FINAL DEGREE THESIS

AEROSPACE TECHNOLOGY ENGINEERING & ENGINEERING PHYSICS

Earth Observation Satellite Planning Problem with Deep Reinforcement Learning and Transformers

UNIVERSITAT POLITÈCNICA DE CATALUNYA

CFIS, ESEIAAT, ETSETB

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH
Centre de Formació Interdisciplinària Superior

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH
Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

telecos BCN

Author:
Gerard Morales Riera

Professors:
Daniel Selva Valero
Alex Ferrer Ferre

May 15, 2025

Abstract in Catalan

El canvi climàtic és cada vegada més present. Les causes són molt clares i les conseqüències molt adverses, les quals prenen la forma de desastres naturals lluny de la capacitat humana per enfrontar-los. Davant aquest perill, sorgeix la necessitat de percebre'ls visualment per preveure els seus riscos i disminuir incidents. Aquí entren en escena les constel·lacions de satèl·lits posades en òrbita amb l'objectiu d'observar la terra. Davant el desgast econòmic que genera una posada en òrbita, juntament amb les limitacions dels sensors òptics a disposició, apareix un problema a optimitzar basat en com orientar aquests satèl·lits per tal d'adquirir les millors imatges possibles. Aquesta problema s'ha formulat de diverses maneres, començant per mètodes de programació dinàmica, que condueixen a solucions properes a òptimes però són molt costoses de generar. L'aproximació feta en aquesta tesi ressalta la possibilitat de resoldre aquest problema amb aprenentatge automàtic i, en concret, aprenentatge per reforç.

Abstract in Spanish

El cambio climático está cada vez más presente. Las causas son muy claras y las consecuencias muy adversas, que toman la forma de desastres naturales lejos de la capacidad humana para enfrentarlos. Ante este peligro, surge la necesidad de percibirlos visualmente para prever sus riesgos y disminuir incidentes. Aquí entran en escena las constelaciones de satélites puestas en órbita con el objetivo de observar la tierra. Ante el desgaste económico que genera una puesta en órbita, junto con las limitaciones de los sensores ópticos a disposición, aparece un problema a optimizar basado en cómo orientar estos satélites para adquirir las mejores imágenes posibles. Este problema se ha formulado de diversas formas, empezando por métodos de programación dinámica, que conducen a soluciones cercanas a óptimas pero son muy costosas de generar. La aproximación hecha en esta tesis resalta la posibilidad de resolver este problema con aprendizaje automático y, en concreto, aprendizaje por refuerzo.

Abstract in English

Climate change is becoming increasingly prevalent. The causes are very clear, and the consequences are very adverse, taking the form of natural disasters beyond human capacity to cope with. Faced with this threat, there is a need to visually perceive them in order to predict their risks and reduce incidents. This is where satellite constellations launched into orbit for Earth observation come into play. Given the costly nature of launching into orbit, coupled with the limitations of available optical sensors, an optimization problem arises based on how to orient these satellites to acquire the best possible images. This problem has been formulated in various ways, starting with dynamic programming methods, which lead to near-optimal solutions but are very costly to generate. The approach developed in this thesis highlights the possibility of solving this problem with machine learning, and specifically, reinforcement learning.

List of Keywords in Catalan

Satèl·lit
Observació de la Terra
Maneig orbital
Camp de visió
Camp d'observació
Planificació
Aprenentatge automàtic
Aprenentatge per reforç
Intel·ligència artificial
Processos de decisió de Markov
Programació dinàmica
Transformador
Sensor òptic
Red neuronal profunda
Programació lineal
Sostenibilitat
Impacte ambiental
Impacte econòmic
Impacte social
Problema NP-complet
Simulació
Algorisme
Constel·lació de satèl·lits
Processament de senyals
Dades d'observació
Aprenentatge profund
Codificació posicional
Optimització
Política estocàstica
Acció contínua
Espai d'estats
Espai d'accions
Funció de recompensa
Processament seqüencial
Atenció
Inferència
Predicció
Pèrdua
Generalització
Eficiència computacional

List of Keywords in Spanish

Satélite
Observación de la Tierra
Maniobra orbital
Campo de visión
Campo de observación
Planificación
Aprendizaje automático
Aprendizaje por refuerzo
Inteligencia artificial
Procesos de decisión de Markov
Programación dinámica
Transformador
Sensor óptico
Red neuronal profunda
Programación lineal
Sostenibilidad
Impacto ambiental
Impacto económico
Impacto social
Problema NP-completo
Simulación
Algoritmo
Constelación de satélites
Procesamiento de señales
Datos de observación
Aprendizaje profundo
Codificación posicional
Optimización
Política estocástica
Acción continua
Espacio de estados
Espacio de acciones
Función de recompensa
Procesamiento secuencial
Atención
Inferencia
Predicción
Pérdida
Generalización
Eficiencia computacional

List of Keywords in English

Satellite
Earth observation
Orbital maneuvering
Field of view
Field of regard
Planning
Machine learning
Reinforcement learning
Artificial intelligence
Markov decision processes
Dynamic programming
Transformer
Optical sensor
Deep neural network
Linear programming
Sustainability
Environmental impact
Economic impact
Social impact
NP-hard problem
Simulation
Algorithm
Satellite constellation
Signal processing
Observation data
Deep learning
Positional encoding
Optimization
Stochastic policy
Continuous action
State space
Action space
Reward function
Sequential processing
Attention
Inference
Prediction
Loss
Generalization
Computational efficiency

List of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Earth observation missions are served for diverse scientific purposes. With climate change being a reality, natural disasters is one of the main reasons why these spatial sensor are of great use. In this matter, Earth Observation Satellites help to reduce the severity of incidents by the use of weather data they collect.

There have been plenty of Earth Science missions taken place by many space agencies like NASA, JAXA, ESA, etc. Form NASA reports on their Earth Science division missions we can extract some of them and their respective purposes based on the instrumentation they contain and their focus on Earth's scientific interests.

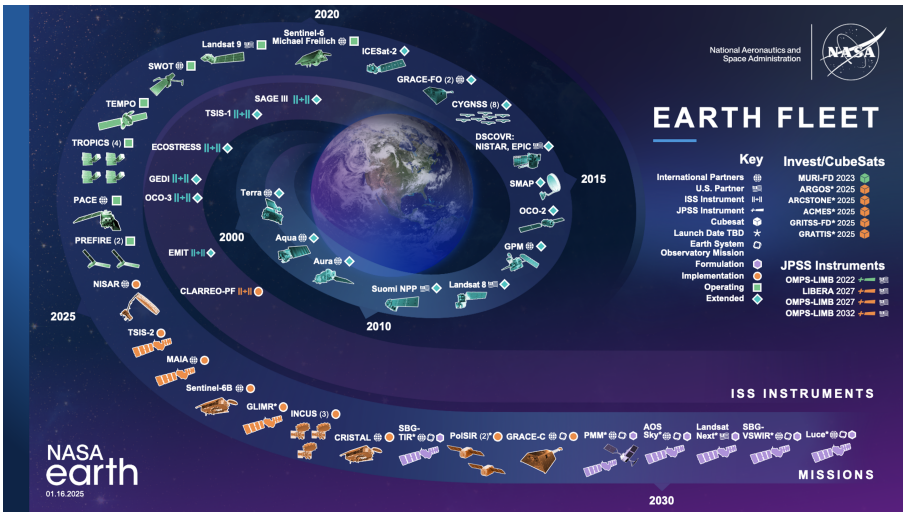

Figure 1.1: NASA Fleet for Earth Science missions. [1]

In Figure 1.1, the timeline for Earth Science missions can be seen from the early 2000s to some of the incoming 2030s-projected ones. The main purpose of each of these missions can be vary. Four of the most present missions objectives are:

- Water and Energy Cycle monitoring missions.
- Tropical Cyclone observing missions.

- Sea Surface Height measurement missions.
- Ocean Color detection missions.

An overview of these four will be performed below.

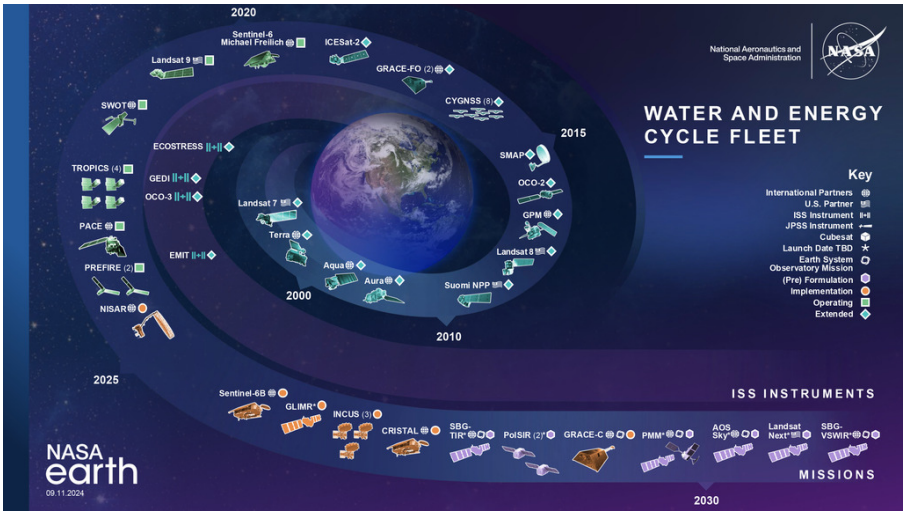

Figure 1.2: NASA Fleet for Water and Energy Cycle monitoring missions. [1]

What Figure 1.2 shows at first sight is how many of the set of Earth Science missions are dedicated to the water and energy cycle. Missions like Terra and Aqua led the way on the Earth Science missions with their wide range of capabilities. Both of them were international collaborations mainly with Japan's space agency (JAXA) and Canada's (CSA). Terra was dedicated to collect information form Earth's atmosphere, snow, oceanic water, energy cycle, etc. Aqua also collects a huge amount of data centered on Earth's water.[2][3]

Other important missions where SMAP, for Soil Moisture, launched in 2015 and CYGNSS, which is a constellation of eight satellites dedicated to cyclone detection, launched in 2016.

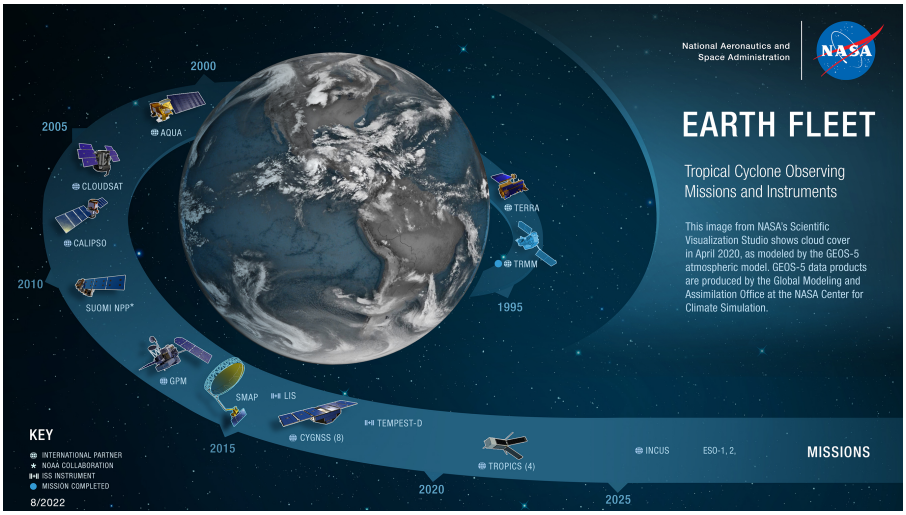

Figure 1.3: NASA Fleet for Tropical Cyclone observing missions. [1]

The missions SMAP and CYGNSS previously mentioned also belong to the category of tropical cyclone observing missions, as seen in Figure 1.3.

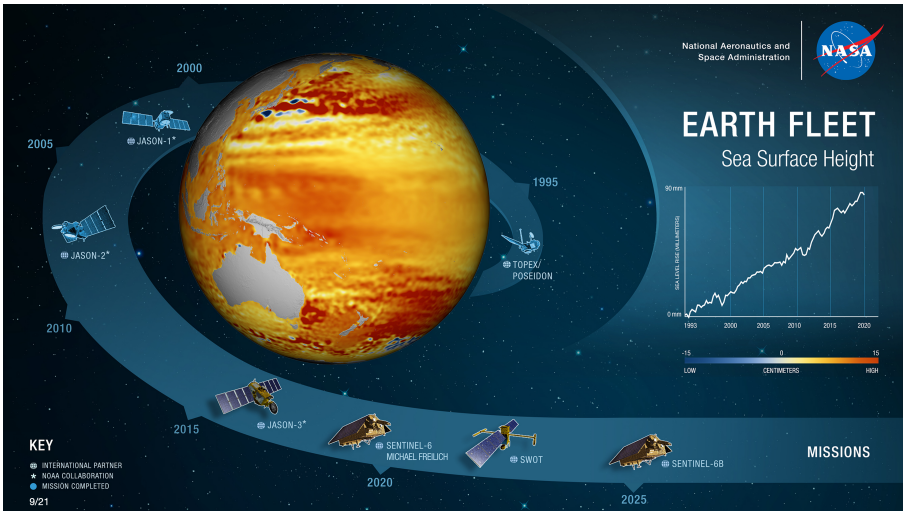

Figure 1.4: NASA Fleet for Sea Surface Height measurement missions. [1]

Back in 1992, the TOPEX/Poseidon mission was the first high-precision ocean altimeter mission. It led the way for the data acquisition of sea level height within less than 5cm of accuracy. This mission helped predict El Niño in 1997-1998 and acquired a lot of relevant data for global climate studies. This mission was later followed by Jason-1 and Jason-2. All of these were initially tasked to last a few years but each of them overcame the decade-long operational period.[4]

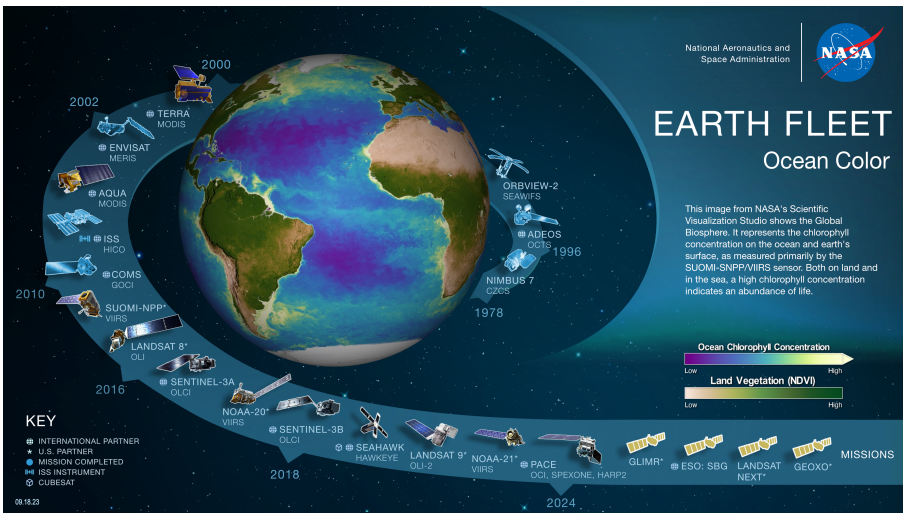

Figure 1.5: NASA Fleet for Ocean Color detection missions. [1]

Ocean color has been tracked by a numerous amount of satellite missions. Some of them, like Terra or Aqua, have already been mentioned. However, there have been mission since back in 1978 with Nimbus 7. More modern ones are comprised by the Sentinel-3A and Sentinel-3B, Seahawk or the International Space Station.

As a summary of the previous figures and comments, here lie two tables containing the main missions mentioned and a few of their long-lasting operational lives and countless features and capabilities.

