figure 6.7. However, in more restrictive coordination scenarios, both algorithms shift their focus towards observation tasks, as seen by the lower number of communications per satellite in Figure 6.8. This shift indicates a strategic adjustment where the algorithms seek to maximize their performance based on the constraints of the environment.

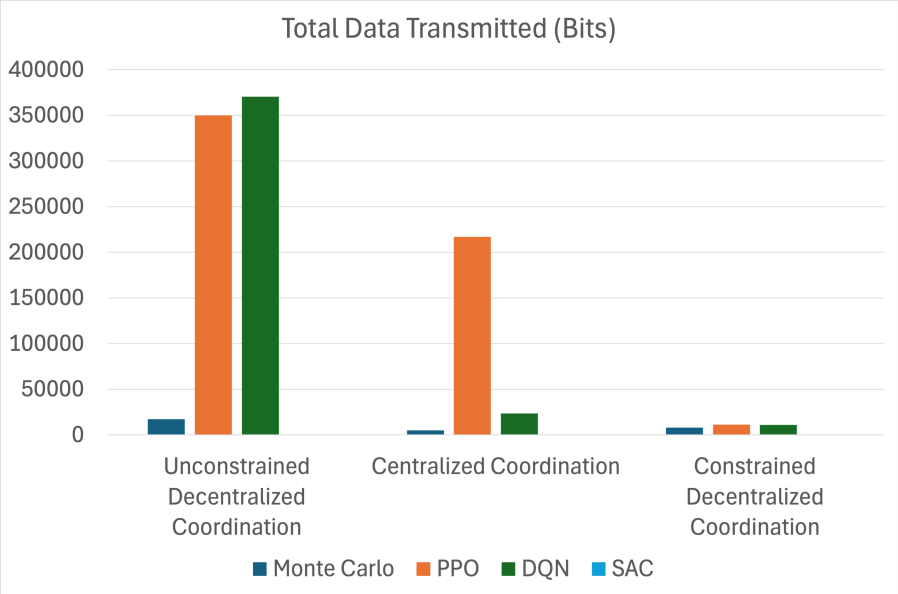


Figure 6.7 Total Data Transmitted (Bits) across different coordination scenarios.

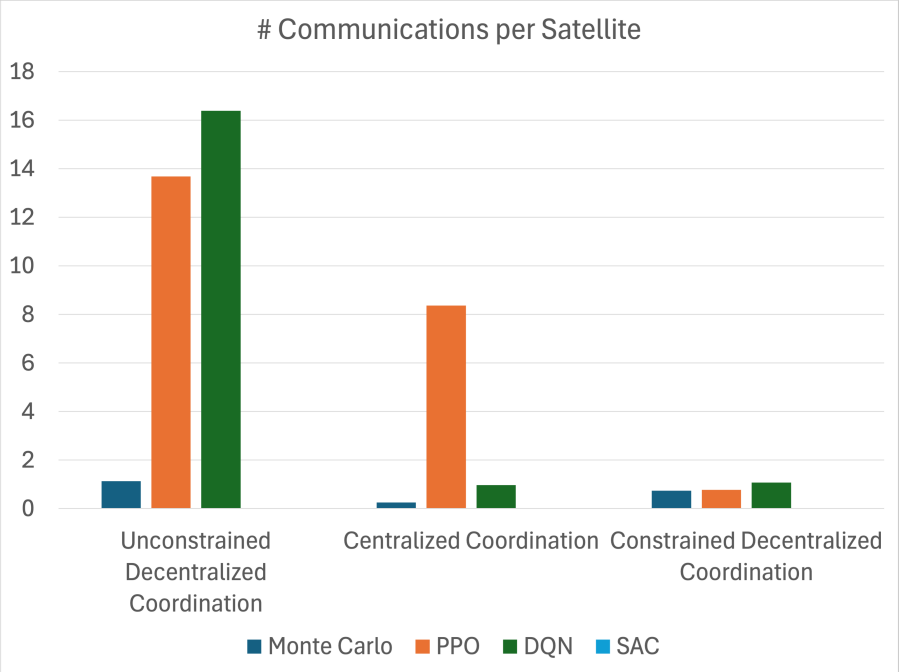


Figure 6.8 Number of Communications per Satellite across different coordination scenarios.

Therefore, the strengths of PPO in both resource management and communication adaptability are clear, making it a robust choice for satellite operations that involve varying degrees of coordination freedom. Meanwhile, the performance of DQN suggests that while it can be highly effective in familiar environments, it may require additional tuning or training to achieve similar results in less familiar scenarios. The clear distinction in how these algorithms prioritize tasks based on coordination constraints offers valuable insights for future mission planning and algorithm selection.

6.2.3 Observation Tasks

While the quality of observations is highly influenced by external factors such as random orbit generation for both observers and targets, making it somewhat arbitrary, the quantity of observations can still provide valuable insights into the behavior of the algorithms. Figure 6.9 illustrates the number of observations per satellite across different coordination scenarios.

In more restrictive coordination environments, both DQN and, more notably, PPO strategically increase their observation count to accumulate more rewards. This behavior indicates that these algorithms adapt to the constraints by focusing on maximizing the observable data they can gather, which directly translates into higher rewards. This strategic adaptation is crucial in environments where communication is limited, as it allows the algorithms to continue performing effectively despite the constraints.

In contrast, under the fully decentralized coordination scenario, these algorithms shift their focus from maximizing observations to prioritizing data transmission. This shift suggests that in environments where coordination is not a limiting factor, PPO and DQN recognize the higher value of transmitting information across the network rather than increasing the number of observations. This ability to adapt their strategy based on the coordination environment demonstrates the flexibility and sophistication of these algorithms.

The Monte Carlo baseline, on the other hand, exhibits significant variability in the number of observations. This variability is expected given the random nature of the Monte Carlo approach, which does not employ a strategic method for maximizing observations or rewards. Despite this variability, PPO consistently outperforms the baseline, reinforcing its effectiveness in both restrictive and open coordination scenarios. The consistency of

PPO in optimizing for the overall reward, regardless of the scenario, highlights its robustness and reliability in various operational environments.

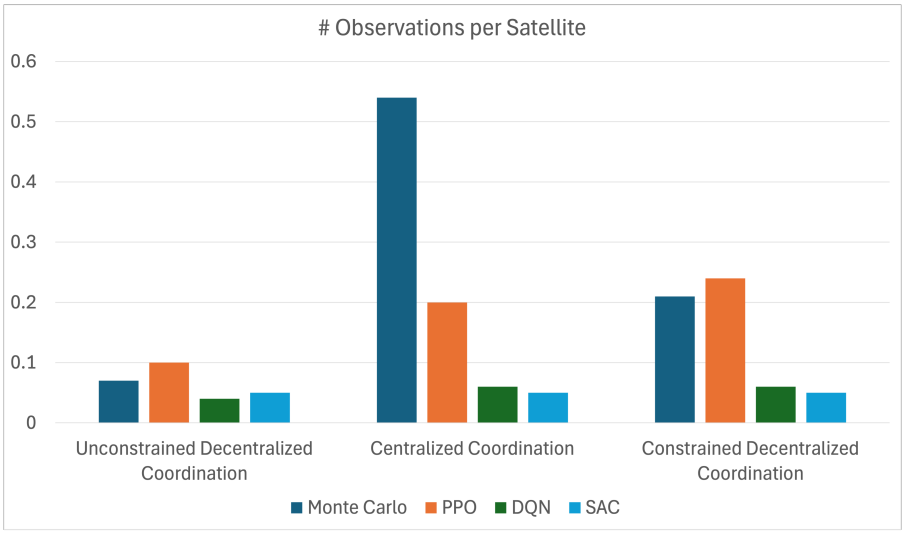


Figure 6.9 Number of Observations per Satellite across different coordination scenarios.

6.3 Overall Mission Performance

The overall mission performance is assessed through various metrics, including observation status, mission duration, and rewards. These metrics provide a comprehensive view of how well each algorithm performed in terms of fulfilling the primary mission objectives.

Observation status is a critical metric, as it directly reflects the mission's success in terms of observing and sharing information about the targets. The observation status is defined as follows:

- **0 - Undetected**: the target has not been detected by any satellite.
- **1 - Detected**: the target has been detected, but it is not currently being observed.
- **2 - Being Observed**: the target is actively being observed by a satellite.
- **3 - Fully Observed**: the target has already been observed.

The closer the average observation status is to 3, the more successful the mission has been in propagating information across the network. This metric provides a nuanced understanding of how effectively the satellites were able to fulfill their primary observation tasks.

Figure 6.10 depicts the average observation status achieved by all participating satellites across different coordination scenarios. This status is crucial as it represents the degree to which the mission's primary objective—observing and sharing information—has been fulfilled.