Table 1.1: Featured Earth Science missions overview. [1]

Mission Name	Date and Status	Contributors
SWOT (Surface Water and Ocean Topography)	December 16, 2022; Operating: 3 years, possibly extended	NASA (US), CNES (France), CSA (Canada), UKSA (UK)
GPM (Global Precipitation Measurement Core Observatory)	February 27, 2014; Extended: 3 years initially, but still operating 10 years later	NASA (US) and JAXA (Japan)
SMAP (Soil Moisture Active Passive)	January 31, 2015; Extended: 3 years initially, but still operating with reduced capability	NASA (US)
CYGNSS (Cyclone Global Navigation Satellite System)	December 15, 2016; Extended: 2 years initially, but still operating	University of Michigan (US), NASA (US)
TROPICS (Time-Resolved Observations of Precipitation Structure and Storm Intensity with a Constellation of Smallsats)	May and June 2023; Operating: 2 years at least	NASA (US)
CALIPSO (Cloud-Aerosol Lidar and Infrared Pathfinder Satellite Observations)	April 28, 2006; Completed: 3 years initially, but extended for 17 years until 2023	NASA (US), CNES (France)
SUOMI NPP (National Polar-orbiting Partnership)	October 28, 2011; Extended: 5 years initially, but still operating 13 years after	NASA (US), NOAA (US)
TRMM (Tropical Rainfall Measuring Mission)	November 27, 1997; Completed: 3 years initially, deorbited in 2015	NASA (US), JAXA (Japan)
TOPEX/Poseidon	August 10, 1992; Completed: 3-5 years initially, but extended until 2006	NASA (US), CNES (France)
Jason-1	December 7, 2001; Completed: 3 years initially, but extended for more than 11	NASA (US), CNES (France)
Jason-2 (OSTM – Ocean Surface Topography Mission)	June 20, 2008; Completed: 3 years initially, but extended for more than 11	NASA (US), CNES (France), NOAA (US), EUMETSAT (EU)
ORBVIEW-2 / SeaWiFS (Sea-Viewing Wide Field-of-View Sensor)	August 1, 1997; Completed: 5 years initially, but extended until 2010	Orbital Science Corporation (US), NASA (US)
ADEOS-I / OCTS (Ocean Color and Temperature Scanner)	August 1996; Failed: one year later due to power loss	NASA (US), JAXA (Japan)
Nimbus-7 / CZCS (Coastal Zone Color Scanner)	October 24, 1978; Completed: lasted 16 years	NASA (US)
SeaHawk Cubesat	December 3, 2018; Completed: decommissioned in 2023	University of North Carolina Wilmington (UNCW, US) and NASA (US)

Table 1.2: Featured Earth Science missions technical information. [1]

Mission Name	Purpose	Instruments
SWOT (Surface Water and Ocean Topography)	Great resolution to map rivers, lakes... Ocean circulation and water storage flow	Ka-band Radar Interferometer (KaRIn), NADIR altimeter and radiometer
GPM (Global Precipitation Measurement Core Observatory)	Provides precipitation data and is key for weather forecasting and climate monitoring	Dual-frequency Precipitation Radar (DPR), GMI (GPM Microwave Imager)
SMAP (Soil Moisture Active Passive)	Measures soil moisture globally and is key for drought monitoring, weather forecasting...	L-band Radiometer, L-band Radar
CYGNSS (Cyclone Global Navigation Satellite System)	Observe winds from inside cyclones and detect rapid precipitation intensifications	GNSS-R (Global Navigation Satellite System Reflectometry), DDMI (Delay Doppler Mapping Instrument)
TROPICS (Time-Resolved Observations of Precipitation Structure and Storm Intensity with a Constellation of Smallsats)	High-frequency observation of tropical cyclones	Microwave radiometer
CALIPSO (Cloud-Aerosol Lidar and Infrared Pathfinder Satellite Observations)	Key to understand cloud formation and storm layering	CALIOP (Cloud-Aerosol Lidar with Orthogonal Polarization), IIR (Infrared Imaging Radiometer), WFC (Wide Field Camera)
SUOMI NPP (National Polar-orbiting Partnership)	Hurricane tracking and humidity profiling	VIIRS (Visible Infrared Imaging Radiometer Suite), CrIS, ATMS, OMPS
TRMM (Tropical Rainfall Measuring Mission)	Provided first 3D maps of tropical rainfall	PR (Precipitation Radar), TMI (TRMM Microwave Imager), LIS (Lightning Imaging Sensor)
TOPEX/Poseidon	First high-precision ocean altimetry mission	Poseidon Radar Altimeter, TOPEX Radar Altimeter
Jason-1	Measured global sea level rise, monitored ocean eddies	Poseidon-2 Altimeter, Microwave Radiometer, DORIS, GPS
Jason-2 (OSTM – Ocean Surface Topography Mission)	Data record of global sea level rise and ocean circulation	Poseidon-3 Radar Altimeter, Advanced Microwave Radiometer, DORIS, GPSP
ORBVIEW-2 / SeaWiFS (Sea-Viewing Wide Field-of-View Sensor)	First long-term, dedicated global ocean color mission, mapped chlorophyll-a concentration	SeaWiFS
ADEOS-I / OCTS (Ocean Color and Temperature Scanner)	First Japanese Earth observation satellite focused on marine biology and oceanography	OCTS
Nimbus-7 / CZCS (Coastal Zone Color Scanner)	Measured chlorophyll, suspended sediment, and gelbstoff (dissolved organic matter) in coastal waters	CZCS
SeaHawk	Provide high-resolution ocean color data for coastal ecosystems	Hawkeye

1.1 Natural Disasters

All the missions mentioned in Table 1.1 and Table 1.2 reflect very specific purposes towards Earth Science data collection. However, satellite imagery by itself is also an enormous interest to thrive for. For diverse reasons, from mapping to natural disaster protection, in this section there will be a focus on the latter one.

There is a long list of possible natural hazards that are of great interest to observe and analyse their behaviour, as well as predict it. Now, with climate change, there is a particular need to track them, given their increased danger and dimensions. Each of disasters has a different duration and appearance conditions. The table below is a summary of the most important ones and their characteristics.

Table 1.3: Main natural hazards on planet Earth.

Natural Hazard	**Duration**	**Typical Locations**
Earthquakes	Seconds to minutes	Pacific Ring of Fire (Japan, California, Indonesia, etc.), Himalayan region
Tropical Cyclones	Days to 1–2 weeks	Atlantic Ocean, Western Pacific, Indian Ocean
Tsunamis	Minutes to hours	Japan, Indonesia, Chile, Alaska, Pacific Islands
Volcanic Eruptions	Minutes to decades	Ring of Fire, Iceland, Italy, Hawaii
Floods	Hours to weeks	Bangladesh, India, China, U.S. Midwest, Amazon Basin
Droughts	Months to years	Sub-Saharan Africa, Australia, Western U.S., Brazil, Middle East
Wildfires	Days to months	California, Australia, Mediterranean, Canada, Siberia
Landslides	Seconds to hours	Nepal, Colombia, Himalayas, Andes, Pacific Northwest (U.S.)
Blizzards / Snowstorms	Hours to several days	Northern U.S., Canada, Russia, Northern Europe, Himalayas, Japan
Hailstorms	Minutes to a few hours	U.S. Great Plains, Northern India, China, Argentina, Alps
Tornados	Seconds to minutes	U.S. Central Plains, Australia, Central Europe, South Africa
Dust Storms	Hours to days	Sahara Desert, Middle East, Central Asia, Southwestern U.S.
Avalanches	Seconds to a few minutes	Alps, Rockies, Himalayas, Andes
Algal blooms[1]	Days to months	Gulf of Mexico, Great Lakes, Baltic Sea

Notice from Table 1.3 that some of the hazards are orders of magnitude longer in duration compared to some others. That makes a very high difference on the capacity of Earth observing satellites to detect them or not. That makes not only a great difference on the purpose of Earth Science missions nowadays but also creates the main challenge ahead. This main challenge is get the satellites' maneuvering and decision-making be quick enough to not miss any of the items in the table. This is the case of the tornadoes, for instance, since they last seconds to minutes and can only be predicted with approximately eight minutes before happening.

[1] Algal blooms are biological rather than geological. Sometimes, they are not harmful. However, they are considered here given that some of these events release toxins that produce mass killings of fish species, making them worthy of study.

1.2 Limitations in Earth Observation

As seen in Table 1.2, all missions count on high-quality, state of the art instruments to detect their scientific purposes. However, even though how advanced this set of machinery is, there exist some limitations to their capabilities. Lots of them could be listed, but the most convenient to this thesis is the limitation in resolution for cameras and other sensors.

Hypothetically speaking, a possession of extremely good resolution would lead to the capability to capture data of the entire Field of Regard (FoR) of the satellite. The FoR is the entire surface of the Earth that the satellite can perceive, only limited by Earth's curvature and those points physically impossible to directly see. Also, on some cases, the FoR term can be applied to the sensor if this one's is smaller that the satellite's due to mobility constraints. The FoR for most of the satellites is a considerable fraction of the planet's surface. In this order of magnitude of area to perceive, the resolution of the instruments should be unthinkable with nowadays technology.

This lack of capability leads to the entire problem this thesis is about. Given that the FoR is not fully perceivable at once, one must focus on the Field of View (FoV). The FoV is the surface of the Earth that the satellite's sensor is actually collecting data from. The Earth's projection of the FoV (usually 100% of it given that the case of study is Earth-focused) must be contained within the FoR, and generally consists on a small fraction of it. The FoV is the area the sensor is actually good enough to perceive with certain, consistent precision and resolution.

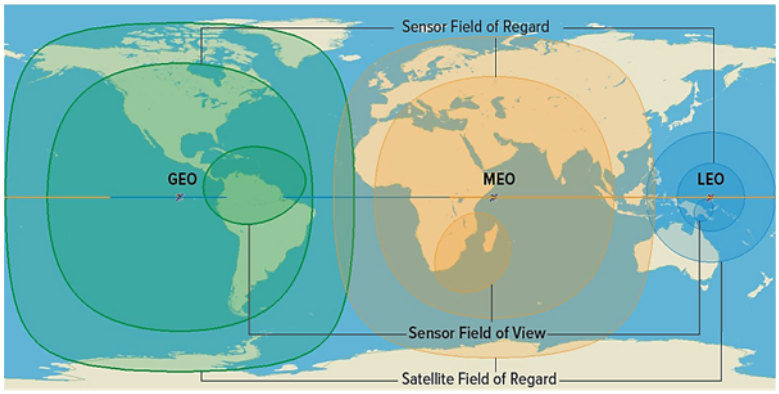

Figure 1.6: Satellite FoR, Sensor FoR and FoV comparison.

Once seen that, the issue is held on the fact that the FoV is too small to collect data from all the events that are happening on Earth. Therefore, a planning/scheduling problem arises from the question: where should the FoV be focused along time?

The answer to this is clearly non-trivial, given all the variables to consider, and the sudden appearance of new interest observation events. Some missions have easier predictability and less need for planning. However, some others need it more.

Chapter 2

Literature Review

The first approaches of the satellite maneuvering for Earth Observing were mostly based on preplanning. That is, the satellite would move its sensors within an established pattern that was to be preset by the controller. This would mean to select a certain number of targets or key areas and perform an almost-manually set FoV path that would cover the zones of interest. This is done through a simple scheduler that selects the tasks to be done (which mean coordinates to see) and then another software program predicts the quickest or more efficient movement to fulfill these tasks.

For several purposes, this way to go is more than enough. There are missions which only need to do an exhaustive scanning of Earth's surface, which reduces the problem to the simplest extent. However, preplanning is inherently sub-optimal for many cases, given the lack of information used in advanced stages of performing the maneuver. Since Earth's change is not considered, replanning would take place.

Replanning is basically allowing an input of new, real-time (or close to) upcoming data into your decision-making system. This makes the problem more complex to solve, and several approaches have been taken. The main limitations of such methods are essentially the complexity of the task and, as it will be mentioned, the computational runtime.

The initial ways to implement replanning included, for instance, rule-based algorithms that evolved into heuristic-enriched constrained optimizations. These did not require a lot of searching or learning capability by the agent[1], which inherently did not give it full autonomy.

2.1 ASPEN

The first case to mention is ASPEN, a JPL-developed (Jet Propulsion Laboratory, NASA in California) framework to serve automated scheduling operations. Although not only made for Earth Observing maneuvering operations, it served this purpose in the New Millennium Earth Observing satellite. The framework was based on using a Temporal Constraint Network (TCN) to search for near-optimal state-space solutions. The TCN

[1] An agent is considered to be the lowest unit of an autonomous (or close to) decision-maker being, which takes decisions based on a certain policy. In this case, a satellite that moves by itself taking its own decisions is an agent.

represent temporal constraints within a graph. Out of this, several scheduling algorithms could be supported, such as repair-based algorithms. [5] [6]

ASPEN was a primitive form of Artificial Intelligence (AI), which a while after became the state of the art for the planning/scheduling problem. [7]

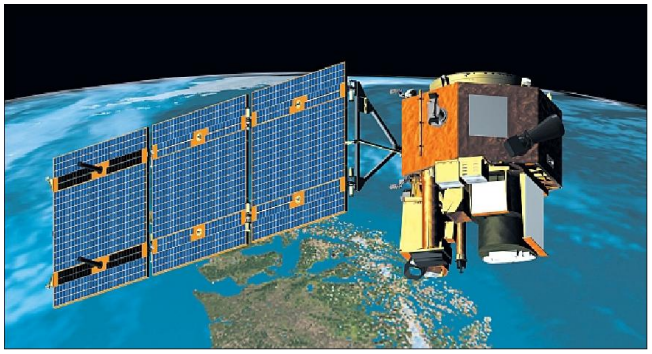

Figure 2.1: Computer-generated picture of the New Millenium (EO-1). [8]

2.2 Dynamic Programming

A formulation that has been widely used for this case of study is Dynamic Programming (DP). DP is a mathematical approach where a problem is broken down into several subproblems that overlap between each other. This method is used to save unnecessary computations. In the task planning domain, the set of tasks the agent should perform are placed in a graph that will compute its optimal ordered tasks allocation. This method, given full knowledge of the task possibilities, can reach solutions very close to optimality. However, the issue with this formulation is how time-consuming it is for practical cases. Reaching optimality is a possibility, but the execution time of the algorithm, which is particularly crucial for Low Earth Orbit (LEO) satellites, makes it too impractical to be used for more complex sets of targets. [9] [10]

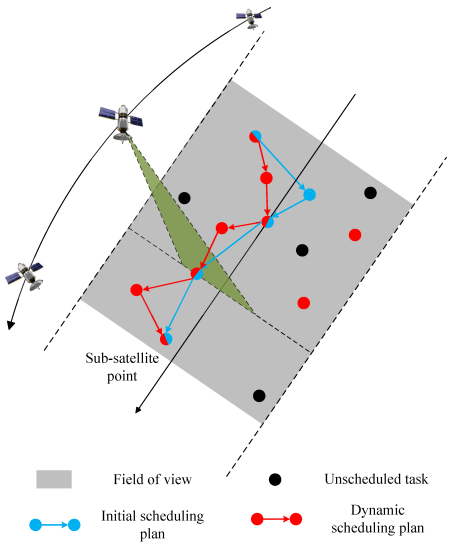

Figure 2.2: Dynamic programming used in Earth observation replanning. [9]

2.3 Linear Programming

Another problem formulation that took place before modern AI-based algorithms Mixed-Integer Linear Programming (MILP). Compared to DP, MILP one is not as capable to reach near-optimal solutions but has a better execution runtime. In 2018, Yuchen She et al. compared an MILP problem formulation for the observing satellite with a Genetic Algorithm solving a similar formulation. They realized the GA would outperform the MILP slightly, but the runtime in the GA was significantly longer. [11]

More recently, Qingyu Qu et al. built an Imitation Learning algorithm which used MILP solutions as the model's benchmark to learn patterns and relationships that featured a good task allocation. This was performed through an iterative process where a Graph Convolutional Neural Network (GCNN) that learning to perform the Branch and Bound (B&B) task of a MILP replanner. Certain strategies were generated by the model through environment interaction (after being fine-tuned) and then stored to be fed to the model back again later. They realized this new approach slightly outperformed heuristic and other algorithms in terms of runtime and performance. [12]

2.4 Reinforcement Learning

Herrmann et al. explored a Reinforcement Learning (RL) formulation to tackle the problem. They used a Basilisk (BSK) Simulator (astrodynamics simulation framework) to run as the environment where the RL agent would be trained. The training takes the form of a Monte Carlo Tree Search (MCTS) algorithm that runs through an iterative process of four steps: select, expand, rollout, backup.

These steps expect to be looking for the maximization of return for the agent, i.e., most incentivized actions to take in their own space definition. Specifically, the *Select* step tries to maximize the expected cumulative sum of rewards. Their MCTS computation is then fed to a feedforward that later runs through the environment. This way, in a repetitive experiment, the network can be trained and later be validated in their environment. Training the network allows to get similar or even better results than the MCTS itself, and all of this in a fraction of its runtime. [13] [14]

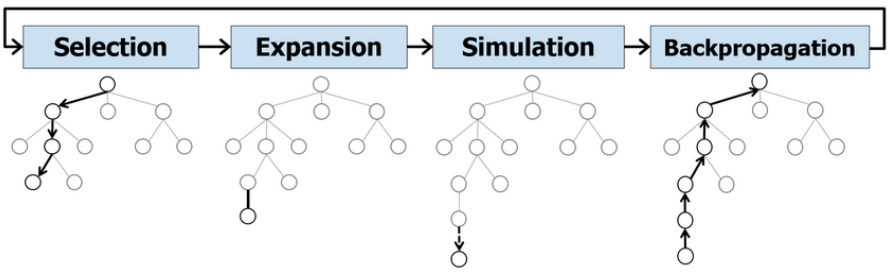

Figure 2.3: Four steps of the MCTS iterative process.

Apart from Hermann, Chen et al. also introduced in 2019 another RL-based planning algorithm with the use of Deep RL to get maximized learning capabilities for the satellite agents. Here they formulate it as a Markov

Decision Process (MDP), where the problem description is as usual within the literature, but their decision-maker is a Deep Neural Network (DNN) where the Attention Mechanism is leveraged. [15]

This format is very curious, since it does not precede the transformer architecture presented by Vaswani et al. in 2017, but maybe it had not yet shown unveiled potential by the time Chen et al. progressed in this problem. Hence, their architecture is based on a widely used encoder-decoder structure with an attention mechanism attached. The encoder is a Convolutional Neural Network (CNN) to encode data in a vectorized latent space, which is then decoded by a Recurrent Neural Network (RNN) with a Gated Recurrent Unit (GRU), given the need to use an architecture with memory to leverage the attention block. Once this, the training algorithm they use is an Actor-Critic. The authors state that their results are, to the best of their knowledge the best ever seen for a multi-dimensional-constrained formulation like theirs. [15] [16]

2.5 Consensus Algorithms

Aguilar et al. explored a decentralized approach to the task allocation scheduling problem. Instead of a single or multi-agent system with centralized or isolated decision-makers, they introduce an interactive, decentralized algorithm called Consensus Bundle-Based Algorithm (CBBA) that allows the agents to share local decide task allocation as a whole. This algorithm is essentially a continuous process of bidding among the satellites in the constellation under a series of ordered tasks. The process offers scalability and knowledge sharing, which is key for bigger networks. However, to enable more reactivity in the planning, they later introduced an improved approach called Asynchronous CBBA (ACBBA), in which the bidding process happened under a specific request from a member of the constellation, with the help of on-board event detection. [17]

2.6 Summary

May the main methods previously mentioned be listed below.

- Constrained Optimization with Temporal Constraint Network (TCN)
- Dynamic Programming (DP)
- Mixed Integer Linear Programming (MILP)
- Monte Carlo Tree Search (MCTS)
- Deep Reinforcement Learning (DRL) with diverse Neural Networks (NNs)
- Asynchronous Consensus Bundle-Based Algorithm (ACBBA)

Form this perspective, with the main algorithms ordered in chronological order[2], one can see how the scheduling problem was initially formulated using graph-based or tree-based methods. These were capable to track the

[2] A couple of them are not really in chronological order. For instance, the DRL from Chen et al. was introduced in 2019 and the MCTS form Herrmann et al. was released in 2024. This is semi-chronological to give a fair perspective of how the nature of the algorithms used evolved.

entire set of tasks that an agent needed to allocate them properly. However, and despite how near-optimal they could get to be like DP, the general issue was computational runtime.

These were heavily dependent on the fact that the satellite would be hovering over a place on Earth that was of no interest to their mission, allowing a long enough wait to plan the next window of moves. However, as the missions get more complex and are intended to be more efficient, these planners escape from performance requirements and new AI-based approaches arise.

The good side of AI-based algorithms is that the execution runtime is extremely low. A usual NN can do a forward pass in a negligible amount of time. Even larger networks with complex attention mechanisms (which are relative consuming) take longer that NNs but still lie orders of magnitude far away in runtime from what a graph-based approach would take.

Chapter 3

Problem Definition

In the literature, several kinds of algorithms and frameworks to tackle the planning problem for Earth Observation satellites have been mentioned. Out of all of them, their formulation was purely based of tasks allocation, that is why it they usually call it scheduling problem. This basically consists in having a set of observables, probably targets on Earth's surface or downlink to ground bases for data collection, and then do all the planning side of things where these tasks are ordered and allocated.

However, in this thesis, the aim is to bring this problem to a lower level, where the agent will try to learn the movement basis of the satellite maneuvering. This approach is more fundamental and, at first sight, maybe more complex.

Also, regarding the algorithmic part of the definition, this will be performed with Deep Reinforcement Learning. The specific algorithm will be mentioned on the general discussion, since many hypotheses and deeper formulations are yet to be drawn.

Finally, there will be an imposed requirement to the work, which is to try to figure out a way to unveil the unprecedented potential that the Transformer architecture, presented by Vaswani et al. in 2017, has shown over plenty of different study cases in the field of AI. This does not include to be using only the architecture shown in the paper itself, but trying to leverage the sequential nature that makes this architecture so powerful, given that the application it will be given is also under this condition. [16]

3.1 Research Questions

Here there are the three research questions that the work will try to answer.

1. Can we make an on-board, real-time, Earth Observation Satellite maneuvering decision-maker with modern Reinforcement Learning algorithms capable to outperform the general scheduling formulations?
2. Which model architecture and algorithm are most suitable for the task in (1)?
3. To what extent can the model work as the central brain of a multi-agent system?

Essentially, the two first research questions (1, 2) focus on wether this approach previously mentioned is going to be enough to build a better planner than the literature has shown. The third focuses on performance instead. The idea is to first let the model be for one satellite, and then make general enough for a centralized planner for an entire constellation.

After that, see if this model is capable to understand the nature of having different elements in the constellation and looking for near-optimal solutions. Meaning, taking some actions for a certain satellite considering the position and observation of the other satellites too.

Chapter 4

Methodology

The overall schedule of the project resolution worked as a weekday work at the laboratory followed by a weekly meeting with Dr. Selva to discuss the progress of the work and the next steps to take. This was usually held every week except for calendar incompatibilities and usually on Fridays.

The general topic of the thesis is highly related to the topic a NASA-granted project in the laboratory. However, this approach was particularly different. Then, despite the thesis was all self-contained on its own, many colleagues in the laboratory were able to provide guidance on the field and present their point of view to certain questions as additional knowledge to the one Dr. Selva himself provided.

4.1 Tools and Technology

All the coding of the three repositories used for the thesis were done locally on a personal desktop at the laboratory and stored online in the GitHub official organization account of the research group. All of these have MIT License, which is a very permissive open-source license that allows to copy, modify, merge and publish the work done inside them.

Also, one of the repositories requires a licensed software service, called STK12 from Ansys corporation. Ansys develops highly technical software not only for orbit propagation, but also for fluid mechanics, structural design, etc. The license used for it was provided by the Dr. Selva and the Texas A&M university, which is held for educational and research purposes.

Regarding software, all the three repositories were mainly coded in Python, given the high adaptability of the state of the art machine learning libraries. The one used is PyTorch, developed by Meta AI. In term of operative system, Windows Subsystem for Linux was used to have a Linux environment in the personal Windows machine. This allowed less overhead in the code execution and better performance. Such a thing was a little tricky at the beginning, given that the STK12 software has not been implemented in Linux until relatively recently. Despite some complications, it could be finally installed in a Linux environment, where the graphical user interface (GUI) was not used.

Finally, in terms of performance, the use of a graphics processing unit (GPU) could be leveraged to run the biggest architectures used in the testing phase. For this, CUDA, a software developed by NVIDIA to use their GPUs for parallelized processing, could be implemented within the PyTorch library and consistently improve the capabilities to run bigger models with less computation time. This was not efficient for smaller models, but it was just present when configured by the user. The GPU used for these particular cases was a NVIDIA Quadro RTX 4000.

4.2 Gantt Chart

The entire procedure of resolution must follow an ordered workflow. For this, a Gantt Chart has been created, where the very initial steps already gone through in this report were included just as well as the tasks that follow them. This Gantt Chart has been adjusted to what the final scheduling of the thesis actually looked like.

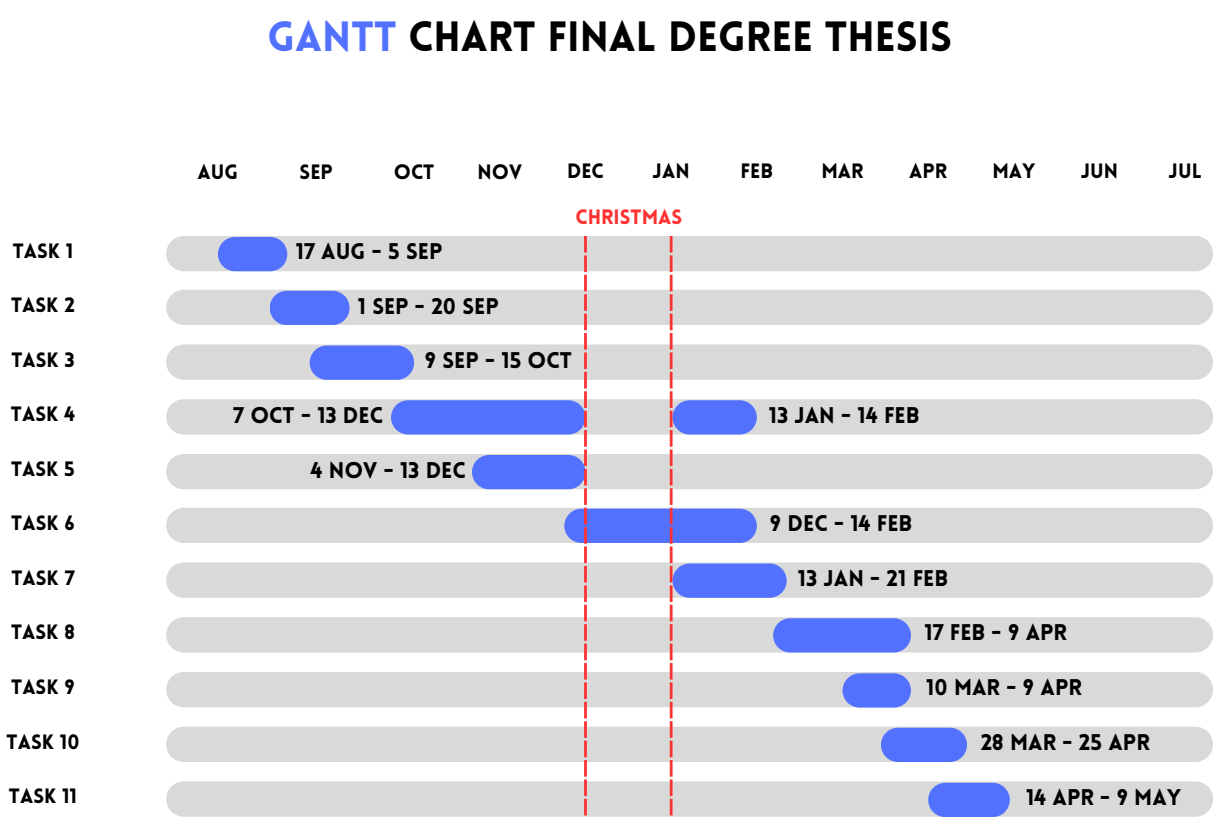

Figure 4.1: Gantt Chart of the thesis.

Below lie the descriptions and explanations for each of the tasks shown in the Gantt Chart.

1. **Define specific topic:** out of the diverse propositions to work within the scope of my intentions and Dr.

Selva's laboratory, there was the need to define a specific approach to the very preliminary intentions of working with machine learning in aerospace.

2. **Literature review:** once defined the topic, learning the basics of the scheduling problem for the satellite maneuvering is a basic and necessary step for further development.
3. **Learn about ML, RL and transformers:** given the background in aerospace and physics, some foundations of knowledge about machine learning, reinforcement learning, transformer architecture and other aspects of the field were necessary.
4. **Code Earth Gym:** one of the largest task was the overall creation of the reinforcement learning environment with which to train the agents.
5. **Code Soft Actor-Critic :** coding the algorithm based on its respective journal article for later testing.
6. **Test Soft Actor-Critic:** run trainings with this algorithm.
7. **Code Proximal Policy Optimization:** code the algorithm based on its respective journal article for later testing and comparison.
8. **Test Proximal Policy Optimization:** run trainings with this algorithm and compare results with the previous one.
9. **Prepare thesis presentation:** make some slides for the presentation at the lab with all the members and Dr. Selva, as well as prepare the oral presentation which would take around 40 minutes in addition to respective questions.
10. **Test final configurations:** run longer, final trainings with the best model-algorithm hyperparameter configurations.
11. **Write thesis report:** concentrate all the procedure and results of this set of tasks into a single report.

Also, one must consider the Christmas break in the middle of the period, as shown in Figure 4.1, where the amount of working hours was drastically reduced during that period. That time, then, was very profitable to perform a good amount of training runs for the SAC algorithm.

To clarify, the Earth Gym code was almost in constant development. The very first versions were enough to be used for the SAC testing. However, given the use of a third-party software service for the orbit propagation and many other calculations, the runtime was not ideal, so there has been put a lot of effort to make it as time-efficient as possible, allowing more configurations to be tested. Also, some additional capabilities have been added along the way.

Chapter 5

General Discussion

In this chapter, all the aspects of the problem resolution will be treated. From the Reinforcement Learning theory side of this, going through the architecture behind the learning model and the code generated to implement these ideas.

First let's do an overview of the method, i.e., the RL formulation, so to see what the overall algorithm will look like and parts it needs.

5.1 Reinforcement Learning

Inside Machine Learning (ML), a few subfields identified by how the learning data is acquired or processed are encompassed. Their difference can be thought as they way the learn too. These are:

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

One could make an analogy with supervised of a kid being taught math in a class with a teacher that clearly guides the knowledge to acquire, which is a very efficient way to learn time-wise. On the other hand, reinforced could be compared to a baby trying to walk, where falling would cause pain and hence act as a negative incentive to the actions taken moments before. Whereas unsupervised, would be a way to learn from poorly processed data, maybe in an environment where no incentives are created and no guides are mentioned, such as Newton observing nature, and trying to figure out how the laws of motion work by seeing an apple falling from a tree.

These diverse ways of machine learning are used based on the amount of data one has from a problem. In this case, where the goal is to match certain maneuvering actions to getting good observations, it is very difficult to generate data matching these two events. One would need to generate lots of information of existing or simulated satellites moving in an orbit. And, given the complexity and the uniqueness of each orbit, this data would require terabytes of storage to contemplate every case possible and of interest. Aside from being a lot

of data to acquire and then have as an intake, its processing would be too painful and long-lasting. Regarding unsupervised learning, a similar problem arises, the problem to learn as an agent is not a linear regression or similar. As mentioned in some papers of the literature, this is an NP-hard problem, where the near-optimal solutions are everything but trivial to find. [15]

Hence, the natural solution for a problem of this shape would be to go for reinforcement learning. This subfield is known to be very intuitive to understand the general idea (not further than that). It is based on having an agent that interacts within an environment that is supposed to be its own world, where it will perform after training too (or similar at least).

From there, the environment can tell the agent which and in what magnitude actions are better than others. Being actions the way the agent has to interact with the environment, rewards are the way the environment has to tell the agent which incentives to follow.

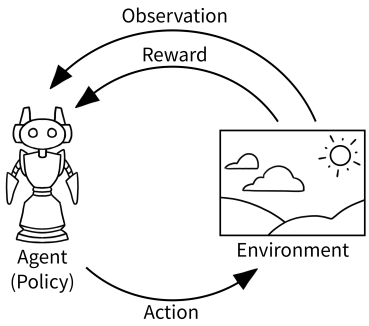

Figure 5.1: General concept of an RL system.

The usual formulation in RL is done though what is called a Markov Decision Process (MDP).

5.1.1 Markov Decision Process

Markov Chains (MCs) are sequence graphs used in statistics that model the behaviour of a state-space progression where the probability to move from one state (node) to another is stochastic and only depends on the current state. This last feature is called the Markov Property.

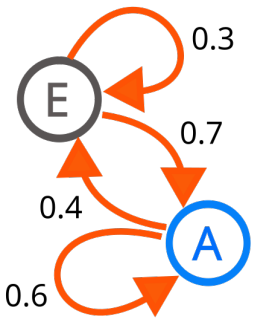

Figure 5.2: Small, two-node MC example.

MCs can be expressed as a transition probability matrix, which define the probability of going from one state to another. Considering, from Figure 5.2, that A is the first state and E the second one, the matrix would be:

$$P = \begin{bmatrix} 0.6 & 0.4 \\ 0.7 & 0.3 \end{bmatrix} \tag{5.1}$$

See the property that all rows sum up to 1, given that they group the probability to migrate form one state, which is certain.

MDPs, then, are an extension of MCs, where they add actions and rewards to the system. Formally, MDPs are defined by the tuple:

$$(\mathcal{S}, \mathcal{A}, P, r) \tag{5.2}$$

These elements represent, respectively:

- $\mathcal{S}$: state space, which represents the domain of configurations where the agent and environment can be uniquely identified.
- $\mathcal{A}$: action space, the set of capabilities the agent has to navigate within the state space.
- P: the transition probability function, which is the function that determines the probability to be moving form state s to state s' once taken action a. It is expressed as $P(s'|s, a)$.
- r: reward function, expressed as $r(s, a)$ (or, more generally, $r(s, a, s')$), that determines how good and or bad and in what measure a certain state-action pair or state-action-next action trio is.

Discounted MDPs are represented by the tuple

$$(\mathcal{S}, \mathcal{A}, P, r, \gamma) \tag{5.3}$$

and the γ they include is a value between 0 and 1, $\gamma \in (0, 1)$, called the discount factor.

This formulation is a very useful mathematical formalism to express real-life control problems, and it is the basic mathematical approach to RL. And, so that the mathematical formulation holds, MDPs must fulfill the hypothesis of the Markov Property, existent in MCs already, as mentioned before.

As said, navigation among the state space is done by the actions taken from within the action space, which are controlled by the policy π. The policy is the learnable feature of an agent that defines its behaviour. The probability of taking action a form state s is expressed as $\pi(s, a)$.

A policy can be good or bad in some measures, based on whether the agent really does what it is incentivized to do or not. They way to mathematically determine this is the value function $V_\pi(s)$ of the policy.

$$V_\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, \pi\right] \tag{5.4}$$

This formulation can also take the form of:

$$V_\pi(s) = r_\pi(s) + \gamma(P_\pi V_\pi)(s) \tag{5.5}$$

This means the discounted sum of rewards from a certain trajectory, i.e., a set of consecutive actions, taken by the policy in question. From that, one can define the optimal policy π^* as the policy with the highest value over the set of all policies Π.

$$\pi^* = \arg\max_{\pi \in \Pi} V_\pi \tag{5.6}$$

Note that the policy is what actually induces a Markov Chain, given that the probability transition matrix within an MC is $P(s, s')$, but the MDP lacks this transition function shape since there is another variable to be considered, the action. For a fixed policy $\pi(s, a)$ in an MDP, its transition function $P(s'|s, a)$ reduces to $P(s, s')$.

Another useful function in RL algorithms is the action-value function of a policy $Q_\pi(s, a)$, defined as

$$Q_\pi(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a\right] \tag{5.7}$$

and can also be formulated as:

$$Q_\pi(s, a) = r(s, a) + \gamma \sum_{s'} P(s' \mid s, a) V_\pi(s') \tag{5.8}$$

Just like with the policy, there exist the optimal value and action-value functions, which are linked to the optimal policy too. What is of most interest is to find methods that help to find the these optimal functions.

From dynamic programming, Richard Bellman introduced the Bellman equation which was then brought into the field of RL. With it, and from the definition of value function, the Bellman Optimality Equation came to place.

$$V^*(s) = \max_a \left(r(s, a) + \gamma \sum_{s'} P(s' \mid s, a) V^*(s') \right) \tag{5.9}$$

This represented the form of the optimal value function, to which one would get by iteration with Bellman Operator. This operator, $T : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$, is applied to a value function as expressed in the following equation.

$$(TV)(s) = \max_a \left(r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[V(s')\right]\right) \tag{5.10}$$

The operator can be proved to be a contraction mapping when having a bounded MDP (discount factor lower than the unit) with respect to its infinity norm $\|\cdot\|_\infty$.

$$\|TV_1 - TV_2\|_\infty \leq \gamma \|V_1 - V_2\|_\infty \tag{5.11}$$

This means that, by Banach's fixed point theorem, the application of such operator is guaranteed to converge to the optimal value function. This leads to the Bellman Equation applied to the value function.

$$V_\pi(s) = \mathbb{E}_{a \sim \pi(s)} \left[r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[V_\pi(s')\right]\right] \tag{5.12}$$

These can be applied to the action-value function too, using the operator $F : \mathbb{R}^{|\mathcal{S}||\mathcal{A}|} \to \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$, which differs from T on the dimensions due to the function it is applied to. However, it still follows the same principle.

Planning Problem Formulation

Given this overview of the mathematical formulation, one can define the discounted MDP tuple as best fits to the case of study.

Firstly, though, let's have a qualitative perspective of what the problem looks like. The system is based on an environment to be the Earth itself and the surrounding space, where the agents, satellites, will orbit. The Earth will be filled with points that represent events of observation.

On their orbit, the satellites will, of course, be constantly moving through it and they will forcibly have a sensor. Sensor can have different specifications and shapes, but these will mainly be simple conic with a certain degree of aperture. The aperture is usually quite low, given the observability limitations mentioned in Section 1.2.