Contrary to expectations, the Monte Carlo baseline, which employs random actions, surprisingly achieves the highest observation status across all coordination scenarios. This outcome suggests that the inherent randomness in Monte Carlo may occasionally lead to favorable outcomes, particularly in dynamic and less predictable environments. However, this does not imply strategic superiority, as Monte Carlo lacks consistent decision-making processes.

PPO and DQN, while designed to optimize observation and communication strategies, perform well but do not surpass Monte Carlo in observation status. Notably, PPO performs consistently across all coordination types, indicating its strong adaptability to different coordination constraints. In the fully decentralized coordination scenario, both PPO and DQN show solid performance, although neither matches the peak observation status

of Monte Carlo. In centralized coordination, PPO slightly edges out DQN, showing better resource allocation and communication management. In partially decentralized coordination, PPO again maintains its consistency, whereas DQN shows a slight drop in performance, indicating potential difficulties in managing resources without a central node.

SAC, on the other hand, significantly underperforms in all coordination scenarios. This poor performance was expected and highlights SAC's struggle with the resource-sharing and coordination tasks essential for achieving better observation statuses in these kinds of environments.

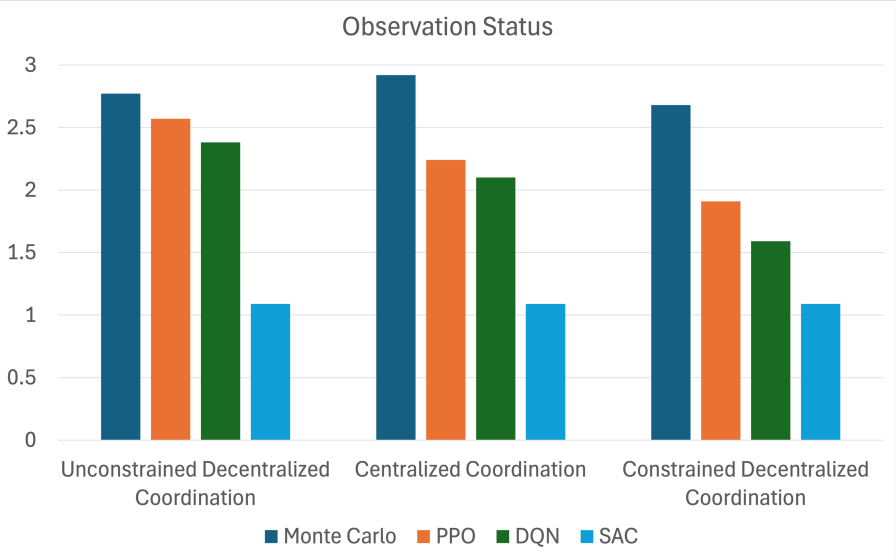


Figure 6.10 Mean observation status achieved in the simulation.

The duration of the simulation, illustrated in Figure 6.11, offers important insights into the efficiency of each algorithm. A longer mission duration typically suggests that the satellites are effectively managing their resources, thereby delaying the point at which a satellite runs out of battery or storage. Among the algorithms, DQN stands out by significantly extending the mission duration, particularly in the unconstrained fully decentralized coordination scenario. This indicates that DQN is highly efficient in resource management, potentially prioritizing battery life to maximize mission longevity. However, despite the prolonged mission duration, the battery levels at the end of the simulation are considerably lower in DQN than in PPO, as noted in previous sections. This discrepancy suggests that while DQN may excel in overall resource management, PPO performs better in maintaining individual battery levels.

Overall, the trade-off between mission duration and rewards becomes evident when considering the outcomes in Figure 6.12. While DQN extends the mission duration, this does not necessarily translate into the highest rewards. This implies that simply prolonging the mission may not align with an optimal reward strategy. In contrast, PPO achieves a balance between mission duration and effective completion of objectives, securing high rewards while maintaining reasonable mission lengths.

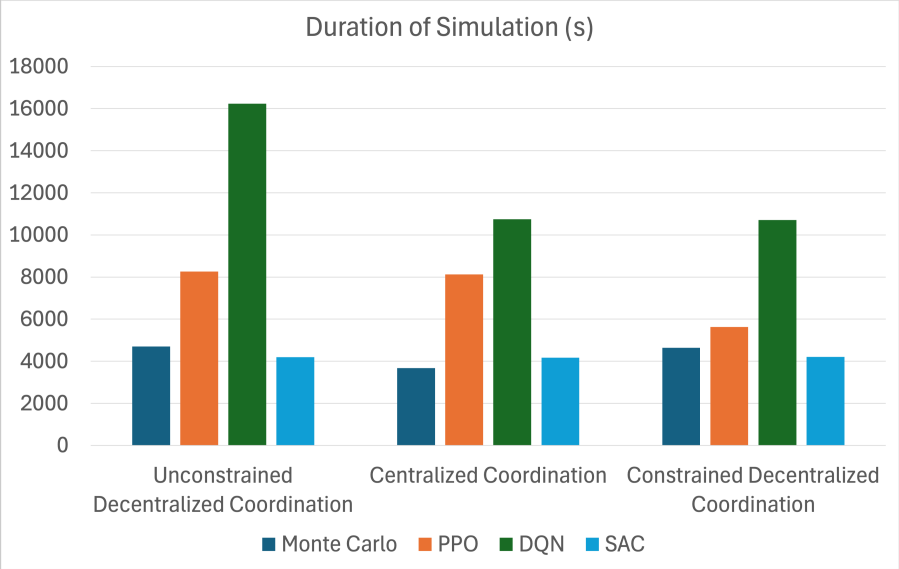


Figure 6.11 Duration of the simulation in seconds.

Figure 6.12 highlights the disparities in reward allocation among the different algorithms. The Monte Carlo approach, as expected, results in lower rewards due to its random action selection, which lacks the strategic decision-making evident in the learning-based algorithms. Interestingly, both PPO and DQN achieve positive rewards, yet there is a notable difference in how these rewards correlate with mission performance metrics like observation status and mission duration. This discrepancy suggests that the reward function may require further refinement to ensure it aligns more closely with mission-critical objectives, such as maximizing observation status and ensuring efficient resource management.

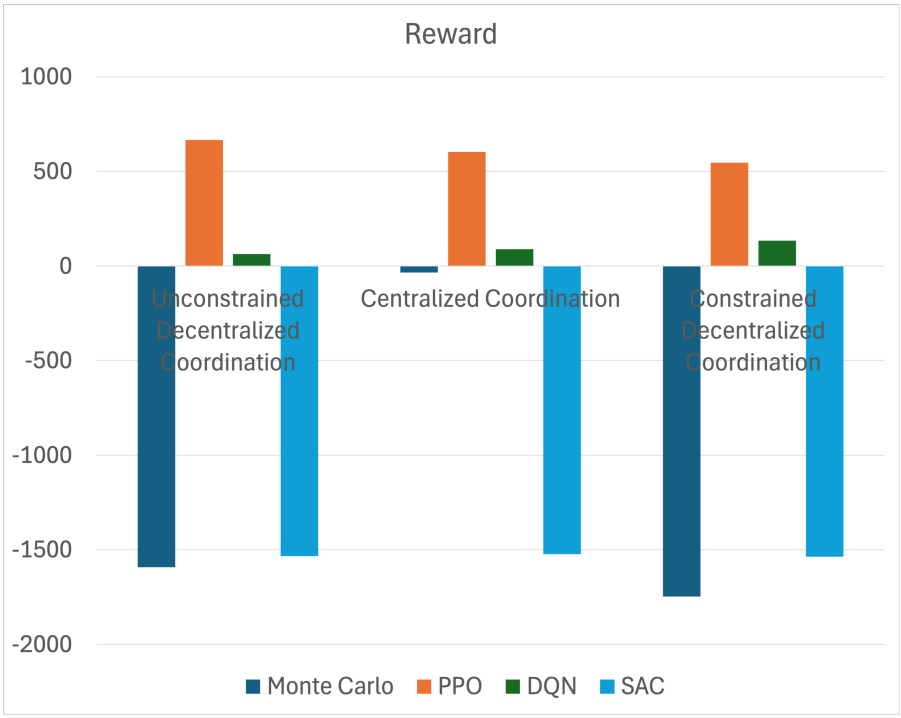


Figure 6.12 Rewards achieved by different algorithms.

In conclusion, while PPO demonstrates a strong balance between maximizing rewards and achieving mission objectives, DQN excels in prolonging mission duration but may need further tuning to align its strategy more

closely with reward maximization. The surprisingly strong performance of the Monte Carlo baseline in terms of observation status, despite its lack of strategic planning, underscores the importance of refining the reward structure and potentially reconsidering the weighting of different mission objectives in the learning process. The consistent underperformance of the SAC algorithm across all scenarios suggests that it may not be well-suited for the discrete action space and coordination-focused environment of the FSS. This underperformance indicates a need for further investigation into its applicability and perhaps a re-evaluation of its use in similar contexts.