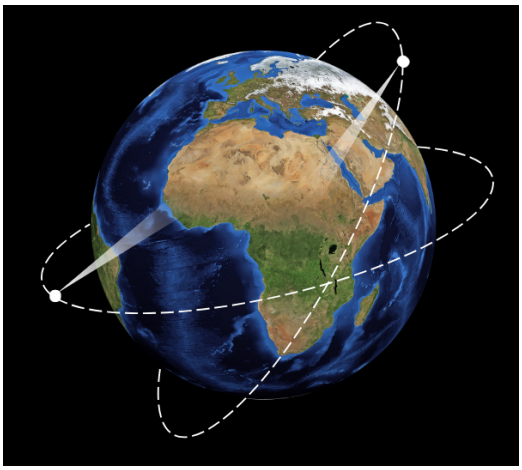

Figure 5.3: Schematic representation of the system.

The way satellites will be able to modify the orientation of the sensor, i.e., the position of the Field of View (FoV) within the Field of Regard (FoR) is by making attitude maneuvering. This maneuvering will mainly be among their pitch and roll.

Also, it will be enforced to have continuous variables for each of the properties previously mentioned. This way, maximum precision (floating point) is ensured.

In this problem, the code built to use the algorithms has a lot of flexibility to use diverse state spaces, action space, reward spaces and discount factor. However, the general features of it must be clarified first.

- $\mathcal{S}$: values related to
  - Orbital coordinates
  - Detic coordinates
  - Attitude orientation
  - Points of interest
- $\mathcal{A}$: values related to
  - Differential of attitude orientation
- P: too complex to be modeled. Hence, unknown.
- r: dependency on
  - Priority of point observed*
  - Angle of point observation*
  - Re-observation times*
  - Slew constraint**
  - Coverage grid**

  *For every point observed in the set of targets available
  **If used
- γ: usually takes high values (within its domain) to ensure focus on future rewards, like 0.99.

The orbital coordinates in the state space can be expressed by six variables with different configurations. Mainly, the six orbital parameters

- a: semi-major axis.
- e: eccentricity.
- i: inclination.
- Ω: longitude of the ascending node.

- ω: argument of periapsis.
- ν: true anomaly.

with the tuple $(a, e, i, \Omega, \omega, \nu)$ or the components of the position and velocity of the body, (x, y, z, v_x, v_y, v_z). The detic coordinates coordinates represent the latitude and longitude of the nadir projection of the satellite on Earth's surface and its altitude. Finally, the attitude represents the actual pitch and roll (θ, ϕ), respectively.

Aside from the positional features, the state space also includes information about the points of interest, taking the form of latitude, longitude coordinates, as well as the priority of the event in question.

In terms of the action space, it simply is the increment in the pitch and roll required to move $(\Delta\theta, \Delta\phi)$. This is not directly translated into an increment in the values, but in a three-dimensional rotational transition where roll is rotated first and pitch afterwards. The roll, pitch and yaw rotations are applied in different axis, matching the three spatial coordinates. Them all put together form the matrix

$$\begin{aligned} R(\phi, \theta, \psi) &= R_z(\psi) R_y(\theta) R_x(\phi) \\ &= \begin{pmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi \\ \sin\psi\cos\theta & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{pmatrix} \end{aligned} \tag{5.13}$$

which, considering the yaw increment to be null reduces to:

$$R(\phi, \theta) = \begin{pmatrix} \cos\theta & \sin\theta\sin\phi & \sin\theta\cos\phi \\ 0 & \cos\phi & -\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{pmatrix} \tag{5.14}$$

About the transition function, it not a trivial function to figure out. Given the requirement to have a continuous state space and action space, one can not assign a three-dimensional matrix to it. Instead a function must be created, but would take an extremely complex function to model this behaviour. It will be considered that the transition function is unknown. Having it is not absolutely necessary, since many algorithms assume that form the beginning.

Finally, let's bring some light to how the rewards will be computed. Here it is very important to see that it is also not a fixed mapping of states and actions to a real value, given the continuous requirement. Instead, simple functions that depend on values from the experiences made by the agent will result in a value for it. Qualitatively speaking, the reward is based on how many points are observed and the quality of the observation made. φ is the elevation angle of the observation, i.e., the angle between the line from the target to the satellite and the flat surface tangent to Earth's surface at the target. Also, assuming n, is the number of times the event has been observed (counting the present one) and p is the priority of the event in question, the reward is

$$r_{targets} = f_{priority}(p) f_{angle}(\varphi) f_{reobs}(n) \tag{5.15}$$

where the partial reward functions are computed as

$$\begin{cases} f_{priority}(p) = p^{k_1} \\ f_{angle}(\varphi) = sin(\varphi)^{k_2} \\ f_{reobs}(n) = \frac{1}{n^{k_3}} \end{cases} \tag{5.16}$$

and the different k_i mean custom parameters to adjust the weight of each incentive. Also, this function is computed for every target observed only if the target was observed for a period longer than k_4.

In addition to that, it is known that for a satellite in orbit during several years (as seen in Chapter 1) the energy consumption is a crucial aspect of the mission. The satellite maneuvering originates a lot of fuel consumption, so it is quite a necessity to be implementing a policy that cares about how much the satellite rotates. This is seen in the slewing constraints added into the reward function, which are negative reward based on how much movement is performed.

$$r_{slew} = -(k_5 \Delta\theta + k_6 \Delta\phi) \tag{5.17}$$

As seen in Equation 5.17, the k_i parameters are scaling factors for the increment in pitch and roll.

Just like the slewing constraint, a coverage grid can be used if necessary. With it, additional rewards can be given to the agent based on the points observed in a predefined and Earth-static grid and how close these points are to real targets. This way, the agent can receive help on where the points are not only by observing them but by observing Earth's surface close to them. Such a help would make learning easier in configurations of sparse rewards, where the agent is struggling to see targets.

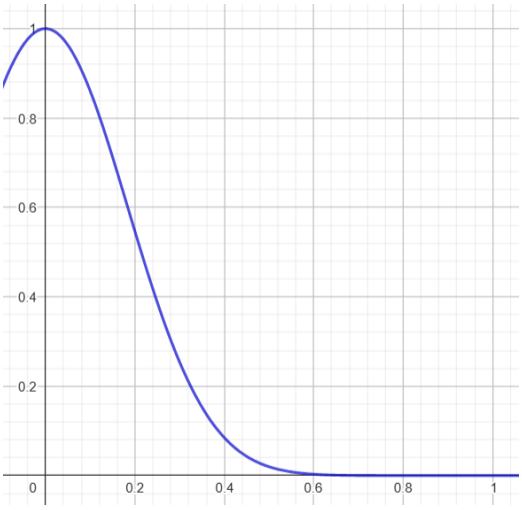

Figure 5.4: Decay cosine function for the coverage grid.[1]

Figure 5.4 shows a cosine function elevated to a high power that it is what is used to multiply the reward gained

[1]Image generated with GeoGebra.

by observing points in the coverage grid. This way, the closest these points are to the real targets, the more this coefficient is close to 1. However, it is of high interest to make this coefficient very low pretty quickly to avoid the agent focusing too much on grid rewards and completely ignoring the real targets to observe. This particular function is quite arbitrary, some others like an exponential decay scaled to fit within the desired bounds would also be a suitable option.

Once that, the overall reward achieved in a time step would be:

$$r_{total} = r_{targets} + r_{slew} + r_{grid} \tag{5.18}$$

5.1.2 Algorithms Overview

Chosen the formulation, the next step to proceed with the development is to choose an RL algorithm capable of training the agent. These algorithms can be classified from different perspectives. An easy, binary classification is whether a model is model-based or model-free.[18]

This distinction depends on whether to use or not to use the model of the environment to learn from it (and even predict it). However, this first step is quite easy to take since, in this case, the model of the environment would be extremely complicated to get, as mentioned in the problem formulation. Hence, the algorithms to use must be model-free.

Out of these, the main and most modern ones are shown below.

Policy Optimization		Q-Learning
Policy Gradient	AWR	DQN
A2C / A3C	DDPG	QR-DQN
TRPO	TD3	C51
PPO	SAC	HER

Figure 5.5: Overview of model-free RL algorithms.

In Figure 5.5, a distinction is made among the algorithms present. Some of them are centered on Policy Optimization, some others use Q-Learning and the rest are a combination of both.

Policy Optimization is based on a policy iteration that eventually seeks the optimal policy directly. Policy gradient is one of the first of this kind. The most modern of these, Trust Region Policy Optimization (TRPO) for instance, are based on the concept that one must find better policies within a certain trust region, as the name says. This concept seemed to work so well that Proximal Policy Optimization (PPO) is a child algorithm of TRPO, and highly probably the most used algorithm nowadays. For this reason, it will be one of the algorithms used in this work.

On the other hand, Q-Learning is based on training Q-Networks and let the policy learn from them. One of the first algorithms of this kind was Deep Q-Network, introduced in a very famous paper and caused a lot of

different variations from it. Double DQN (DDQN) and Quantile Regression DQN (QR-DQN) are a couple of them. Hindsight Experience Replay (HER) and others will not be further considered for the case.[19]

Within the one in the middle column of Figure 5.5, all of them are highly used too. Deep Deterministic Policy Gradient (DDPG) is a parent algorithm of Twin Delayed DDPG (TD3). However, Soft Actor-Critic (SAC) gets the biggest interest for this topic. It is not only quite recent and used, but also seems to perform particularly good in continuous state and action spaces. Given that this work aims to be fully continuous if possible, SAC will be the other algorithm to work with.

5.1.3 Soft Actor-Critic

The Soft Actor-Critic (SAC) algorithm was introduced by Haarnoja et al. in 2018 with a publication called *"Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor"*.[20]

This is an off-policy algorithm that uses a Replay Buffer $\mathcal{D}$ to store experiences and optimize from samples of it. The fact of being off-policy means that it can optimize from samples taken with a policy long outdated to the one it has in the actual moment. This is a great property to achieve since off-policy algorithms can be very sample efficient. This means they can make a lot of updates compared to the experiences taken.

The general shape of the process is to loop over iterations, which contain two loops inside. The first one is to capture experiences from the environment, called environment steps, and the second one is to optimize, called gradient steps. Below lies a pseudocode of it.

Algorithm 1 Soft Actor-Critic[20]

Initialize parameter vectors ψ, $\hat{\psi}$, θ, ϕ
for each iteration **do**
  for each environment step **do**
    $a_t \sim \pi_\phi(a_t|s_t)$
    $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
  end for
  for each gradient step **do**
    $\psi \leftarrow \psi - \lambda_V \nabla_\psi J_V(\psi)$
    for $i \in \{1, 2\}$ **do**
      $\theta_i \leftarrow \theta_i - \lambda_Q \nabla_{\theta_i} J_Q(\theta_i)$
    end for
    $\phi \leftarrow \phi - \lambda_\pi \nabla_\phi J_\pi(\phi)$
    $\hat{\psi} \leftarrow \tau\psi + (1 - \tau)\hat{\psi}$
  end for
end for

The θ_i represent the Q-Networks' parameters, where there are too in order to reduce the overestimation bias[2]. It could be done with one only, but the original paper uses two already, and they clearly mention there is an increase in performance. This is why the same approach will be applied here.[20]

[2]See the Annex for explanation about the overestimation bias.

The ψ and $\hat{\psi}$ represents the Value Network and Target Value Network parameters, respectively. The value network receives updates from backpropagation and optimization just like the Q-Networks, but the Target Value Network is softly updated to converge to the normal one but at a slower pace ($\tau << 1$).

The ϕ represents the Policy Network's parameters. The policy is considered to be the actor, whereas the other networks are the critics.

Now, let's see which are the losses equations that allow the policy to learn. In the first place, the loss function of the critics is basically defined by a mean square error of targets and actual values (networks' outputs). The V-network critic loss is:

$$J_V(\psi) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[\frac{1}{2} \left(V_\psi(s_t) - \mathbb{E}_{a_t \sim \pi_\phi} \left[Q_\theta(s_t, a_t) - \log \pi_\phi(a_t|s_t) \right] \right)^2 \right] \tag{5.19}$$

While that, the Q-network critic loss is

$$J_Q(\theta) = \mathbb{E}_{(s_t, a_t) \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_\theta(s_t, a_t) - \hat{Q}(s_t, a_t) \right)^2 \right] \tag{5.20}$$

where the estimator $\hat{Q}$ follows the Bellman equation:

$$\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p} \left[V_{\bar{\psi}}(s_{t+1}) \right] \tag{5.21}$$

Now, the loss of the actor (and hence the policy) is computed by cross-entropy criteria using the Kullback-Leibler divergence. The loss itself is the expectation over the states in the buffer of the expectation over the actions in the policy to have the cross-entropy stated below.

$$Cross - Entropy = log \left(\frac{\pi_\phi(a_t|s_t))}{\frac{exp(Q_\theta(s_t, a_t))}{Z_\theta(s_t)}} \right) \tag{5.22}$$

This leads to the actor loss

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}, \epsilon_t \sim \mathcal{N}} [log(\pi_\phi(a_t|s_t)) - Q_\theta(s_t, a_t)] \tag{5.23}$$

where the action is output from a stochastic function by using the so-called reparametrization trick. This makes the stochasticity of the policy be in terms of a noise rather than a direct Gaussian from the policy so that differentiation during backpropagation can be done.

$$a_t = f_\phi(\epsilon_t, s_t) \tag{5.24}$$

In this work the actual values after reparametrization will be

$$a_t = \mu_t + \epsilon_t \sigma_t, \quad where: \ \epsilon_t \sim \mathcal{N}(0,1) \tag{5.25}$$

where the μ and σ represent the mean and standard deviation of the action. This format implies that the model architecture will have to output double the amount of action dimension. From there, content will need to be reorganized so that the mean and standard deviation are properly extracted and used.

Notice that, in practice, the probabilities of states and actions coming from the expectation formula are not computed. The practical presence of them is within the possibility of getting them from the buffer or from the Gaussian distribution, respectively.

In the gradient steps, such differentiation is performed by automatic differentiation, using the loss functions defined in the paper. When the logarithmic probability of the policy is required, one has to find the value of the the probability density function to obtain such action. Since the Gaussian probability density distribution follows the expression

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-0.5(x-\mu)^2/\sigma^2} \tag{5.26}$$

one must compute the log[3] of the product of this function for each of the features in the action. This turns into:

$$log(\pi_\phi(a_t|s_t)) = -0.5 \sum_i \left(\left(\frac{a_i - \mu_i}{\sigma_i} \right)^2 + log(2\pi\sigma_i^2) \right) \tag{5.27}$$

Also, notice that, as said before, the overestimation bias from the Q-functions leads us to choose the lowest value from both of them for the loss computation of the V-network and the policy. Hence, whenever a Q-value is asked for their equations, they should be given the lowest of the two.

5.1.4 Proximal Policy Optimization

Prior to SAC, Proximal Policy Optimization was introduced by Schulman et al. in 2017. The publication was called *"Proximal Policy Optimization Algorithms"*.[21]

Differently to the SAC, PPO is an on-policy algorithm, meaning that it computes updates based on experience form the current policy or a very recent one. Also, as mentioned before too, it follows the principle of updating the policy within a certain trust region, which avoids going through drastic changes.

Below there is a pseudocode of the algorithm from the original paper[4].

The θ and ψ correspond to the Policy Network and Value Network parameters, respectively. The process itself is quite simple. First, T experiences are collected. This is the horizon, and all of these experiences are computed with the current policy. With all the data from it, the advantage estimates are computed:

[3] See the Annex for further explanations on enforcing the log to be corrected after bounding the action space of the system.
[4] Here, the update of the value network parameters was added for clarification.

Algorithm 2 Proximal Policy Optimization[21]

for iteration $= 1, 2, \ldots$ **do**
  for actor $= 1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L w.r.t. θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
  $\psi_{\text{old}} \leftarrow \psi$
end for

$$\hat{A}_t = -V(s_t) + r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V(s_T) \tag{5.28}$$

In the paper, they specify a more generalized way of the advantage

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1} \tag{5.29}$$

where λ is introduced to truncate this estimator. It reduces to Equation 5.28 when $\lambda = 1$. Also, let δ_t be:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{5.30}$$

Having the advantage, the next step is to find the probability ratio

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \tag{5.31}$$

where the value would be 1 in case of using the same policy for both new and old. This quotient is needed to compute the policy loss, and the numerator of it (i.e. newest-policy probability density) will be the only factor in charge of gradient tracking, from where to compute backpropagation. A few options of policy surrogate are proposed in the paper:

$$\begin{cases} \text{No clipping:} & L_t(\theta) = r_t(\theta)\hat{A}_t \\ \text{Clipping:} & L_t(\theta) = \min\left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right) \\ \text{KL penalty:} & L_t(\theta) = r_t(\theta)\hat{A}_t - \beta\,\text{KL}[\pi_{\theta_{\text{old}}}, \pi_\theta] \end{cases} \tag{5.32}$$

The KL penalty can be fixed and or adaptive. If fixed, the β will take a single value. If adaptive, the $beta$ will try to make the penalty factor match a certain arbitrary target. The clipped seems to be the best performing of all versions, where $\epsilon \approx 2$ seems to be a reasonable value in most cases. Also, a mix between the clipped loss and the KL penalty can be used, where it essentially follows the clipped surrogate but then adds up with the entropy factor.

Finally, the Value Network loss is computed as a simple mean squared error of the network's value and the target estimate coming from the trajectory rewards.

$$L_t(\psi) = \frac{1}{2}\left(V_\psi(s_t) - \hat{V}_t^{\text{target}}\right)^2 \tag{5.33}$$

In this work, the mixed version between the clipped and the KL penalty will be used, since clipping it is the most promising one and a small penalty as an addition could properly enhance the incentives to perform exploration.

5.2 Model Architectures

What remains to be commented on behalf of the RL implementation is the architectures used for the models. Given the complexity of the problem, dense, multi-layered networks will be used to ensure sufficient learning capability. This turns into the subfield of Deep Learning (DL).

There are plenty of functional possibilities regarding artificial networks the potential of which could be leveraged. The following list summarizes the options available.

- Neural Network (NN)
- Recurrent Neural Network (RNN)
- Gated Recurrent Unit (GRU)
- Long Short-Term Memory (LSTM)
- Convolutional Neural Network (CNN)
- Spiking Neural Network (SNN)
- Transformer (and variations)

NNs are the original architectures DL. Several layers of matrix multiplications and biases in between can be stacked to ensure a big enough set of function approximation parameters. RNNs, LSTMs and GRUs are child architectures of NNs but include memory features. This means that a forward pass result is not only dependent on the input but on the previous too.

CNNs are mostly used for two-dimensional data structures, images for instance. These are capable to turn the structures into vectorized forms. In this case, this is not a need unless the case of feeding the agent the image of the Earth and the targets in it. Just as interesting as it looks, this approach still falls out from the scope of this project by now.

SNNs are a more recent form of networks that resemble the structure of biological neurons. The state of the art's intention is to match the unprecedented efficiency of the human brain's capacity and consumption. However, for this purpose, this special form of artificial network is not particularly needed.

Last but not least, the Transformer architecture is a much more interesting one. Since it was introduced in 2017 by Vaswani et al., the world of AI has been revolutionized. Its capabilities are incomparable with the rest of the

architectures for some specific tasks like Natural Language Processing (NLP).[16]

Transformers fall into the group of networks that are suitable for sequential tasks. Just as RNNs, LSTMs and GRUs, they use previous information to process current one. However, they do it in a more self-contained manner thanks to the great contribution of the attention mechanism. Due to the sequential nature of the problem in this work, this architecture will also be implemented, instead of the other ones with memory.

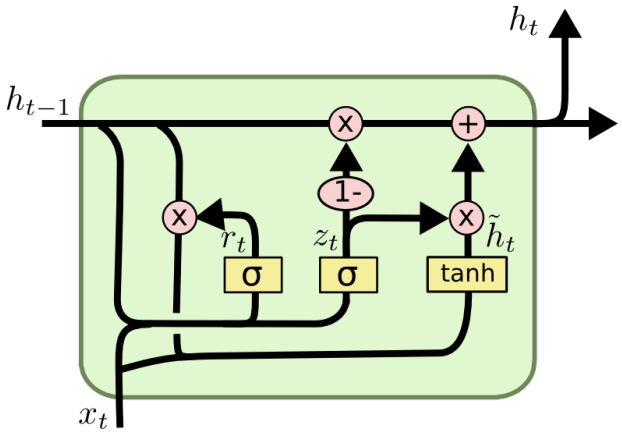

Figure 5.6: Inner structure of an LSTM to see memory technique. [22]

5.2.1 Transformer

The usual NNs have a very simple working principle. There is no further comment than the fact that they are concatenated matrix multiplications. However, the Transformer is a different story.

The *"Attention Is All You Need"* publication by Vaswani et al. found out a way to leverage the potential of the attention mechanism in a very particular way to manage the inputs and outputs of the model.

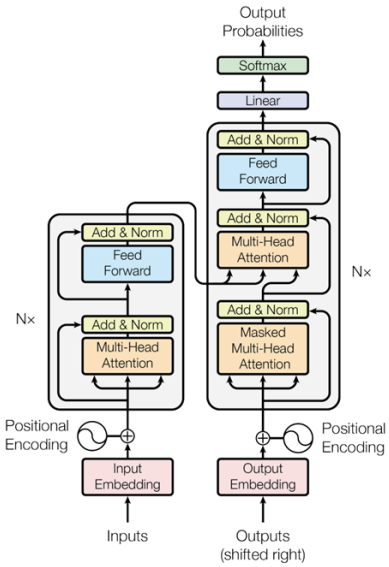

Figure 5.7: General structure of the transformer. [16]

Figure 5.7 is a scheme of general architecture. It is comprised by an encoder (left block) and a decoder (right block). The input of the encoder would be information providers, and the input of the decoder would the output of the whole model shifted right. This means that the decoder gets in and out data of the same nature. The shifted right feature is done because, how the transformer usually works is to have a retro-fed loop where the output concatenates with the previous input to form the newer input and then run it again. Hence, the inputs do not consist only on one state-action experience, but several of them. Actually, in modern NLP applications, it consists on thousands of them.

In the context of this RL problem, the input of the decoder shall be what is called states. The input of the decoder, as well as the overall output, shall be the actions. This is based on how it was framed during the planning problem formulation. However, as explained in Section 5.1.1, the MDP is defined to be using only the current state as the total information necessary to produce an action. So there is a slight conflict in this.

The explanation to this is that, despite the formulation detailed before, and to avoid abuse of notation, the RL side of things will follow a different naming convention to what was stated. What will be called the *augmented state* is going to be the set of states in the encoder and actions in the decoder that would produce a unique action output. From the RL perspective, this is the most proper formalism. However, it was not framed this way due to the use of variations in the architectures that might be using a different augmented state. For instance, NNs, also called Multi-Layer Perceptrons or MLPs, might not be using actions in the augmented state.

Let this be clarified, let's continue with the transformer overview. There is a very important hyperparameter in it called d_{model} which defines the magnitude of the arrays to be processed inside the transformer. This magnitude may be different form the states dimension and actions dimensions, so an embedding class to make this dimensions adaptation is need.

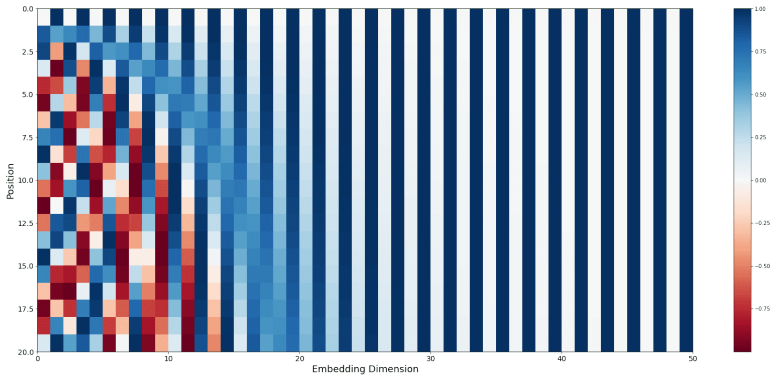

Figure 5.8: Positional encoding value for the first 20 elements and $d_{model} = 50$. [23]

After the embedding, a process called positional encoding (PE) is made. In here, the objective is to give the diverse elements in the encoder and decoder sequences a nuanced value variation to add positional value to it. Figure 5.8 shows the values to be added with the original approach of sine superposition with different frequencies. This step is necessary because the inner structure of the transformer or the attention mechanism is completely symmetric with all the elements in the sequence, adding one before another or vice versa does not make any difference. This is why a notion of time series order is needed to learn teh sequential nature and

dependencies of the problem. These value, which are applied every time a forward pass through the model is done, are enough to make it understand this property.

The one in Figure 5.8 is just one form of PE. There exist other which might be more appropriate for certain cases. For instance, learned parameters to determine the encoding, embeddings rotation... In this work, only the original version (sine superposition) and a segmented version (explained in Section 5.2.2) will be used.

Finally, notice that in Figure 5.7 the three attention mechanisms that exist are differently applied. In the original paper (where the picture is from), it was a translation case of study where the encoder's input did not have any temporal dependency, since the translator has all of the sentence from the beginning. For this reason, there exists no masking in the multi-head attention (MHA) of the encoder. However, this work does have a temporal dependency within the different elements of the states sequence. This forces the need to have a masked[5] MHA there. Also, the cross-attention MHA of the decoder (second MHA), should also be masked too.

Encoder-only Version

Despite how interesting and useful the transformer architecture is, it has some (tractable) limitations. One is that it was designed for very specific purposes, with language translation. The other one is that the attention mechanism brutally elevates the number of parameters, making the forward pass runtime still very quick but significantly longer than the MLPs' one.

However, the good feature of being a very compartmentalized and flexible architecture is the key to solve these issues in certain measure. In this case, using the actions as an input of the model might be useful, but it should not be necessary, and not even close to crucial, for good performance. This is why a simpler form of the transformer will be used, the encoder-only transformer.

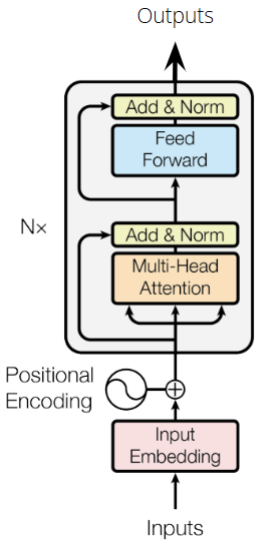

Figure 5.9: Encoder-only transformer version.

[5]Masking is cancelling a few values of the mechanism to avoid attention towards future outcomes (unrealistic).

As seen in Figure 5.9, this architecture is just a cut-off fraction of the original one. Instead of having en encoder-decoder structure, the so-called encoder (which does no longer have the encoding role), performs all the necessary self-attention MHA computations and uses the feedforward that follows it to learn better some dependencies.

In this project, some encoder-only models reached the number of ≈20 million parameters, which significantly increase with the full architecture. All the inner operations these perform add up when training the models for long periods, and using this format indubitably reduces the time consumption.

5.2.2 Discretization

At a certain point of the project, the transformed-based architectures used were performing considerably worse than the MLPs. Surprised by this aspect, a second look into the architecture was done.

One strong suspicion about what could be causing the difficulty during learning was the use of the positional encoding. Transformers have been widely used in NLP practices. However, the NLP formulation is different to this one in the sense that they usually include a vocabulary of tokens. These tokens represent the inputs and there are a finite amount of them, ≈ 30k tokens in usual Large Language Models (LLMs). So, their embedded space has a certain number of arrays to understand.

However, this formulation relies on the continuous nature of the states and actions to ensure precision. So, what could be wrong was that having such a dense embedded state and action spaces would lead to confusion within the model's input interpretation. Maybe, the PE would be leading certain state so close to others that the positional nuances of it would not help sequential positioning recognition but induce state confusion. Meaning, maybe the model would be understanding states differently to what the user gave it.

Two solution approaches were considered:

- Discrete-State Transformer
- Discretized-Input Transformer

The Discrete-State Transformer consisted on giving up the continuous values and focus on a discrete state and action space with certain predefined precision. The issue with this, and also one of the reasons go bid for the continuous values at the beginning of the project, is that the state space should be huge to ensure a relatively acceptable precision.

This method was tried but it was clearly unconceivable. Even an unacceptable resolution of the state space led to an embedding class with thousands of millions of parameters, significantly bigger that the encoder and decoders themselves. This was impractical and hence disapproved.

On the other hand, the Discretized-Input Transformer would consist on an extremely simple procedure. Instead of giving floating-point precision to the PE, the input values would be discretized to a custom resolution (i.e. 0.001, which is more than acceptable). This would still ensure high precision because the actions that are directly fed to the agent are still floating-point precise, which is great. The only noticeable effect for the model

would be to make the augmented state space less dense and leaving more space for the nuances of the PE values. This would avoid confusion among the embeddings.

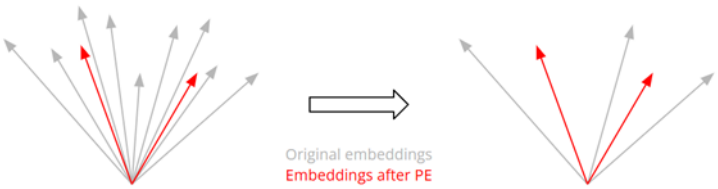

Figure 5.10: Reduction of the augmented state space density with discretization.

This tiny change in the implementation helped, indeed. The transformer-based architectures started learning properly and clearly outperformed the best MLP runs, by quite far, after this. Further comments will be made on Chapter 6.

5.3 Coding Overview

All the coding of the project was made with Python. This choice was made by the familiarity with the language, as well as the good set of resources it has. Several frameworks or libraries have been made for the ML/RL side of things. The most famous ones are, by the time this is written, *Tensorflow* and *PyTorch*. Given null experience with these before the project, a small research on the topic led the choice towards PyTorch given how convenient it is for more modern Deep Learning features.

Also, the CUDA platform from NVIDIA corporation was used to accelerate the training procedures of the models, using their GPU-adapted high-performance computing drivers and libraries. This was successfully implemented and the most powerful GPU used in the project (key for several long trainings) was an NVIDIA Quadro RTX 4000.

All the coding needed for this project is contained, except from the STK12 software mentioned in Chapter 4, in three main repositories publicly available in GitHub.

- **earth-gym**: environment, called Earth Gym, where the agent collects experiences.
- **sac-eos**: SAC algorithm to create and train the model.
- **ppo-eos**: PPO algorithm to create and train the model.

5.3.1 earth-gym

Earth Gym is a self-made code dedicated only to be the environment of the RL system. Its purpose is to receive the number of commands that make an agent initiated with the configuration file be moved around the world. To ensure compartmentalization, this environment is isolated from the algorithms executors. It stands by itself as a separate code execution and communicates with the other services as a server host via HTTP port connections (locally).

This means that Earth Gym acts like a listener to the demands of the algorithm and returns them to the client as soon as the task is finished. Them, going back to the listening status again.

The code always must be initialized first, since it creates the socket connection. Then, the two commands it understands are *get_next* and *shutdown*. The first tells the environment to move with a certain agent with certain action for a certain amount of time. The second command is simply the last one of the training and tells the code to shut down everything appropriately and stop running. Below lies an example of the JSON format for usual agent-environment interaction. The second example of the time increment is understood as a single number in seconds.

```
{
    "command": "get_next",
    "agent_id": 0,
    "action": {
        "d_pitch": 1,
        "d_roll": 2
    },
    "delta_time EXAMPLE 1": {
        "seconds": 1,
        "minutes": 2,
        "hours": 3,
        "days": 4,
        "months": 5,
        "years": 6
    },
    "delta_time EXAMPLE 2": 12.3456
}
```

The organization of the code is entirely an Oobject-Oriented Programming (OOP) structure with two main classes called *Gym* and *STKEnvironment* that are built once and contain the major bulk of main computations. However, in their classes they delegate plenty of tasks to additional classes that are specialized in certain aspects of the environment. For instance, there is the *DateManager*, which is in charge of everything related with the date and time tracking. Just the same way, one can imagine the tasks of the *AttitudeManager*, *SensorManagner*, *TargetManager*, *FeatureManager*[6] and *GridManager*. The reward calculations take place in the *Rewarder* class.

A great component of this code's development was the necessity to reduce the computational costs at the lowest level possible. Some RL libraries are highly optimized for this task, just like *Gymnasium* from OpenAI.[24]

However, this environment does not have the same number of development hours behind and is limited by the runtime of its API calls to the Ansys STK12 software. This has brought several issues with too big runtime. For that reason, a complete profiling has been made many times during the code development to see which calls and calculations took the most time. Then, these were built again with improved efficiency. For instance, calculating the detic coordinates of a satellite, in whichever orbit it is, is extremely time consuming. For this reason, a Piecewise Cubic Hermite Interpolating Polynomial (PCHIP) interpolation was added, so that it could

[6]This name is not that intuitive, but it tracks the diverse state and action variables of the agent.

extrapolate to the next detic value of interest. However, due to using extrapolation and not interpolation, the precision negatively compounds. Under this condition, there is a configuration feature called *LLA_step_gap* that tells the environment after how many steps it should make an STK12 call again instead of extrapolating.

In addition to these performance capabilities, a mode called *deep_training* was created. If this is not enabled, performance decreases so that the STK Scenario created can later be rendered in the GUI (on a Windows OS). This is the case of testing and visualizing policies. If the parameter is activated, the code gets rid of a big amount of data during the process that makes it impossible to visualize afterwards, but ensures the process is a lot quicker.

It must be said that this code is highly generalized. It can accept a numerous amount of configurations on the state space of the agents and a couple different variations of their actions. Also, as any agents as needed can be run. This increases the computational cost of the code linearly, which is not critical but something to consider, and the performance of this multi-agent system has also been optimized, getting rid of the data not necessary anymore at every point, and just keeping the vital.

5.3.2 sac-eos

The Soft Actor-Critic algorithm is fully contained in this repository, and accesses Earth Gym via the *Client* class, that gets the host and port as argument and connects to the existing server the environment created.

Below lies a fragment of the configuration file that is required to run this code, focusing on the algorithm-related configurations and its hyperparameters.

```
"Soft Actor-Critic": {
    "algo_version": "Original",
    "replay_buffer_size": 1000000,
    "minimum_samples": 1000,
    "batch_size": 64,
    "environment_steps": 5,
    "gradient_steps": 1,
    "reward_scale": 0.1,
    "critics_atoms": 1,
    "truncated_atoms": 1,
    "lr_v": 1e-4,
    "lr_q": 1e-4,
    "lr_pi": 1e-5,
    "smooth_coeff": 1e-4,
    "discount": 0.4,
    "temperature": 1.0
}
```

First of all, there exists an *algo_version* parameter because there are a couple variants of it that were done after the original publication. OpenAI presented a version with small changes in the computations. Although implemented, the original version has been the one used the most.[25]

Then, the rest are mainly hyperparameters of the algorithm presented in Section 5.1.3, with the exception of *critic_atoms* and *truncated_atoms*. This is an extended feature added later on due to the necessity to counteract overestimation bias. This will be further explained on Chapter 6. In a few words, the method included the Q-networks to have several so-called atoms and perform a truncation on their values.[26].

The algorithm was entirely made from scratch, relying on the original explanations from the paper presented in Section 5.1.3, but also with another one that treats this algorithm and some variants that are included in a second paper from Haarnoja et al., where they test the influence of slight changes on the algorithm and use it for certain applications.[27]

Finally, the code also accepts a multi-agent system. This means that several agents can be trained in this framework. The constraint is that these must share the same brain. Meaning, this was made for the centralized formulation. Hence, a separate tracking of the agents' states and actions is taken, but only one policy is trained. If training one agent only, the overall policy can be fit to be orbit-specific. This allows one to skip the orbit variables in the state. However, in the multi-agent case, these features are needed to let the model differ one from another.

5.3.3 ppo-eos

Just as with the previous algorithm, this is the repository containing the Proximal Policy Optimization. Its connection with Earth Gym work exactly the same as seen before.

The code snippet below is dedicated to a fragment of the configuration file, where the general algorithm parameters are contained.

```
"Proximal Policy Optimization": {
    "learn_steps": 1e6,
    "test_steps": 3e3,
    "trajectory_len": 120,
    "horizon": 1024,
    "minibatch_size": 64,
    "optim_steps": 10,
    "max_grad_norm": 1.0,
    "clip_epsilon": 0.2,
    "discount": 0.99,
    "gae_lambda": 0.95,
    "v_loss_coef": 1.0,
    "entropy_coef": 0.01,
    "lr": 3e-6,
    "lr_schedule": true,
    "lr_min": 0.0
}
```

A difference between the workflow of SAC and PPO is that SAC keeps running until the Earth Gym virtual time ended. That ending triggered a response that the run was done and it could no longer continue. However, this version sets a certain number of learning and testing steps. These steps, in the training side, are separated

into *horizon* blocks. Whenever the total steps are done or the virtual time is past (whatever comes first), the code stops and proceeds to shut down.