7 Application to Real Satellites

To evaluate the feasibility of deploying RLs algorithms on CubeSat platforms, particularly those with limited computational resources, we conducted performance tests on different computing environments. These tests measured the average time required to compute a single action, a crucial factor in determining the practical application of these RL models in real-time satellite operations.

The tests were performed on a MacBook Pro M2 (2022) for benchmarking and an NVIDIA Jetson AGX Orin 32GB, with the latter running in limited power mode (30W) to simulate more realistic space conditions [51]. The tests involved executing at least 72,000 actions, which corresponds to one hour of simulated time with 20 agents.

Device	PPO (ms)	DQN (ms)	SAC (ms)
MacBook Pro M2	4.13	5.55	6.64
Jetson AGX Orin (30W)	10.26	11.14	13.33

Table 7.1 Average computation time per action for different RL algorithms on various hardware platforms.

The results indicate that the MacBook Pro M2 significantly outperforms the Jetson AGX Orin in terms of computation time, which is expected given the M2's more powerful architecture designed for general-purpose computing. However, in this case the Jetson AGX Orin's performance is of particular interest due to its relevance for onboard satellite processing.

Even in limited power mode, the Jetson AGX Orin achieved average computation times ranging from 10.26 ms for PPO to 13.33 ms for SAC. These computation times are within an acceptable range for real-time processing in many satellite operations, where timely decision-making is crucial. The results suggest that deploying RL algorithms on CubeSats equipped with a Jetson AGX Orin is feasible for real-time operations, such as collision avoidance, dynamic resource allocation, or adaptive communication strategies.

The Jetson's limited power mode provides a conservative estimate of performance, which is vital for understanding its behavior in the energy-constrained environment of a CubeSat. While the computation time increases slightly compared to the MacBook Pro M2, this trade-off is reasonable given the power constraints inherent to space hardware.

In conclusion, the performance metrics suggest that deploying RL algorithms such as PPO, DQN, and SAC on CubeSats with Jetson AGX Orin hardware is not only feasible but also practical for real-time satellite operations. Future work may focus on further optimizing these algorithms for reduced power consumption and exploring their integration into full satellite systems.

8 Conclusions

This thesis has advanced the understanding and implementation of decentralized reinforcement learning (DRL) for satellite constellations, particularly focusing on enhancing the federation capabilities of satellite networks. By decentralizing decision-making and prioritizing collective goals, this approach enables more scalable, resilient, and autonomous operations. The integration of artificial neural networks, particularly through DRL, has shown significant potential for improving both the efficiency and speed of decision-making processes, surpassing traditional planning methods [16, 12]. This advancement underscores the feasibility of applying DRL in real satellite operations, demonstrating the potential for deploying such technologies in embedded systems within satellites.

The comparative analysis of key parameters across different coordination models, as summarized in Table 8.1, further reinforces the effectiveness of DRL. The table highlights how different algorithms perform under various coordination scenarios, showcasing the adaptability and robustness of the PPO algorithm, in particular. These findings emphasize the readiness of DRL techniques for practical application in the increasingly complex and dynamic environments of satellite constellations.

	Adjacency	Battery Available	Data Transmitted (bits)	Global Communication Counts	Global Observation Counts	Global Observation Status	Reward
Unconstrained (Fully) Decentralized Coordination							
PPO	0.84	0.42	349747.1	13.68	0.10	2.57	666.48
DQN	1.00	0.28	370352.65	16.39	0.04	2.38	62.96
SAC	0.05	0.27	0	0	0.05	1.09	-1533.7
Centralized Coordination							
PPO	0.14	0.37	216930.96	8.37	0.20	2.24	603.63
DQN	0.14	0.27	23477.17	0.97	0.06	2.10	88.58
SAC	0.05	0.27	0	0	0.05	1.09	-1522.16
Constrained (Partially) Decentralized Coordination							
PPO	0.19	0.35	11249.72	0.77	0.24	1.91	545.88
DQN	0.29	0.27	10872.72	1.07	0.06	1.59	134.43
SAC	0.05	0.28	0	0	0.05	1.09	-1536.17

Table 8.1 Key Parameters Comparison of DRL Algorithms under Different Coordination Models

8.1 Research Questions Revisited

RQ1: How does the application of reinforcement learning differ between centralized and decentralized coordination models in satellite federations, and what are the respective impacts on scalability, resilience, and adaptability?

Reinforcement learning demonstrates distinct advantages when applied to decentralized coordination models in satellite federations, particularly in terms of scalability, resilience, and adaptability. Decentralized RL approaches, such as PPO, are well-suited for large-scale, dynamic environments where satellites operate independently but share collective goals. These models enable more scalable and resilient operations by allowing each satellite to make decisions based on local observations without the need for a global state, thereby reducing the computational load and avoiding single points of failure. Moreover, decentralized RL excels in adaptability, as it allows the system to respond to changes in the environment quickly and efficiently.

In contrast, centralized RL approaches offer a more comprehensive view of the environment, which can be advantageous for highly interconnected systems requiring a unified strategy. However, the scalability of centralized models is limited, as they require extensive computational resources to maintain and process global

state information. This can lead to inefficiencies and potential bottlenecks as the system grows in complexity. Therefore, while centralized coordination can be effective for smaller or more static satellite federations, decentralized RL models are generally more robust and efficient for large-scale, autonomous operations.

RQ2: How do various reinforcement learning algorithms compare in terms of computational efficiency and performance when applied to decentralized versus centralized coordination in satellite networks?

The thesis reveals that different reinforcement learning algorithms exhibit varying levels of performance and computational efficiency depending on whether they are applied in decentralized or centralized coordination models. PPO consistently shows strong performance across both decentralized and centralized settings, with particular strengths in adaptability and resource management. PPO balances mission objectives effectively, achieving high rewards while maintaining efficient resource utilization.

DQN, on the other hand, is particularly effective at extending mission duration, making it suitable for scenarios where longevity is critical. However, DQN's performance can decline in partially decentralized environments where the lack of global state information may lead to suboptimal decisions. Despite this, DQN remains a strong candidate for applications where stability and computational efficiency are prioritized.

SAC underperforms across both decentralized and centralized models, particularly in environments requiring discrete actions. SAC struggles with the specific demands of satellite coordination tasks, such as effective resource sharing and communication management. The thesis suggests that SAC may not be well-suited for these types of missions without significant modifications to its approach.

Overall, PPO emerges as the most versatile and robust algorithm, capable of adapting to various coordination models while maintaining high performance and computational efficiency. DQN, while less adaptable, offers strong computational efficiency and mission longevity, making it a viable alternative in specific contexts.

RQ3: How do observation tasks impact the performance of reinforcement learning algorithms, particularly in applications such as object detection, and what are the key considerations for optimizing performance in such scenarios?

Observation tasks in object detection within space environments are highly dynamic and unpredictable, with observation windows often lasting only a few seconds. This unpredictability poses a significant challenge for reinforcement learning algorithms, which must quickly adapt to changing conditions. The thesis demonstrated that while DRL algorithms like PPO and DQN can effectively manage these tasks, their performance is heavily influenced by the availability and quality of observations. Key considerations for optimizing performance include improving the algorithms' ability to prioritize critical observations and ensuring that the reward functions are closely aligned with mission objectives. Additionally, incorporating advanced techniques such as attention mechanisms could further enhance the ability of these algorithms to focus on the most relevant data.

8.2 Benefits of DRL in Satellite Operations

The integration of DRL into satellite operations offers several key benefits:

- **Enhanced Autonomy**: DRL enables satellites to make real-time decisions autonomously, reducing reliance on ground-based control and minimizing response times.
- **Optimized Resource Utilization**: by learning optimal policies, DRL helps satellites efficiently manage limited resources, such as power and bandwidth, improving mission outcomes.
- **Robustness to Environmental Changes**: DRL agents can adapt to unforeseen changes in the environment, ensuring continued operation and mission success despite dynamic conditions.
- **Scalability**: the use of multi-agent reinforcement learning (MARL) allows for scalable solutions in satellite swarms, where multiple agents can coordinate and optimize their actions collectively.

In conclusion, the application of reinforcement learning, particularly DRL, represents a significant advancement in the field of satellite operations. By enabling autonomous, adaptive, and efficient decision-making, DRL holds the potential to revolutionize satellite missions, paving the way for more resilient and capable space systems.

8.3 Major Contributions

This work contributes to the state of the art in satellite constellation management by demonstrating the ability to optimize resources and improve mission outcomes using DRL. Furthermore, it provides a modular framework that can be expanded and customized to train multi-agent reinforcement learning policies for autonomous satellite operations. This framework not only supports current satellite missions but also serves as a foundation for future developments in satellite capabilities. An open-source repository of the project is also available.