Also, differently to the SAC, this code's algorithm is guided by the TorchRL library from Python, a distinct but from the same developer as PyTorch. This is the one specifically dedicated to RL implementations, and there exist examples on using a more optimized version of the PPO algorithm. This one uses a specific class called *ClipPPOLoss*, which computes the needed loss values after getting the right set of data. Also another module is used to compute the advantages of a trajectory as quick as possible.[28]

Hence, the algorithm has not been entirely self-developed, since the use of optimized classes ensure more secure calculations and higher performance.

Another addition to the code is the scheduled learning rate. With it, one can set whether to use and to which minimum have a cosine annealing on the learning rate so that it favors convergence of the policy and allows it to get to better local or global minima.

Finally, for calendar and time available, this code is not generalized for multi-agent training. The setup is made to only accept one agent at a time. This means it is not completely developed for the overall purpose, but still entirely functional for the trainings of single-agent systems.

5.3.4 Models

Both *sac-eos* and *ppo-eos* share a very similar *models.py* file where the agent's policy architecture possibilities are defined. Only details that depend on the implementation of the algorithm differ one from another.

The models implemented are the following:

- Simple MLP[7]: simple multi-layer perceptron with three hidden layers and $tanh()$ activation functions in between. Prepared to be either a policy network (stochastic) or a value function.
- MLP: more customizable multi-layer perceptron with ReLU (Rectified Linear Unit) activation function and normalization layer to avoid values explosion. Same capacities as the simple MLP.
- Transformer: the model itself includes the *Transformer* class from PyTorch, that resembles the encoder-decoder structure of a transformer. Then, self-made classes called *FloatEmbedder* to embed the state and action data into the desired dimensions, *PositionalEncoder* to apply the positional encoding to the input embeddings as done in the original paper of *"Attention Is All You Need"* with sine superposition, *SegmentPositionalEncoder* to apply a different positional encoding explained in Chapter 6 and *Projector* to take the high-dimensional arrays form the transformer and bring them to the desired shape. Its values can be discretized as shown in Section 5.2.2. Adapted to both policy and value function.
- Transformer encoder: similar structure as the transformer, but using the *TransformerEncoderLayer* and *TransformerEncoder* classes from PyTorch to build the core of the encoder architecture. Same capabilities as the transformer.

[7]Simple MLP was actually not implemented in the *sac-eos*. It was introduced in the PPO development due to its simplicity of configuration and how adequate it was for smaller general cases.

- Discrete-state transformer encoder: identical architecture to the transformer encoder but with the different data acquisition and emission because of the discrete states and actions and self-built vocabulary.

In the case of the transformer and transformer encoder, it has been ensured that all Multi-Head Attention (MHA) layers are performed with masking, necessary for the temporal dependencies of the problem. Also, notice that the simple MLP and the MLP can not receive data in the order of a sequence. For that reason, the algorithms were properly made so that if the model to receive the data is non-transformer-like architecture, stretch the input to be the concatenated set of items in the sequence. Hence, although the MLPs do not consider temporal dependencies, they could in some way be learned depending on the position of the relatively large state space the variables were in. Of course, this was performed in a consistent and flexible manner than ensure secure data processing for the models and did not impede learning.

Finally, it is worth mentioning that there exist customizable features regarding performance details of the architectures, as can be the initialization methods of the linear layers, their dropout to prevent overfitting, activation functions, etc. Below lies an example of a configuration file code snippet from the *ppo-eos* repository where the models' architectures definition is exposed.

```
"Architectures": {
    "policy_arch": "TransformerEncoder",
    "value_fn_arch": "TransformerEncoder",
    "archs_available": [
        {
            "name": "Transformer",
            "d_model": 512,
            "nhead": 8,
            "num_encoder_layers": 1,
            "num_decoder_layers": 1,
            "dim_feedforward": 2048,
            "embed_dropout": 0.1,
            "pos_dropout": 0.1,
            "transformer_dropout": 0.1,
            "position_encoding": "segment",
            "activation": "relu",
            "batch_first": true,
            "kaiming_init": false,
            "discretized": false,
            "tolerance": 0.01
        },
        {
            "name": "TransformerEncoder",
            "d_model": 512,
            "nhead": 8,
            "num_encoder_layers": 6,
            "dim_feedforward": 2048,
            "embed_dropout": 0.1,
```

```
            "pos_dropout": 0.1,
            "encoder_dropout": 0.1,
            "position_encoding": "sine",
            "activation": "relu",
            "batch_first": true,
            "kaiming_init": false,
            "discretized": true,
            "tolerance": 0.01
        },
        {
            "name": "DiscreteStateTransformerEncoder",
            "d_model": 512,
            "nhead": 8,
            "num_encoder_layers": 1,
            "dim_feedforward": 2048,
            "embed_dropout": 0.1,
            "pos_dropout": 0.1,
            "encoder_dropout": 0.1,
            "intervals": 5,
            "position_encoding": "sine",
            "activation": "relu",
            "batch_first": true,
            "kaiming_init": false
        },
        {
            "name": "MLP",
            "hidden_layers": [128, 128],
            "dropout": 0.1
        },
        {
            "name": "SimpleMLP",
            "n_hidden": 256
        }
    ]
}
```

Chapter 6

Results

As the Gantt diagram from Figure 4.1 shows in Chapter 4, the two algorithms were not built in parallel. SAC was developed first, and PPO went after that. Testing held the same format too. First, a total of 31 deep training took place first with the SAC algorithm and 34 others went after with PPO, in a total of 65 deep trainings.

Each of the sets of deep trainings has faced its own challenges and during different timelines, so this is why seems convenient to explain them separately.

6.1 SAC Deep Trainings

The algorithm in this case, as mentioned in Section 5.1.3, is off-policy. The way to begin the training is by doing a warm up, which is collecting a set of samples (a fraction of the total replay buffer size) to have a non-empty replay buffer and avoid overfitting on the first iterations. After that, the actual training algorithm starts.

What happened in the very first iterations was that the learning was, of course, was null when the warm up happened. But then, the rewards showed an incredibly stiff positive progression up to a certain value.

Figure 6.1 shows the clear increase in rewards. They start increasing at the 100,000th step. This step is the size of the replay buffer for that iteration and, at that point, the warm up was thought to be filling the entire replay buffer. This aspect was later corrected to be a smaller fraction given that it a too low-quality policy learning focus. Also, be mentioned that the original SAC paper recommends to use a buffer size of 1,000,000 samples, as general advice. This does not imply the size used was inappropriate, but it can point to a flaw in the strategy.

After that stiff growth, the curve no longer increases. However, this training is too short to get any conclusions on possible learning in more advanced stages.

Notice that rewards are negative due to the slewing constraint affecting the reward value. Although still observing events, the constraint takes over. Even though this condition, it is not inherently bad to count on negative rewards, as long as they improve towards positive values and they eventually take place.

In terms of the loss for Figure 6.1, no convergence is appreciated at all. To be fair, there exists a little divergence

in the critics' networks and a clear one in the policy's. Although policy loss gets to be very negative, these orders of magnitude might not be the most appropriate behaviour on seeking a slow convergence to the optimal policy.

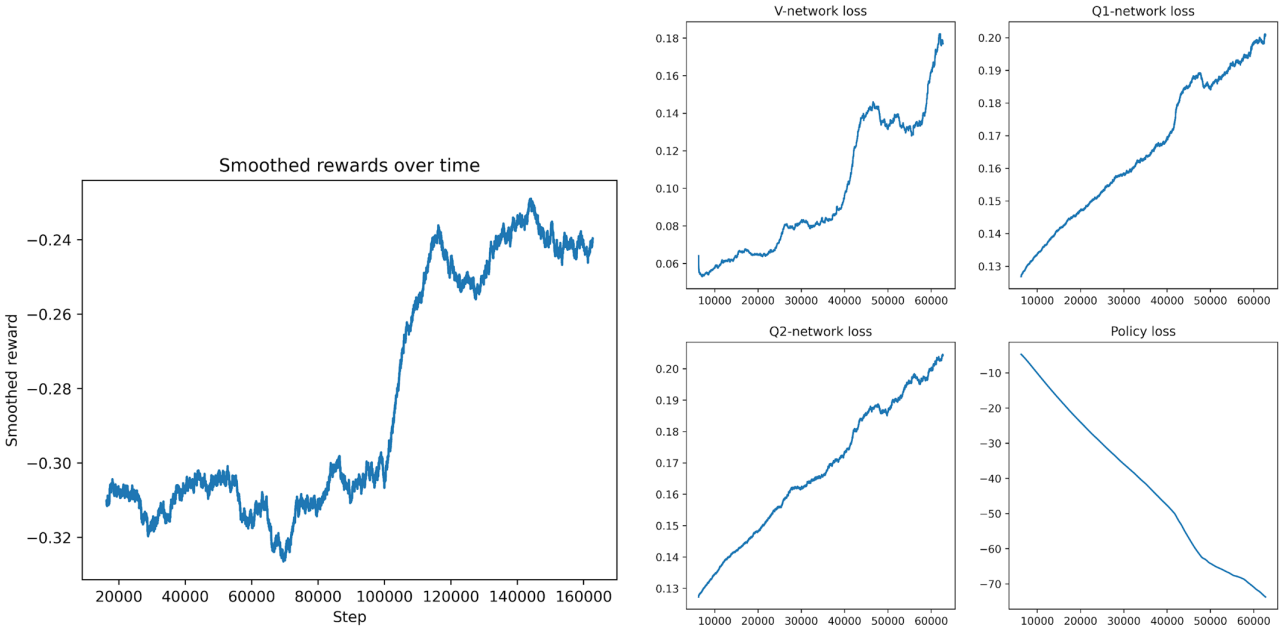

Figure 6.1: Deep training 1 (SAC; transformer; 100,000 buffer size).

In Figure 6.2, deep training 3 seems to follow the same pattern and confirms the behaviour of not increasing rewards after the initial growth happens. Further deep trainings with more steps corroborate this too. Also, despite losses still seem to diverge, a slight stabilization of the values appears, seemingly going through a sinusoidal convergence to very extreme values.

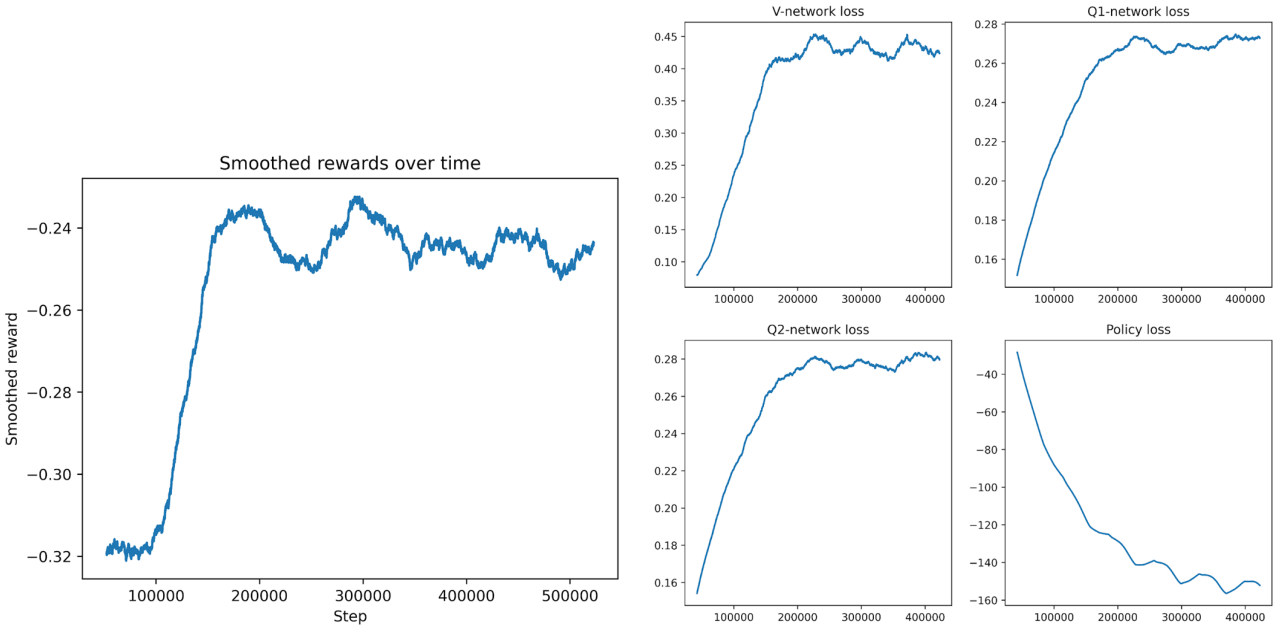

Figure 6.2: Deep training 3 (SAC; transformer; 100,000 buffer size).

From these, it was needed to check whether, for any reason, the best learning possible was achieved. Doing some GUI testing, where the policy is deterministic instead of stochastic (its performance should improve), these trained policies could not achieve a motion control better than a random policy at first sight. Analyzing the data from the runs, it was seen that the model could only learn to make smaller movements in pitch and roll, meaning that it only focused on the negative rewards the slew constraint was giving it, but not on observations.

Notice that, in both Figure 6.1 and Figure 6.2, the stabilization value of the rewards was at -0.24. All the trainings that were done with the same rewards metrics got to the same exact value (or close by). There seemed to be a limitation to learn further beyond.

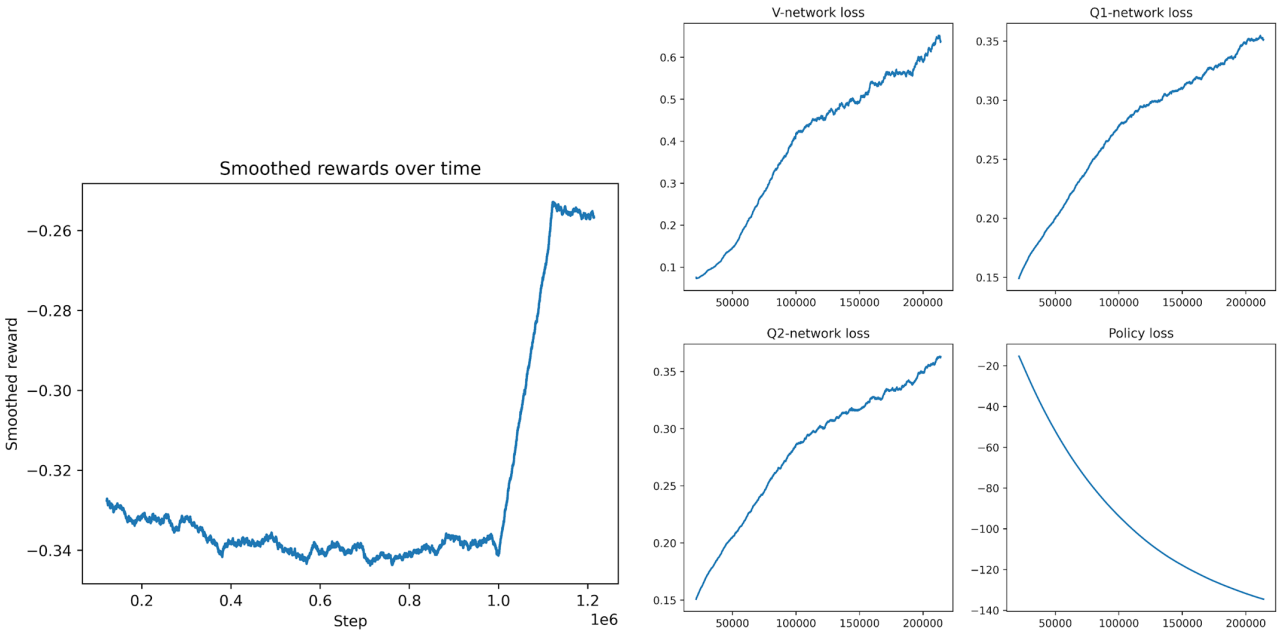

Figure 6.3: Deep training 7 (SAC; transformer; 1,000,000 buffer size).

For a training with bigger buffer size (a million samples), as seen in Figure 6.3, the profile of the rewards remains the same. Although it seems short, the training phase, lots of iterations were taken and even more optimizations. Longer trainings show that same pattern too. Whereas for the losses, these still diverge a lot in value, especially the policy one.

With all this confusion, a lot of hyperparameter tuning was done. From the learning rate to the environment and gradient steps repartition, nothing was able to improve this learning curve. MLPs happened to do the same but even with an inferior achievement with the rewards, where they stopped increasing at -0.45. The only thing that could affect the losses convergence was the discount factor. Previously, a factor of 0.99 was used. After using a smaller one, this divergence got less extreme. This was mathematically linked to the q-networks and v-networks values.

When using lower discount factors, i.e. 0.4, the bootstrapping[1] made less emphasis in future rewards and held lower values. This led to easier convergence of the actor and the critics (still with no improvement in rewards).

[1]Bootstrapping is the process of estimating a value or a Q-value with a network.

However, it was seen that, despite the value network converged to null loss eventually, the Q-networks did not. These always stopped at a value of 0.12 or so. It seems very low, but it was a consistent limit. When analyzing the training data, it was seen that bootstrapping was predicting much higher values than the estimated target they should match. Despite all the optimizations, these values could never match.

Once that, it was considered that an overestimation bias issue was happening. Given this, a deep dive was taken on literature about this effect. It is a well-known issue in RL, where sometimes it can be coped by the algorithm, but sometimes not. The action taken came from a publication by Kuznetsov et al., where they proposed a method to extend multiple Q-values. Instead of using many Q-networks (high computational cost), they use two that have lots of atoms. These atoms represent outputs of the network. Instead of 1, there can be 10, for instance. After that, a truncation is performed on the atoms based on the value they take. And, finally a mixture between the two sets of remaining atoms is taken to get a value generally lower than expected.[26]

Although functional, this method did not seem to improve anymore the already seen behaviour. After a deep look on hyperparameters again and more ways to counteract the overestimation bias, which is still not sure this is the cause of the poor learning, the testing with this algorithm came to an end.