8.4 Future Work

Future research should focus on incorporating additional real-world factors, such as line-of-sight constraints, atmospheric interference, and sensor limitations, to further enhance the realism of the simulations. These factors are critical for developing DRL algorithms that can be effectively deployed in actual satellite missions, where such constraints significantly impact performance.

Moreover, integrating maneuvering capabilities, including collision-avoidance and general spacecraft maneuvering, as additional actions within the DRL framework could lead to more sophisticated decision-making processes. This would allow satellite federations to autonomously navigate complex environments, improving their overall resilience and operational efficiency.

Another important area of future work is the further scaling of the training process. Leveraging multi-node supercomputing resources could significantly accelerate the training of DRL models, enabling the development of more complex and capable algorithms. This approach would allow for the handling of larger federations and more intricate mission scenarios by distributing the computational load across multiple nodes in a supercomputer. Such scalability is crucial for testing and refining the algorithms to meet the demands of real-world satellite operations.

The reward-driven behavior of DRL algorithms has proven to be a key factor in their success, and thus, the design and refinement of reward functions must be deeply addressed. Future research should explore more sophisticated reward structures that better align with mission objectives, ensuring that the DRL algorithms are not only optimizing for performance but also adhering to operational constraints and goals.

Additionally, the potential applications of DRL in other satellite missions should be explored. Beyond satellite federations for communication and observation, DRL could be applied to Earth observation, navigation, and even space exploration missions. These applications would benefit from the autonomous decision-making capabilities that DRL provides, enabling more efficient and effective mission planning and execution.

Improving visualization tools for simulation results could also significantly enhance the accessibility and impact of this research. Effective visualization would allow stakeholders with less technical expertise to better understand the implications of DRL algorithms in satellite federation management, fostering broader adoption of these advanced technologies.

Finally, a deeper exploration into hybrid centralized-decentralized frameworks could yield even more robust and versatile satellite federations. The refinement of these frameworks, along with the continuous improvement of reward functions, could enable satellites to perform increasingly complex missions in dynamic space environments, paving the way for more autonomous and adaptive space operations.

A Annex

A.1 Class Diagrams

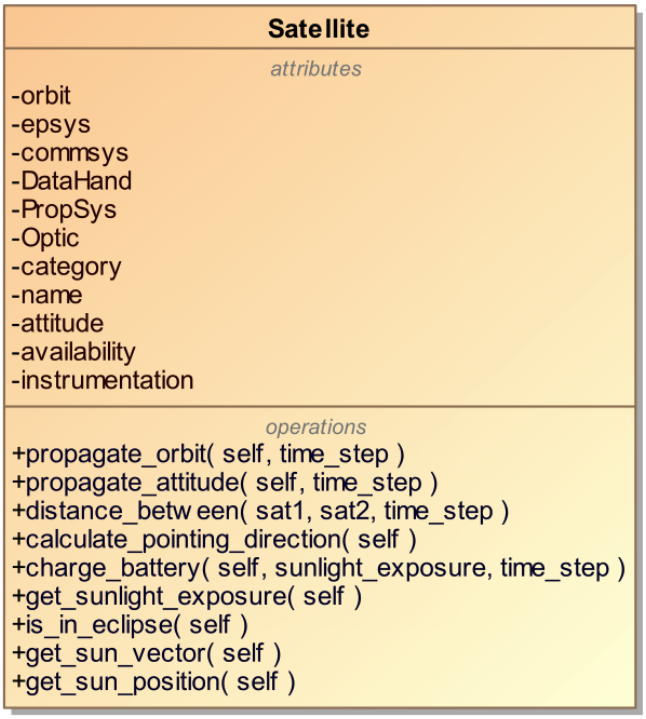


Figure A.1 Satellite class attributes and methods.

Figure A.2 Observer satellite class attributes and methods.

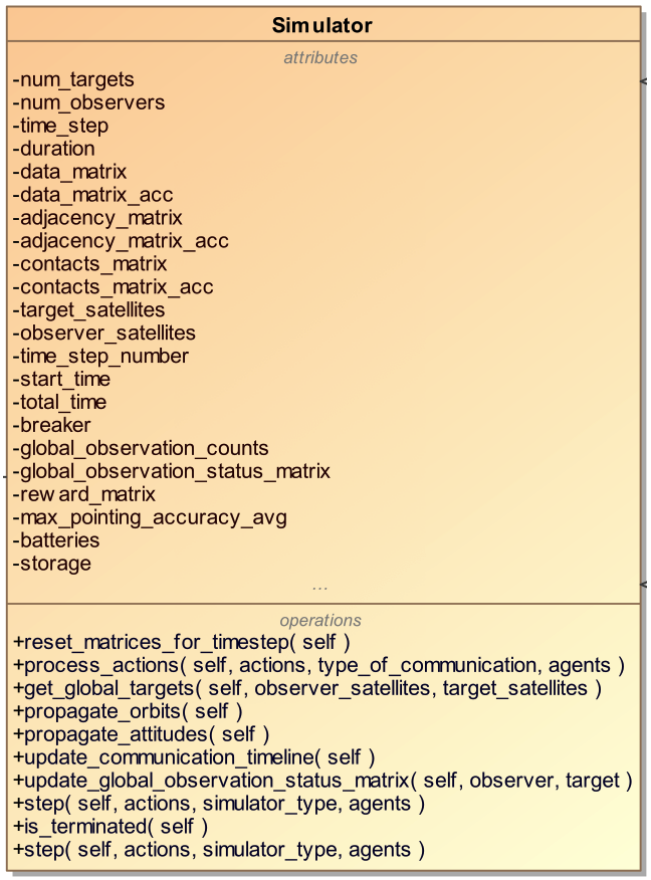


Figure A.3 Simulator class attributes and methods.

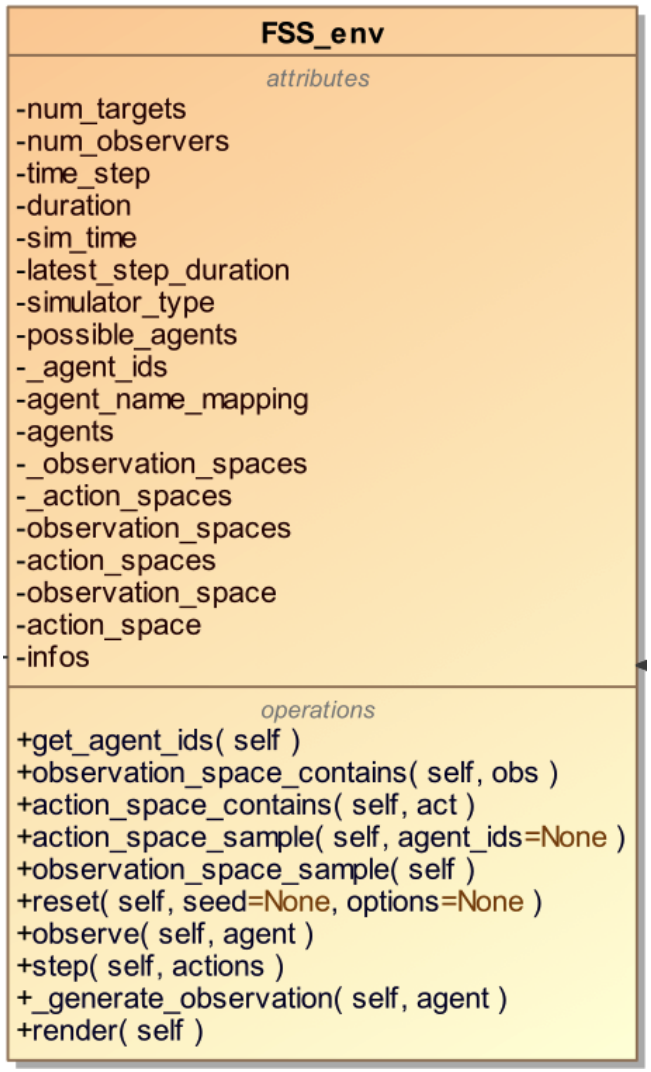


Figure A.4 Federated Satellite System Environment class attributes and methods.

A.2 Sequence Diagram

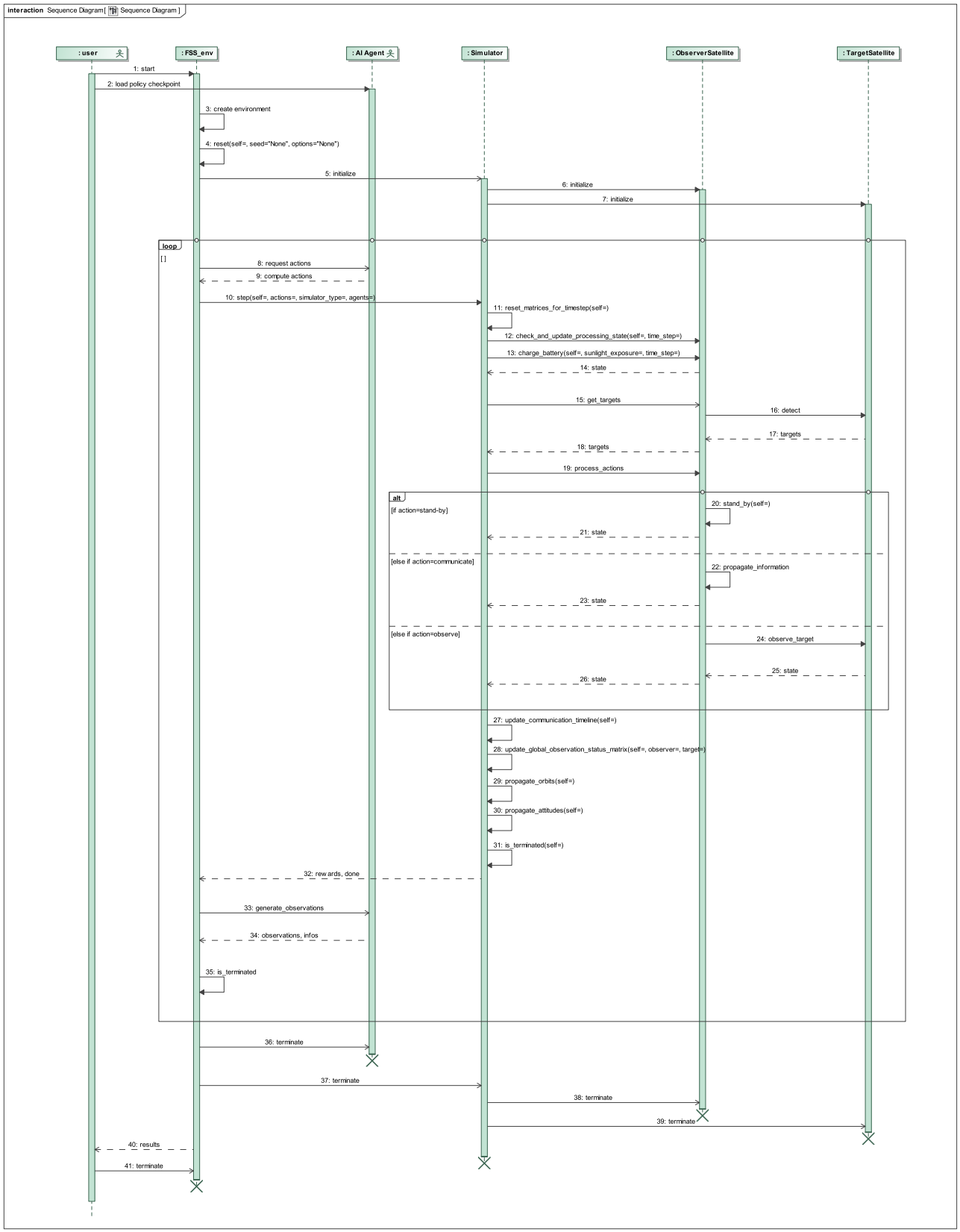


Figure A.5 Federated Satellite System Environment sequence diagram.

A.3 Validation and testing of simulator

```
    ######### Attitude propagation test
       ########################################

Initial attitude parameters:
Quaternion: [1. 0. 0. 0.], Angular velocity: [0.01 0.01 0.01]

Custom attitude propagator results after propagation:
Quaternion: [0.9999625  0.00499981 0.00499981 0.00499981]

SciPy solver results after propagation:
Quaternion: [0.99985001 0.00999938 0.00999938 0.00999938]

Max Quaternion error: 0.008660226974575889
.
######### Battery propagation test
   ########################################

Initial battery state:
Energy Available: 50.0 Wh, Solar Panel Size: 0.12 m^2, Efficiency: 0.3,
   Solar Constant: 1370 W/m^2

Power consumption constant (1 for normal, 3 for faster discharging): 3
Satellite energy depleted (observer). Terminating simulation.

Final Energy Available: 0.0 Wh
.
######### Distance and Effective Data Rate Test
   ########################################

Distance between satellites: 100000.00 m
Effective Data Rate: 19200.00 bits/sec
Available and new info to communicate: True
Can communicate with other: True
observer can communicate with observer
observer has communicated with observer and transmitted 3000.00 bits of
   data
Reward: 1.1, Data transmitted: 3000, Steps: 1

Distance between satellites: 500103.31 m
Effective Data Rate: 19200.00 bits/sec
Available and new info to communicate: True
Can communicate with other: True
observer can communicate with observer
observer has communicated with observer and transmitted 3000.00 bits of
   data
Reward: 1.1, Data transmitted: 3000, Steps: 1

Distance between satellites: 2300022.46 m
Effective Data Rate: 19200.00 bits/sec
Available and new info to communicate: True
```