It was later on than, while studying about RL, a chapter of Sutton & Barto called *The Deadly Triad* was seen. This chapter explained that there are three features of RL that when they come together, they can weaken the learning convergence so much that it is no longer possible.[29]

These three features are:

- Function approximation
- Bootstrapping
- Off-policy training

When someone takes a quick look, realizes that SAC checks all the three boxes. Hence, it is susceptible of being within the trap. However, some other algorithms are too. The truth is that SAC has been proven to work in many cases and, indeed, at a very high level. The deadly triad is just an orientation on how things could go wrong during training. So, all there is to say is that, maybe, the force of the deadly triad together with some features of the problem, which is not by any means simple, could lead to these negative results.

6.2 PPO Deep Trainings

Since PPO is an on-policy algorithm, as explained in Section 5.1.4, no warm up phase is required. The length of the training is fully focused on actually optimizing the policy. However, in *ppo-eos*, an additional feature was added. Every ten horizons, an extra one is purely dedicated to testing. This way, a small percentage of the training can help see the performance of the model with a deterministic policy.[2]

In the first one, deep training 32, there existed a clearly good learning profile, as can be seen in Figure 6.4.

[2]Testing plots are only available for the second half of the trainings.

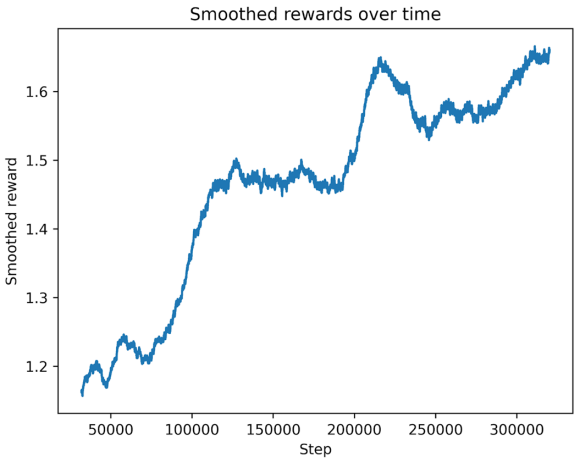

Figure 6.4: Deep training 32 (PPO; simple MLP; 512 horizon length).

Notice that the rewards differ a lot in value from the SAC case (even the initial ones). This is not only for the increase in performance, but due to different use of reward metrics, the coverage grid, etc. In further cases, rewards will be lower because of not using the coverage grid, which seemed to not be critical for learning.

Afterwards, two very long runs took place in parallel. Both with the exact same configuration, one used a simple MLP and the other one a transformer encoder. Figure 6.5 shows how the simple MLP (left picture) performed an amazing run and increase in performance, up to doubling the positive rewards it perceived. Testing showed this policy was actually improving the satellite's motion. Also, no slew constraint was applied to simplify the first trials of the learning process. This was also tried in SAC but without success.

The problem is that the right picture shows the transformer performance and clearly states that no reward improvement happened. This was a very curious outcome given that previous experience had shown this same model much better (within the limits of the poor SAC learning) than the MLP.

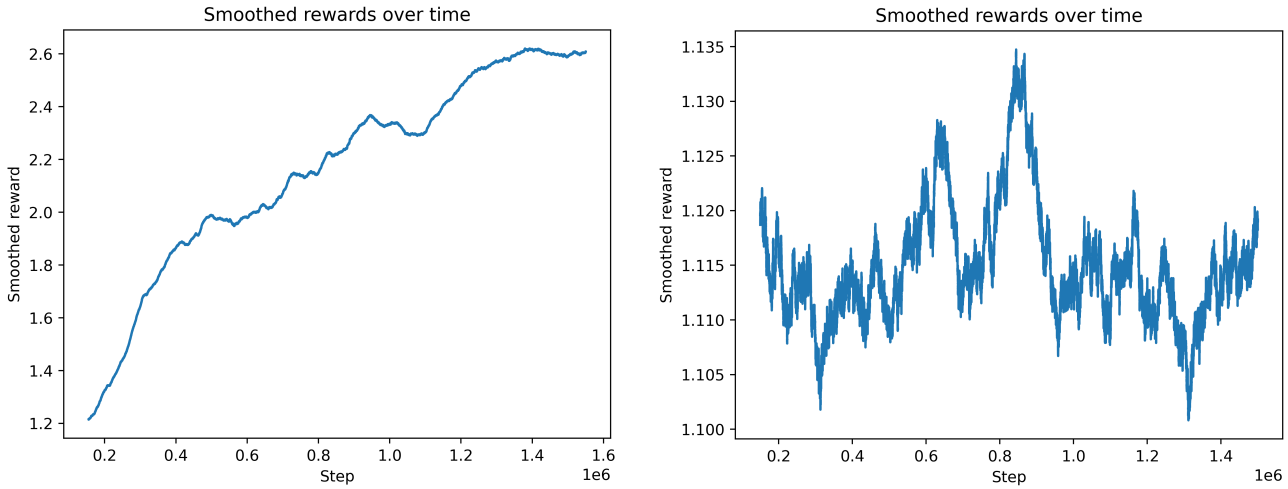

Figure 6.5: Deep training 33 (PPO; simple MLP; 512 horizon length) in the left picture and deep training 34 (PPO; transformer encoder; 512 horizon length) in the right picture.

After that, the work focused on understanding why this difference in learning quality happened. Many hyperparameters fine-tuning took place, but nothing seemed to help a bit the learning capability of the transformer-based architectures.

With much work on this path, some doubts came around the role of the positional encoder. Due to the continuous nature of the problem, there was the hypothesis that the sine superposition values could be affecting or confusing the model. This was built on the idea that having continuous augmented state space inherently leads to making it very dense. Although $d_{model} = 512$ was used, floating-point precision is very low, so the space can be considered dense enough that the influence of the values of the PE actually confuse the agent because these can belong to other states. This confusion effect would make the model incapable of learning the actual functionality of the PE and incorrectly interpreting the augmented state it has been given.

Under this suspicion, another version of the PE was created. The *SegmentPositionalEncoder* class made a different version of this practice. It received the embedded state and action arrays with a $\approx$ 500-dimension array and created a $\approx$ 12-dimension array containing the positional encoding information. This was tried but rapidly decommissioned, given the lack of improvement.

Nevertheless, it was likely that the problem still lied in the PE. Maybe this approach was not correct, but the solution had to come in a similar manner. For that reason, the discrete-state and discretized models came to light, as explained in Section 5.2.2. The *DiscreteStateTransformerEncoder* was also rapidly discarded given the impractical number of parameters required for a good resolution (in the order of billions). However, the simple change of discretizing the values coming into the embedding class changed the paradigma.

The transformer-based model, which still had an acceptable resolution on the input, i.e. 0.01 or 0.001, and floating-point precision in the output, started to perform better. Indeed, they quickly began performing a lot better than the simple MLP and MLP models.

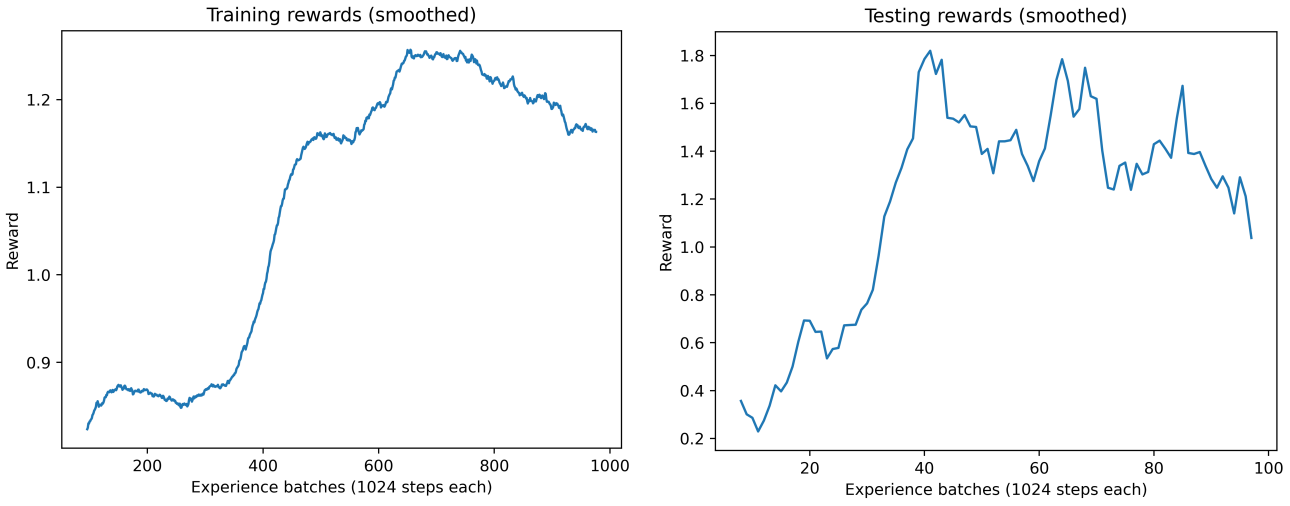

Figure 6.6: Deep training 53 (PPO; transformer encoder; 1,024 horizon length).

The improvement shown in Figure 6.6 is quite clear. The final testing of the PPO training rewarded the agent an average of 2.62 during 3000 steps. However, when doing visualization testing with the GUI, the policy

seemed to be extremely greedy. Careless about the minimal slew constraint added, it started performing a rather extensive scanning of everything around. Taking drastic movement in roll and very little in pitch, it was seeing everything on Earth but with a very inefficient policy energy-wise. The way to proceed was now focusing on reward metrics fine-tuning in order to find the best configuration possible to get the desired outcome.

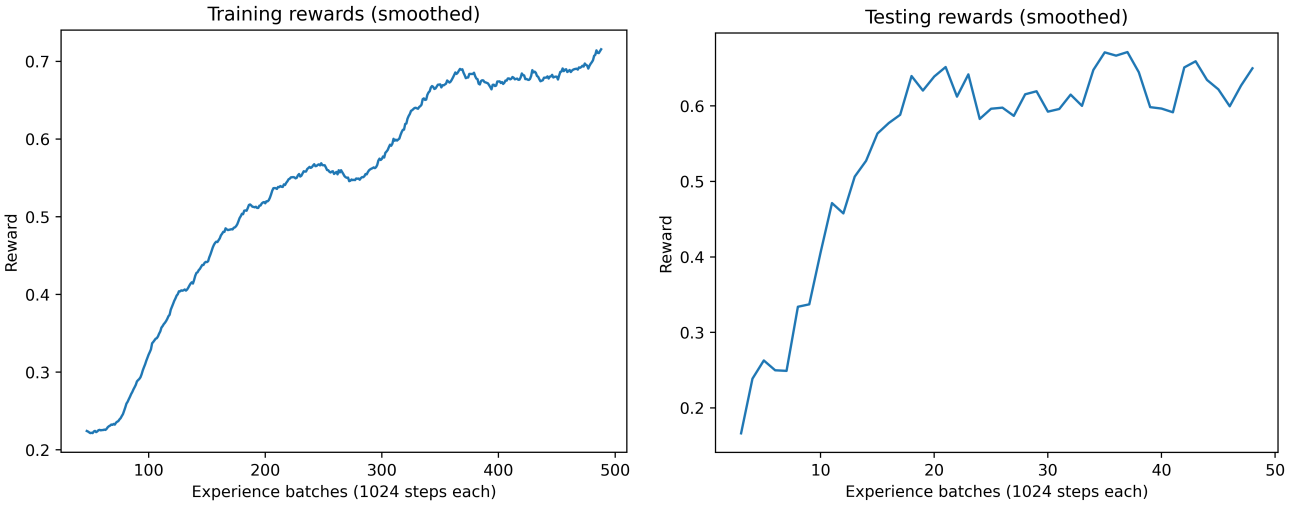

Figure 6.7: Deep training 59 (PPO; transformer encoder; 1,024 horizon length).

For deep training 59, the slew constraint was incremented and the minimum amount of time required to observe a target was increased. This way, let the agent try to take the slew constraint more into account and avoid extensive scanning, where targets are seen for a short interval only, without actually focusing to them.

The result was also good in terms of rewards, as seen in Figure 6.7. Nevertheless, what the policy was actually doing this time was focusing too much on looking with a fixed orientation towards Earth's surface. It was looking around a zone where the area of the sensor's FoV was the highest. The satellite ensured observations were long enough. The result was not satisfactory since it was not really focusing enough on the state space.

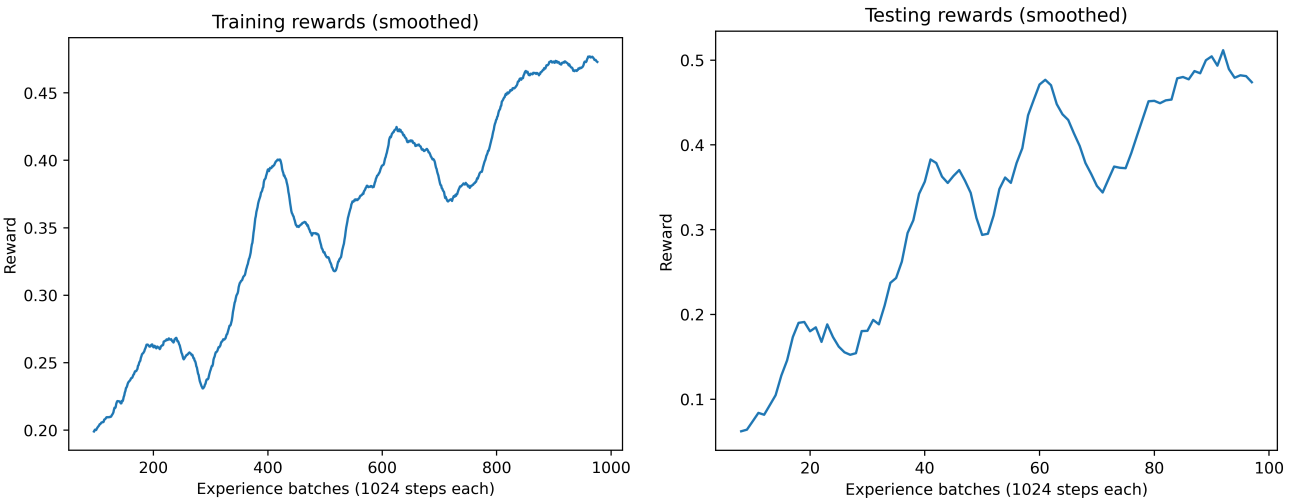

Figure 6.8: Deep training 61 (PPO; transformer encoder; 1,024 horizon length).

Finally, Figure 6.8 corresponds to deep training 61, where the learning curve is impressive. Not only on the rewards increase, but also on the way the policy kept falling into local minima, shown as the intermediate peaks, but then escaping it at the cost of considerably lower reward to get somewhere better. The training here has not converged yet, but the tendency looks promising.

Its actual behaviour was a lot more satisfactory than all the other runs. This time, the model was actually looking towards the targets drawn on Earth's surface. It was not following a greedy policy, since movement were taken accordingly to the events around. To achieve this, a middle point between the minimum observation time of the 53rd and the 59th was used. A similar adjustment was done with the slew constraint.

With respect to the losses of these runs, they do not show anything concerning. Below, Figure 6.9 shows the objective, critic and entropy losses of the last run mentioned.

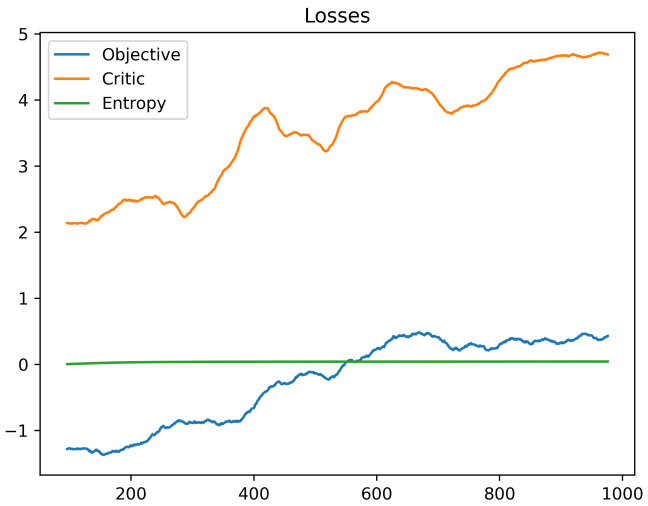

Figure 6.9: Losses of deep training 61 (PPO; transformer encoder; 1,024 horizon length).

Notice how the entropy loss is very small compared to the rest. It seems like null all the time, but it is not. This one runs in a different order of magnitude, which does not mean the hyperparameters are wrongly tuned. Its effect should not be the same as the actual objective loss of the policy. In general, as said, nothing to worry about is seen. There is not a specific convergence towards a value, but the numbers seem reasonable.

Finally, mention that all the notes taken within the period of PPO testing also improved the knowledge on what could go wrong with SAC. Some runs were done again with SAC, with many changes acquired in the proceedings of the PPO testing, but none of them resulted better than what was obtained before. And, about the PPO final runs, these were dedicated to improve the policy achieved in deep training 61. A few runs started from that policy and continued for longer periods. The improvement in performance thereafter was not huge, but noticeable. Apart from that, no multi-agent trainings could be done.

Chapter 7

Sustainability Report

The set of evidences about climate change point to an unstoppable global warming by the next few years that would extend just as long as the human species continues squeezing the natural resources. It is not a partial duty for humanity to recycle and participate in a few initiatives towards uninsured Earth's healthcare. It is a personal responsibility to provide the best sustainability outcome in any purpose one has.

For that, satellite constellations are not an exception. For that, the diverse impacts this work could have in the world and the ethical concerns about will be treated in this chapter.

7.1 Sustainability Matrix

The three main pillars of the human's impacts on the rest of the world and Earth will be considered here. The purpose is to identify the influence this work could have on each field, as well as identify the weaknesses and strength of future outcomes provoked by it. For every pillar, there will also be an analysis of the impact during project development and execution, as well as the possible risks and limitations.

7.1.1 Environmental Impact

On the environmental side, do not be confused by the fact that satellites belong to space. In fact, they don't, Earth's atmosphere limit is considered to be Kármán's line, which is 100km above the surface. Most of the Earth observing satellites are within this range.

This means that, just as cars and planes do, the satellites that use chemical propulsion to maneuver produce atmospheric emissions. Also, not only for Earth observing, but for all space missions in general, to put a body into orbit costs an extremely high amount of fuel (usually kerosene). So the more there are, the more emissions will be made.