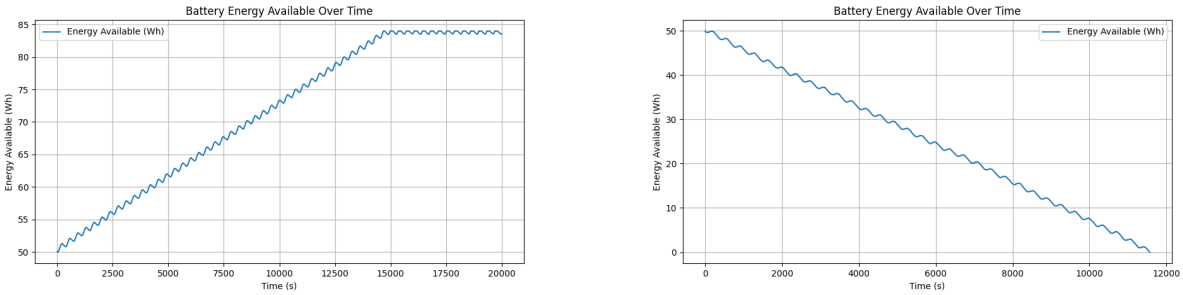
```
Can communicate with other: True
observer can communicate with observer
observer has communicated with observer and transmitted 3000.00 bits of
    data
Reward: 1.1, Data transmitted: 3000, Steps: 1

Distance between satellites: 2325022.22 m
Effective Data Rate: 0.00 bits/sec
Available and new info to communicate: True
Can communicate with other: False
Reward: -0.01, Data transmitted: 0, Steps: 1

Distance between satellites: 2350021.98 m
Effective Data Rate: 0.00 bits/sec
Available and new info to communicate: True
Can communicate with other: False
Reward: -0.01, Data transmitted: 0, Steps: 1

Distance between satellites: 2400021.52 m
Effective Data Rate: 0.00 bits/sec
Available and new info to communicate: True
Can communicate with other: False
Reward: -0.01, Data transmitted: 0, Steps: 1

.
######## Starting Dummy Simulation ########

Observer-1 is observing Target-1 with pointing accuracy 1.00
Observer-1 has observed Target-1 with pointing accuracy 1.00
Observer-1 can communicate with Observer-2
Observer-1 has communicated with Observer-2 and transmitted 1056.00 bits
    of data
Step 10000 done
Satellite energy or storage depleted (Observer-1). Terminating simulation
    .
Simulation terminated after 3.2085468769073486 seconds.
Forced termination at step 11042
Episode finished

Total steps: 11042
Total duration of episode: 3.208 seconds
Total reward: -97.000

Results:
Adjacency Matrix:
[[1 1]
 [1 1]]
Data Matrix:
[[   0. 1056.]
 [1056.    0.]]
Contacts Matrix:
```

```
[[1]
 [1]]
Global Observation Counts:
[[1]
 [0]]
Max Pointing Accuracy Avg:
[1.]
Global Observation Status Matrix:
[[3]
 [3]]
Batteries:
[0.         0.35775238]
Storage:
[0.64840965 0.8498539 ]
.
######### Observation Accuracy Test
   ##########################################

Testing positions (100 km apart, perfect alignment):
Observer 1: (7142846.101516889, -1129.3850175896164, 7380.564140244884)
Target: (7242846.101516889, -1129.3850175896164, 7380.564140244884)
observer is observing {'Name': 0} with pointing accuracy 1.00
observer has observed {'Name': 0} with pointing accuracy 1.00
Distance: 100000 m
Reward: [2.]
Pointing Accuracy: 1.00
Observation Status Matrix: [3]
Cumulative Pointing Accuracy: [[1.]
 [0.]]
Observation Counts: 1
Global Observation Counts: [[1]
 [0]]
Max Pointing Accuracy Avg: [1.]

Testing positions (263 km apart - on the edge of max distance, perfect
   alignment):
Observer 1: (7142846.101516889, -1129.3850175896164, 7380.564140244884)
Target: (7405846.101516889, -1129.3850175896164, 7380.564140244884)
observer is observing {'Name': 0} with pointing accuracy 1.00
observer has observed {'Name': 0} with pointing accuracy 1.00
Distance: 263000 m
Reward: [2.]
Pointing Accuracy: 1.00
Observation Status Matrix: [3]
Cumulative Pointing Accuracy: [[1.]
 [0.]]
Observation Counts: 1
Global Observation Counts: [[1]
 [0]]
Max Pointing Accuracy Avg: [1.]
```

```
Testing positions (264 km apart - just above the limit, perfect alignment
    ):
Observer 1: (7142846.101516889, -1129.3850175896164, 7380.564140244884)
Target: (7406846.101516889, -1129.3850175896164, 7380.564140244884)
Distance: 264000 m
Reward: -0.02
Pointing Accuracy: 0.00
Observation Status Matrix: [0]
Cumulative Pointing Accuracy: [[0.]
 [0.]]
Observation Counts: 0
Global Observation Counts: [[0]
 [0]]
Max Pointing Accuracy Avg: [0.]

Testing positions (200 km apart, 90   rotation):
Observer 1: (7142846.101516889, -1129.3850175896164, 7380.564140244884)
Target: (7342846.101516889, -1129.3850175896164, 7380.564140244884)
Distance: 200000 m
Reward: -0.02
Pointing Accuracy: 0.00
Observation Status Matrix: [0]
Cumulative Pointing Accuracy: [[0.]
 [0.]]
Observation Counts: 0
Global Observation Counts: [[0]
 [0]]
Max Pointing Accuracy Avg: [0.]

Testing positions (200 km appart, -60   rotation):
Observer 1: (7142846.101516889, -1129.3850175896164, 7380.564140244884)
Target: (7342846.101516889, -1129.3850175896164, 7380.564140244884)
Distance: 200000 m
Reward: -0.02
Pointing Accuracy: 0.00
Observation Status Matrix: [0]
Cumulative Pointing Accuracy: [[0.]
 [0.]]
Observation Counts: 0
Global Observation Counts: [[0]
 [0]]
Max Pointing Accuracy Avg: [0.]

Testing positions (200 km appart, 20   rotation):
Observer 1: (7142846.101516889, -1129.3850175896164, 7380.564140244884)
Target: (7342846.101516889, -1129.3850175896164, 7380.564140244884)
Distance: 200000 m
Reward: -0.02
Pointing Accuracy: 0.00
Observation Status Matrix: [0]
Cumulative Pointing Accuracy: [[0.]
```

```
 [0.]]
Observation Counts: 0
Global Observation Counts: [[0]
 [0]]
Max Pointing Accuracy Avg: [0.]

Testing positions (200 km appart, -10  rotation):
Observer 1: (7142846.101516889, -1129.3850175896164, 7380.564140244884)
Target: (7342846.101516889, -1129.3850175896164, 7380.564140244884)
observer is observing {'Name': 0} with pointing accuracy 0.00
observer has observed {'Name': 0} with pointing accuracy 0.00
Distance: 200000 m
Reward: [1.00199602]
Pointing Accuracy: 0.00
Observation Status Matrix: [3]
Cumulative Pointing Accuracy: [[0.00199602]
 [0.        ]]
Observation Counts: 1
Global Observation Counts: [[1]
 [0]]
Max Pointing Accuracy Avg: [0.00199602]

.
######### Orbit propagation test
   ##########################################

Initial orbit parameters:
x: 7142846.1015168885 y: -1129.3850175896164 z: 7380.564140244884 vx:
   -7.7969655127956345 vy: -1128.2563498920574 vz: 7373.19229203744

Custom Kleperian orbit propagator results:
'x': 7142834.406071798, 'y': -2258.768804218457, 'z': 14761.12023612507,
   'vx': -15.593922527374481, 'vy': -1128.2545025247753, 'vz':
   7373.19230411004

Poliastro orbit propagator results:
{'x': 7142834.390458391, 'y': -2259.8993152273333, 'z':
   14768.50815865133, 'vx': -15.61734457690526, 'vy':
   -1130.5132678264322, 'vz': 7387.937288559563}

Position error: 7.474 m
Velocity error: 14.917 m/s
.
```

```
Ran 6 tests in 6.293s

OK
```



(a) Charge test

(b) Discharge test

Figure A.6 Charge and discharge battery tests.

A.4 Results

	Adjacency	Battery Available	Computation Time (s)	Contacts (Targets)	Data Transmitted (bits)	Global Communication Counts	Global Observation Counts	Global Observation Status	Maximum Pointing Accuracy	Steps	Reward	Storage Available
Unconstrained Decentralized Coordination												
Monte Carlo	0.91	0.28	100.53	1	17088.99	1.13	0.07	2.77	0.37	4705.84	-1591.69	0.56
PPO	0.84	0.42	625.1	1	349747.1	13.68	0.1	2.57	0.41	8257.39	666.48	0.56
DQN	1	0.28	1367.22	1	370352.65	16.39	0.04	2.38	0.27	16228.56	62.96	0.56
SAC	0.05	0.27	535.49	0.99	0	0	0.05	1.09	0.04	4194.68	-1533.7	0.56
Centralized Coordination												
Monte Carlo	0.14	0.33	137.16	1	4886.65	0.25	0.54	2.92	0.72	3675.7	-33.53	0.56
PPO	0.14	0.37	640.89	1	216930.96	8.37	0.2	2.24	0.52	8128.5	603.63	0.56
DQN	0.14	0.27	10000.1	1	23477.17	0.97	0.06	2.1	0.3	10741.98	88.58	0.57
SAC	0.05	0.27	532.89	0.99	0	0	0.05	1.09	0.04	4166.02	-1522.16	0.56
Constrained Decentralized Communication												
Monte Carlo	0.28	0.27	115.61	1	7878.74	0.74	0.21	2.68	0.59	4635.11	-1746.23	0.57
PPO	0.19	0.35	471.67	1	11249.72	0.77	0.24	1.91	0.56	5627.95	545.88	0.56
DQN	0.29	0.27	980.84	1	10872.72	1.07	0.06	1.59	0.32	10742.31	134.43	0.56
SAC	0.05	0.28	540.91	0.99	0	0	0.05	1.09	0.04	4202.37	-1536.17	0.57

Table A.1 Comparison of DRL Algorithms results under Different Communication Models

A.5 Policies Validation

This annex provides a detailed explanation of the code used to evaluate the decision-making performance of reinforcement learning policies (DQN, SAC, and PPO) in various simulated satellite scenarios. The purpose of this test is to determine how well each policy performs under different resource conditions and predefined decision-making situations. The results of these tests are illustrated in the form of action distribution plots in Figure A.7.

The code is designed to simulate a satellite environment where each satellite is tasked with specific objectives such as communication, observation, or standby operations. The environment simulates the operational dynamics of a satellite constellation using the `FSS_env` environment. The primary objective is to evaluate the performance of three different reinforcement learning (RL) policies—Deep Q-Network (DQN), Soft Actor-Critic (SAC), and Proximal Policy Optimization (PPO)—by observing the decisions they make in various predefined scenarios.

The simulation process starts by creating the environment using the `env_creator` function, which initializes the simulation based on a configuration dictionary (`env_config`). The next step is setting up fixed initial conditions for the satellites using the `generate_fixed_initial_conditions` function. This function defines orbital parameters, energy levels, and storage capacities for each satellite, corresponding to different test scenarios such as being in communication range or observation range.

The `initialize_specific_satellites` function then creates instances of `ObserverSatellite` and `TargetSatellite` using the initial conditions. These instances represent the observers and targets in the simulation, each with specific capabilities related to energy storage and data handling. The `test_policy_once` function is responsible for running a single simulation step using the provided RL policy. It computes the actions selected by the policy, applies them to the environment, and updates the observations, rewards, and states accordingly.

The `run_tests_for_policy` function executes the entire simulation for each RL policy. It iterates over multiple test scenarios with varying resource conditions (combinations of battery and storage levels) and collects the actions chosen by the agents. These scenarios are defined by combinations of low and high battery and storage levels, labeled as `low_low`, `low_high`, `high_low`, and `high_high`.

To analyze the decision-making behavior of each policy, the code tracks the distribution of actions selected in each scenario. The actions are indexed from 0 to 21, corresponding to the possible actions in the environment. The results are then plotted as bar charts, where each scenario and resource condition is represented by a separate subplot. This visual representation allows for a comparison of how frequently each action was chosen by different RL policies under varying conditions.

The results generated by this code offer valuable insights into the decision-making behavior of each RL policy. By analyzing the action distribution, we can see that PPO has a more exploratory behaviour, while DQN is more

conservative. SAC is a total failure regarding the policy, choosing the same action regardless of the state of the environment, but the results are included for comparison as well.

In addition, this code collects data from 100 repetitions of each scenario for each policy, ensuring that the results are statistically significant.

The plotted results are presented here as well:

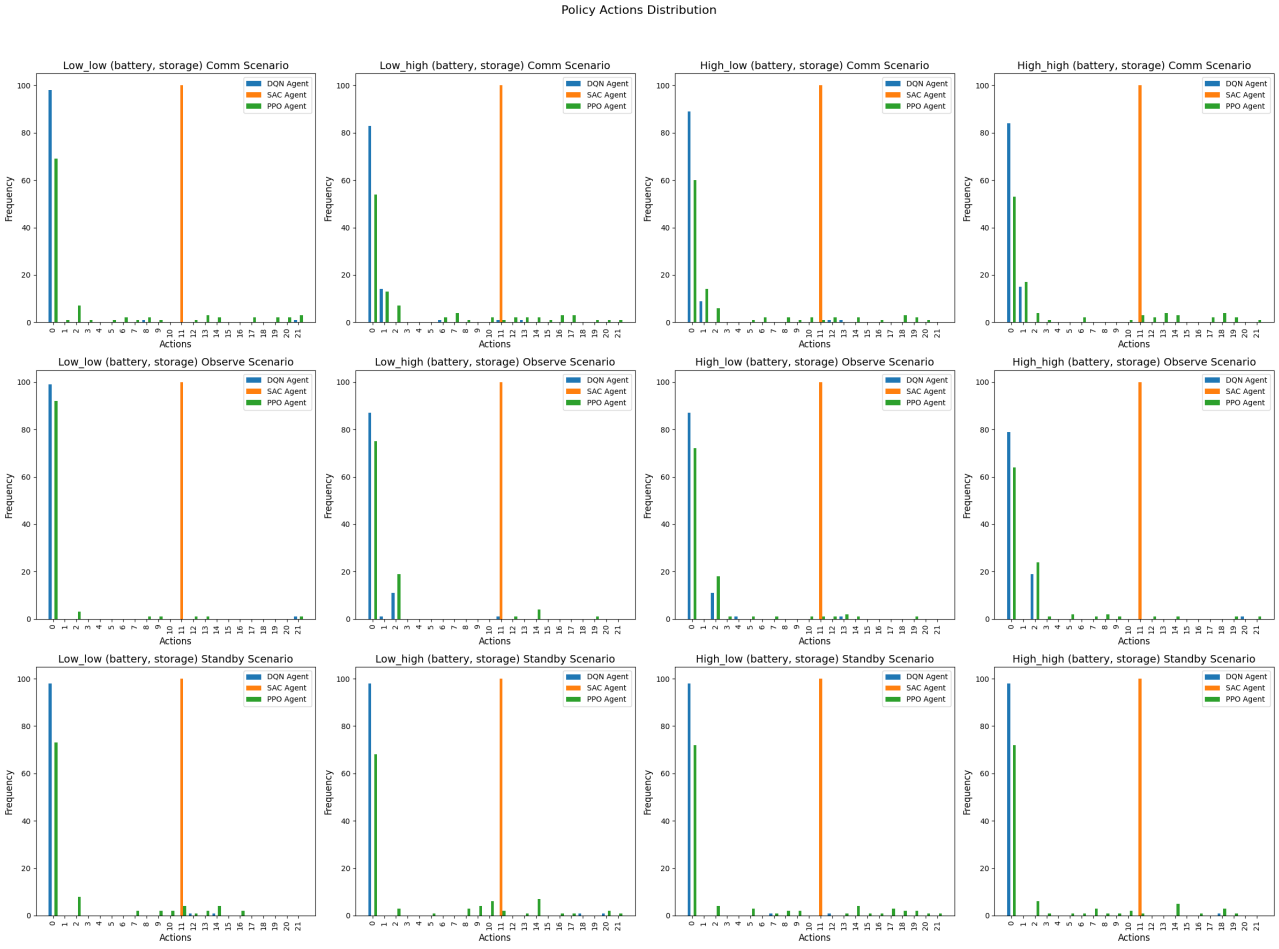


Figure A.7 Policy actions distributions for different test cases.

Bibliography

[1] SpaceNews. "Industry Report: Demand for Satellites is Rising, but Not Skyrocketing". https://spacenews.com/industry-report-demand-for-satellites-is-rising-but-not-skyrocketing/, 2023. Accessed August 2024.

[2] Union of Concerned Scientists. "Satellite Database". https://www.ucsusa.org/resources/satellite-database, 2023. Accessed August 2024.

[3] Alessandro Golkar and Ignasi Lluch i Cruz. "The Federated Satellite Systems Paradigm: Concept and Business Case Evaluation". *Acta Astronautica*, 111:230–248, 2015.

[4] Andrew T. Harris. *"Autonomous Management and Control of Multi-Spacecraft Operations Leveraging Atmospheric Forces"*. PhD thesis, University of Colorado Boulder, 2021.

[5] Adam Paul Herrmann. *"Reinforcement Learning for Spacecraft Planning and Scheduling"*. PhD thesis, University of Colorado Boulder, 2023.

[6] Vincenzo Messina and Alessandro Golkar. "Initial Formulation of a Time Varying Dynamic Graph Decentralized Optimization Framework for Scaled Satellite Network Infrastructure Operations". In *IAC 2023 Congress Proceedings, 74th International Astronautical Congress*, Oct 2023.

[7] Bassel Al Homssi, Kosta Dakic, Ke Wang, Tansu Alpcan, Ben Allen, Russell Boyce, Sithamparanathan Kandeepan, Akram Al-Hourani, and Walid Saad. "Artificial Intelligence Techniques for Next-Generation Massive Satellite Networks". *IEEE Communications Magazine*, 2023.

[8] Yuxi Li. "Deep Reinforcement Learning: An Overview", 2018.

[9] S Knight, Gregg Rabideau, Steve Chien, Barbara Engelhardt, and Rob Sherwood. "Casper: Space Exploration Through Continuous Planning". *IEEE Intelligent Systems*, 16(5):70–75, September 2001.

[10] Gregg Rabideau, R Knight, Steve Chien, A Fukunaga, and A Govindjee. "Iterative Repair Planning for Spacecraft Operations in the ASPEN System". In *International Symposium on Artificial Intelligence Robotics and Automation in Space (ISAIRAS 1999)*, Noordwijk, The Netherlands, June 1999.

[11] Steve Chien, R Sherwood, D Tran, R Castano, B Cichy, A Davies, G Rabideau, N Tang, M Burl, D Mandl, S Frye, J Hengemihle, J Agostino, R Bote, B Trout, S Shulman, S Ungar, J Van Gaasbeck, D Boyer, M Griffin, H Burke, R Greeley, T Doggett, K Williams, V Baker, and J Dohm. "Autonomous Science on the EO-1 Mission". In *International Symposium on Artificial Intelligence, Robotics, and Automation in Space (ISAIRAS 2003)*, Nara, Japan, May 2003.

[12] Steve Chien, J Doubleday, DR Thompson, K Wagstaff, J Bellardo, C Francis, E Baumgarten, A Williams, E Yee, E Stanton, and J Puig-Suari. "Onboard Autonomy on the Intelligent Payload EXperiment (IPEX) CubeSat Mission". *Journal of Aerospace Information Systems (JAIS)*, April 2016.

[13] M Troesch, F Mirza, K Hughes, A Rothstein-Dowden, R Bocchino, A Donner, M Feather, B Smith, L Fesq, B Barker, and B Campuzano. "MEXEC: An Onboard Integrated Planning and Execution Approach for Spacecraft Commanding". In *Workshop on Integrated Execution (IntEx) / Goal Reasoning (GR), International Conference on Automated Planning and Scheduling (ICAPS IntEx/GP 2020)*, October 2020. Also presented at International Symposium on Artificial Intelligence, Robotics, and Automation for Space (i-SAIRAS 2020) and appears as an abstract.

[14] Matthew Dailis, Eric Ferguson, Chris Camargo, and Adrien Maillard. "Aerie: A Modern Multi-Mission Planning, Scheduling, and Sequencing System". In *Proc. of International Workshop on Planning and Scheduling for Space*, June 2023.

[15] NASA Ames Research Center and Massachusetts Institute of Technology. "SPRINT: Scheduling Planning Routing Intersatellite Network Tool". `https://www.nasa.gov/directorates/spacetech/small_spacecraft`, 2020. Accessed July 2024.

[16] Florian Strasser, Vincenzo Messina, Alessandro Golkar, and Valentin Dornauer. "Schedule Optimization for a Heterogeneous Earth Observation Satellite Constellation". In *74th International Astronautical Congress (IAC)*, 2023.

[17] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. "Deep Reinforcement Learning: A Brief Survey". *IEEE Signal Processing Magazine*, 34(6):26–38, November 2017.

[18] Newsha Emaminejad and Reza Akhavian. "Trustworthy AI and Robotics: Implications for the AEC Industry". *Automation in Construction*, 139:104298, 2022. Accessed July 2024.

[19] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Arthur Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. "Mastering the Game of Go Without Human Knowledge". *Nature*, 550:354–359, October 2017.

[20] Adam Herrmann and Hanspeter Schaub. "Reinforcement Learning for the Agile Earth-Observing Satellite Scheduling Problem". *IEEE Transactions on Aerospace and Electronic Systems*, 59(5):5235–5247, 2023.

[21] Andrew Harris and Kedar Naik. "Autonomous Command and Control for Earth-Observing Satellites using Deep Reinforcement Learning". In *2021 IEEE Aerospace Conference*, pages 1–12, 2021.

[22] Mahya Ramezani, M A Amiri Atashgah, M Amin Alandihallaj, and Andreas M Hein. "Reinforcement Learning for Planning and Task Coordination in a Swarm of CubeSats: Overcoming Processor Limitation Challenges", 2023.

[23] Bassel Al Homssi, Akram Al-Hourani, Ke Wang, Phillip Conder, Sithamparanathan Kandeepan, Jinho Choi, Ben Allen, and Ben Moores. "Next Generation Mega Satellite Networks for Access Equality: Opportunities, Challenges, and Performance". *IEEE Communications Magazine*, 60(4):18–24, 2022.

[24] Maurice Martin and Michael Stallard. *"Distributed Satellite Missions and Technologies - The TechSat 21 Program"*. Accessed November 2023.

[25] CubeSat. "CubeSat Information". `https://www.cubesat.org/cubesatinfo`, 2024. Accessed August 2024.

[26] Simone Scrocciolani, Vincenzo Messina, Ramón María García Alarcia, Jaspar Sindermann, and Alessandro Golkar. "Advancing Satellite Network Performance: Network Analysis for Federated Satellite Systems". *IEEE Access*, 12:45616–45630, 2024.

[27] S. Alluru and J. McNair. "An Optical Payload for Cube-Sats". In *SmallSat 2010. Connecting the Dots Conference Proceedings*, pages 1–13, 2010. Accessed August 2024.

[28] Leonhard Kessler. "Optimizing Digital Twin Methodology for CubeSats: Quantitative Analysis for Design, Performance, and Implementation Strategies". Master's thesis, Technical University of Munich, Munich, Germany, July 2024. Chair of Spacecraft Systems, TUM School of Engineering and Design.

[29] Justin Terry. "Multi-Agent Deep Reinforcement Learning in 15 Lines of Code Using PettingZoo". `https://towardsdatascience.com/multi-agent-deep-reinforcement-learning-in-15-lines-of-code-using-pettingzoo-e0b963c0820b`, 2021. Accessed August 2024.

[30] Matthias Niessner, Laura Leal-Taixé, and Angela Dai. "Introduction to Deep Learning (IN2346)". Lecture notes, Technische Universität München, 2024. Compiled on August 27, 2024.

[31] Mark Stephenson and Hanspeter Schaub. "Optimal Target Sequencing In The Agile Earth-Observing Satellite Scheduling Problem Using Learned Dynamics". In *AAS/AIAA Astrodynamics Specialist Conference*, Big Sky, Montana, August 2023. Accessed January 2024.

[32] Martin L. Puterman. *"Markov Decision Processes: Discrete Stochastic Dynamic Programming"*. John Wiley Sons, Inc., USA, 1st edition, 1994.

[33] Duncan Eddy and Mykel Kochenderfer. "Markov Decision Processes For Multi-Objective Satellite Task Planning". In *2020 IEEE Aerospace Conference*, pages 1–12, 2020.

[34] Frans Oliehoek and Christopher Amato. "A Concise Introduction to Decentralized POMDPs". 01 2016.

[35] M. Pumperla, E. Oakes, and R. Liaw. *"Learning Ray: Flexible Distributed Python for Machine Learning"*. O'Reilly Media, 2023.

[36] Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. "RLlib: Abstractions for Distributed Reinforcement Learning", 2018.

[37] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *"Deep Learning"*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[38] Rafael Stekolshchik. "Deep Reinforcement Learning Algorithms". `https://github.com/Rafael1s/Deep-Reinforcement-Learning-Algorithms`, 2024. Accessed August 2024.

[39] Joshua Achiam. "Spinning Up in Deep Reinforcement Learning". 2018.

[40] Vaishak V. Kumar. "Soft Actor-Critic Demystified". `https://towardsdatascience.com/soft-actor-critic-demystified-b8427df61665`, 2024. Accessed August 2024.

[41] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal Policy Optimization Algorithms", 2017.

[42] Kaixin Cui, Jiliang Song, Lei Zhang, Ying Tao, Wei Liu, and Dawei Shi. "Event-Triggered Deep Reinforcement Learning for Dynamic Task Scheduling in Multisatellite Resource Allocation". *IEEE Transactions on Aerospace and Electronic Systems*, 59(4):3766–3777, 2023.

[43] Li Dalin, Wang Haijiao, Yang Zhen, Gu Yanfeng, and Shen Shi. "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.

[44] Craig Boutilier. "Sequential Optimality and Coordination in Multiagent Systems". In *Proceedings of the Sixteenth International Joint Conferences on Artificial Intelligence (IJCAI-99)*, 08 1999.

[45] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. "Gymnasium", March 2023.

[46] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. "OpenAI Gym", 2016.

[47] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[48] Leibniz Supercomputing Centre (LRZ). "AI Systems at LRZ". https://doku.lrz.de/lrz-ai-systems-11484278.html, 2024. Accessed August 2024.

[49] Juan Luis Cano Rodríguez, Jorge Martinez, et al. "poliastro: poliastro 0.17.0", 2023.

[50] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. "A System for Massively Parallel Hyperparameter Tuning", 2020.

[51] NVIDIA Developer Documentation. "Jetson AGX Orin Developer Guide: Power Modes and Profiles". https://docs.nvidia.com/jetson/archives/r34.1/DeveloperGuide/text/SO/JetsonAgxOrin.html#power-modes-profiles. Accessed August 2024.

[52] Hripsime Matevosyan, Christianna E. Taylor, and Alessandro Golkar. *"Evaluating Virtual Satellite Mission Opportunities"*. Accessed November 2023.

[53] Maurice Martin, Pete Klupar, Steve Kilberg, and James Winter. "Techsat 21 and Revolutionizing Space Missions Using Microsatellites". 2001.

[54] Maurice Martin and Michael Stallard. "Distributed Satellite Missions and Technologies - The TechSat 21 Program". In *Space Technology Conference and Exposition*, page 4479, 1999.

[55] Thuy Ngoc Nguyen, Chase McDonald, and Cleotilde Gonzalez. "Credit Assignment: Challenges and Opportunities in Developing Human-like AI Agents", 2023.

[56] Fares Fourati and Mohamed-Slim Alouini. "Artificial Intelligence for Satellite Communication: A Review". *Intelligent and Converged Networks*, 2(3):213–243, 2021.

[57] Osvaldo Simeone. "A Very Brief Introduction to Machine Learning With Applications to Communication Systems". *IEEE Transactions on Cognitive Communications and Networking*, 4(4):648–664, 2018.

[58] Xinwei Wang, Guohua Wu, Lining Xing, and Witold Pedrycz. "Agile Earth Observation Satellite Scheduling Over 20 Years: Formulations, Methods, and Future Directions". volume 15, pages 3881–3892, 2021.

[59] David L Poole and Alan K Mackworth. *"Artificial Intelligence: Foundations of Computational Agents"*. Cambridge University Press, 2010.

[60] Hans Petter Langtangen. "Combining Python with Fortran, C, and C++". *Python Scripting for Computational Science*, pages 169–204, 2004.

[61] Yannick Martel, Arne Roßmann, Eldar Sultanow, Oliver Weiß, Matthias Wissel, Frank Pelzel, and Matthias Seßler. "Software Architecture Best Practices for Enterprise Artificial Intelligence". INFORMATIK 2020, 2021.

[62] Terry Fong. "Autonomous Systems: NASA Capability Overview". Technical report, 2018.

[63] R Sherwood, Steve Chien, R Castano, and G Rabideau. "Autonomous Operations Through Onboard Artificial Intelligence". In *International Conference on Space Operations (SpaceOps 2002)*, Houston, TX, October 2002.