In addition, satellites are not free of collision with small Earth-orbiting objects or other artificial satellites. Especially after they run out of fuel, they are likely to hit or be hit and release parts of their structure which will become space debris. Several studies have come up to track the thousands to millions of space debris

objects around the planet. However, and despite the diverse techniques to clean them, there will always be a high number of them.

Given these problems, the easy conclusion is to make the least satellites possible. However, since this goes quite against space exploration, which in certain measure humanity still wants to progress in, another solution must be met. This comes across the fact of making the missions more reliable, robust and efficient, in order to avoid redundant ones that would increment the number of issues already mentioned. By making an algorithm like this, the intention is to improve the state of the art of satellite maneuvering and provide with a less time-consuming decision-maker for attitude orientation.

Satellite computers are truly small compared to what a usual user has with a regular computer. All of this is to avoid weight and power consumption. Using an AI agent to define the measure in which a satellite moves takes a few hundredths of a second per decision. Compared to minute-level order of magnitude in decision-making for a dynamic programming algorithm, the energy consumption is incomparable, and could make a difference on the amount of mass required to be brought to space and the fuel needed for it.

Aside form that, an algorithm that focuses a lot more on the attitude movement control rather than the task scheduling is a lot more flexible to learn motion patterns than entail savings in the propulsion consumption.

Finally, this is not only work towards developing a more efficient satellite management. The topic of the thesis focuses on scientific missions, which are the leading tool the world has to keep getting better at avoiding global warming. Indeed, Earth Science mission are what allow us to know the increase in ocean height from the last thirty years or more, as seen in Chapter 1 and Figure 1.4. And, as well as with scientific studies, the detection of natural disasters is a job these instruments were made for. Although they are already capable of detecting, measuring and monitoring certain long-lasting disasters, state of the art satellites are not yet reactive enough to perceive some others like flash floods of tornadoes. The improvements on the reactivity on the planning would allow more imaging and data acquisition from these hazards, ensuring better capabilities in avoiding them or the dangers they pose on humans.

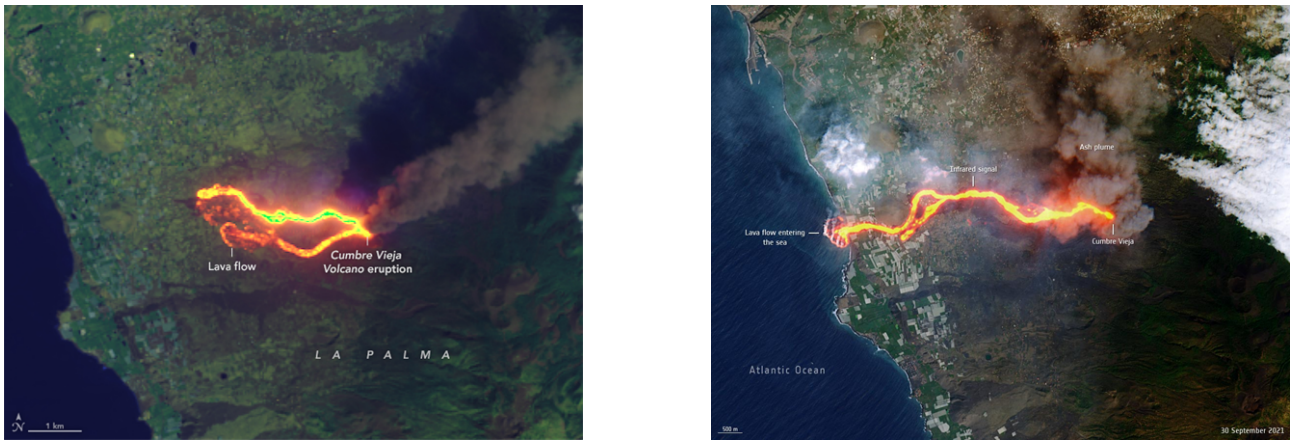

Figure 7.1: Evolution of the volcanic eruption in La Palma in 2021.

In Figure 7.1, the left picture shows the early stages of the lava propagation, which, thanks to the satellite and the surface mapping could help predict the zones affected in future times (right picture). This could lead to

public safety and better damage prevention with more time for inhabitants to evacuate the highest amount of personal belongings as possible.

Project Development

During the development of the project, the influence on the environment is hardly noticeable. The main impact is the use of electricity to run the computer hours of work and the many hours of running trainings. Since it is purely a software development work, no extra material, procedures or third-party people were implicated.

Project Execution

Once it is up and running, this project simply works as a computer program on-board of the satellites using it. Directly, there is no consumption associated with it other than a small-scale energy consumption. Mainly, the cost would come from using long hours of computer and highly optimized GPU systems to train the model, which leads to non-negligible amounts of electricity usage. Another thing to consider in this pillar is the possible impact of testing and validating the trained agent in real life. It would be convenient to use drone prototypes that could simulate the behaviour of a satellite and see the real-time response of running with such a policy. This could take some extra consumption in material, but given the nature of nowadays drones, fuel consumption should be coming only from testing transportation and logistics.

The overall implementation, in the case of real improvement compared to state of the art algorithms for satellite maneuvering, should make a better footprint for the Earth Science missions in general. And, after finishing the useful life of a satellite, no waste is expected to take place directly from this project.

Risks and Limitations

The real innovation of this project is how different the formulation is compared to other approaches of satellite scheduling where the objective is not directly the movement of the satellite but the tasks planned. Given this new way to do the job, there would be required a few validating trials to ensure the model works correctly after long runs. This way, it would be catastrophic if a few weeks or months after launch the model did not perform accordingly due to some unexpected condition. This would complicated the mission and require another mission to do the job, which would double its footprint.

7.1.2 Economic Impact

The economic impact can be divided into two branches. From one end, see how the implementation of this work can influence the budget of the missions it should be involved in. And, form the other end, how many can be saved due to the increase in performance of such missions.

On the first end, and already mentioned before, the improved model could lead to better energy efficiency. This seems unnoticeable economically speaking, but bringing a few less weight in batteries to orbit makes a big deal in terms of costs.

On the second end, the quick detection of natural disasters, as explained in the previous pillar, would lead to

damage prevention. This is, of course, directly related to the economic harm of these hazards. For instance in the example shown in Figure 7.1, with the volcanic eruption in La Palma, total economic losses were in the order of hundreds of millions in euros. Non-negligible savings could take place from slight improvements in prevention systems' performance.

Also, this attitude orientation model might be useful in communication systems too. The learning process could be leveraged with antenna orientation and provide better service and connectivity to the public. This could build more efficient constellations that would save money on deploying costs and user subscriptions.

Budget Analysis

A budget analysis of the initial expected expenses is shown below. Here, one will consider the amount of work needed to train the central decision-making agent of a constellation of satellites and the cost of implementation and maintenance during the first year of operations.

Table 7.1: Project development budget analysis.

Category	Item Description	Estimated Cost (EUR)
Software & Licensing	Ansys STK12 Enterprise + Astrogator (1-year license)	25000
	Python/C++ development environment (open-source)	0
	Additional tools, plotting libraries, etc.	465
Hardware	Development workstation CPU + RTX GPU + SSDs)	7200
	Backup workstation or laptop	1800
	Networking and storage infrastructure (NAS, backups)	1350
Personnel (6 months)	1 Aerospace engineer (mission dynamics) – 6,000€/mo	36000
	1 Software engineer (controls, integration) – 5,000€/mo	30000
	0.5 QA/test engineer – 3,000€/mo	9000
Validation w/ Drones	Drone rental or purchase (2 mid-range UAVs)	8000
	Ground control setup, telemetry tools	2400
	Testing field permits, logistics	1600
	Insurance and safety compliance	1600
Miscellaneous	Travel for testing	13840
	Office costs, business subscriptions	1395
Total Cost		**139650**

Table 7.2: Operational costs during the first year.

Category	Item Description	Estimated Cost (EUR)
Personnel	0.5 Aerospace engineer (maintenance) – 6000€/mo	36000
	0.5 Support staff (monitoring, ops interface) – 2,000€/mo	12000
Software Licenses	STK annual license renewal (Enterprise + Astrogator)	25000
Compute Costs	Cloud server (e.g., telemetry system)	5400
	Backup and storage infrastructure	1800
Validation / Updates	Occasional drone or hardware-in-the-loop tests	4000
	Bug fixes, minor upgrades	2400
Miscellaneous	Trainings, workshops	8000
Total Cost		**94600**

As shown in Table 7.1 and Table 7.2, the respective costs for project development and first-year operations would be **139650€** and **94600€**. See how a good part of both amounts is destined towards the licensing for the Ansys STK12 software. Using a self-made orbit propagator and data collector would help in the task of making this project a lot more competitive economically speaking. Also, it is a must to be using open-source code, since the saving with that are notorious. Also notice how, as could be expected, the most part of the expenses is destined towards human salary.

Project Development

As seen in Table 7.1, the economic impact would be basically a commercial consumption of computer services, software, material and employment.

Some cost-cutting decisions have allowed savings in the budget, just like the use of open-source code environment, because it can still be one hundred percent capable of doing the same task. However, there is a limit were, given the absence of a self-made orbit propagator, there are few options other than purchasing the Ansys STK12 license.

Project Execution

Just as in the development, Table 7.2 shows that most of the economic expense of the operation is salary requirements. This first year would probably be the most expensive compared to the ones after. Although expected issues could arise, the scaling of the project could lead to cost-cutting in terms of reusing the validation drones every other year, as well as investing in an engineer to create the self-made orbit propagator software to save even more in the long term.

Risks and Limitations

The prices mentioned in the tables are nothing but an orientation of the expected costs. These could be deviated up to a 30% due to possible unforeseen issues with the development, the satellites missions involved, the health of an employee or the material.

A realistic scenario would be that the first validation tests taken were done when the algorithm is not fully robust. This could expose the expensive drones to crashing and, hence, additional spending on them.

7.1.3 Social Impact

The direct social impact of this project is barely existent. Being an AI model stored in a computer inside a satellite that does not interact with humans high in the atmosphere makes it difficult to be impactful.

However, of course there are hidden influences this work can make on humans. First and foremost, the entire set of capabilities to detect and prevent natural damage. Just as said with the environmental and economic impacts, decreasing the material damage of the public is more than anything a social responsibility. Protecting the integrity of the personal belongings of people can be very valuable for society, since some goods lost on

urban fires and floods are priceless. A house is worth its price for anyone in the world except for the people living there, the worth of which is multiple times the price.

The data collection for scientific and political purposes is leveraged in all fields and pillars possible. Just as much as science benefits from that, people benefit from science. This AI model would enter a self-contained circular feeding process where the most fundamental purposes are for humanitarian initiatives. Not only raw data but satellite imagery is consistently used all over the globe with maps services.

Even more, in the event of applying this planning method for satellite communications and connectivity provision, barriers to technology would weaken. Underdeveloped countries could be closer to global internet affordability, just as adventurous user could have internet in the Amazon rainforest, the Sahara desert, etc.

Project Development

During the development phase, there is hardly an impact on social matters. Employability of an aerospace engineer would be the main one. No illegal practices are planned on this process, and no harm could be made on anyone if all validating and testing procedures were duly secured. Drones out of control can cause damage, but within the correct safety measures, these dangers can be decreased to inexistent.

Project Execution

The benefit of the right implementation of this work is directed, in a hardly noticeable but present manner, to everyone trying to avoid the climatic disasters or with interests inside the Earth Science field. A few thousands of millions of people could take advantage of this at some point of their lives.

Also, everyone is affected equally without any dependence on their race, gender or national feeling. It is true that richer countries with more developed space agencies are the ones who benefit the most from this. But, since the cornerstone of the project is making these missions more efficient and capable, it makes a step forward towards affordability for less developed countries to have the missions' services too.

Risks and Limitations

The risks to be assumed are on the use of this technology for evil reasons. The satellite fleets of some countries are a great wealth for only a few nations. The reach is beyond most part of the rest of the world, and their influence is beating records as time goes by. If this extremely complex set of machinery pieces was to be taken away from the purposes they serve, issues could arise.

The real concern would be their use for warfare reasons. Maybe one country too powerful to hold their fleet would focus it towards self-interest. In addition, focused cyberattacks could turn down any minimal control of a space agency on their constellations, and maybe use that force against them. The highly advanced technology created with such work and decades of long satellite development could fall in the wrong hand if not enough protective measures are taken.

The most typical cases where satellites receive hostile attacks are:

- Jamming: practice to generate unauthorized transmission at the same frequency another official service is transmitting with. This can cause loss of information and is often used to evade tracking.[30]
- Spoofing: activity to imitate authorized transmission to generate a misleading outcome of communication. This can confuse and endanger aircraft. Also, this practice can focus attacks on Global Navigation Satellite Systems (GNSS) directly and make them nonfunctional.[30]

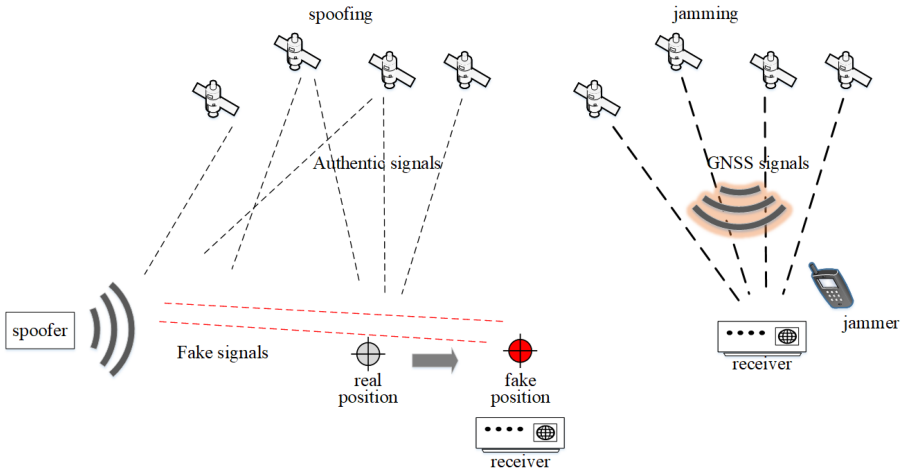

Figure 7.2: Spoofing and jamming techniques, schematic functionality.

7.2 Ethical Implications

When doing a project, an initiative or anything that comprises some action, devotion eats everything else. Sometimes it takes a little step back to understand why something is being done, and under which circumstances it still fulfills a real, worthy desire and need.

This project is not concern-free in the sense that it is not so evident that no harm can ever be caused with it. In Section 7.1.3, the risks of this technology were debated, and they consist an important question to be answered.

However, what and who is this done for? Actually, the real motivation behind this particular problem, to be fair, comes a long way back, and it comes across the purpose of contributing to the deep down desire fo humanity to invest in space technology. Not only for an exploration purpose, but for its influence on planet Earth too. The implications of climate change in our lives is heading humanity to an implacable future, where resources that are now given for granted will build recessions themselves.

The damage done is long unavoidable. But maybe, taking care of the Earth's state right now will extend its well-functioning for generations to come. And, sadly but true, the option to explore beyond the green planet is a need to be covered and let humanity be interplanetary species as soon as possible to evade extinction. This sounds catastrophic, but is a danger so big that even though all the years it may take to come it is a risk worth to be discussed.

In a nut shell, even though the world is replete of evil intentions that could potentially leverage the power of the great constellations against its purpose, the motivation behind is, from an objective standpoint, worth such

a risk. Because, the needs this project is tackling are on not on the name of one's entertainment, but on one's security. Which, by the way, is also curious and quite evident, that this need to monitor our planet is not only natural, but self-imposed with all the years of pollution and resource exploitation accumulated.

7.3 Relationship with the Sustainable Development Goals

Regarding the 17 points of the Sustainable Development Goals (SDG) approved by the United Nations (UN) for the 2030 agenda, this work fall inside a couple of them. This project's purpose is aligned with them.

13. **Climate Action**: this goal is the main one tackled by this thesis project. The climatic purpose on the applications of this AI-based satellite agent is present with the capabilities to help prevent and detect climate hazards in the form of natural disasters. In specific, the 13.b goal is dedicated to enhancing the make of technologies implementation of technologies that help managing climate affairs in underdeveloped countries. Given the orbital nature of satellites, this work can reach these kind of countries and down to the most remote places in the planet.

15. **Life on Land**: the work here also falls partially under the umbrella of this goal. Goal 15.4 specifically says to take action towards the protection of mountain systems and similarly with 15.5 but towards natural habitats. The link with this project is the amount of data that Earth Science missions, with planners like the one proposed, collect from the Earth's soil. For instance, a common case of study is the soil moisture. Some species are heavily conditioned on the humidity of their environment, and a slow-paced desertification could lead these species to eventually become endangered. The rapid detection of these geological changes could help prevent these biologically concerning situations.

Chapter 8

Conclusions

Much knowledge has been acquired from the development of this project. First, lets get an answer to the research questions exposed in Chapter 3 with the problem definition.

1. *Can we make an on-board, real-time, Earth Observation Satellite maneuvering decision-maker with modern Reinforcement Learning algorithms capable to outperform the general scheduling formulations?*

   This question is, admittedly, a little general. However, in terms of whether this project shows that the satellite direct maneuvering control can be done, the question must be yes. With the GUI testing, the visualization of the satellite's behaviour during inference was quite obviously performing in a good way. To be fair, this was not enough for a final version to be implemented, since it was still not very close to optimal behaviour. However, the general intentions to be achieved with the RL model were fulfilled.

2. *Which model architecture and algorithm are most suitable for the task in (1)?*

   Getting more into performance, the results have clearly shown a winner for both of the categories. PPO has performed consistently better than SAC in this particular case and implementation form. About the model architectures, although both could learn the task up to some level, the transformer-based architectures were particularly better than the MLPs. To be specific, the transformer encoder is the one that achieved the best performance seen in all runs.

3. *To what extent can the model work as the central brain of a multi-agent system?*

   The centralized decision-maker for an entire constellation could not be tested. Although a few models were trained for this purpose, this was done in early stages of the thesis calendar, and given the poor performance of SAC, no conclusions could be made. In the final stage, and still with promising performance improvements, not enough time was available for the time-consuming trainings with multiple agents together. This was on hold to secure the single-agent learning process that finally paid off.

In a nut shell, the overall research purpose was achieved to a certain extent. The single-agent case proved to work properly. With more time and resource allocation a fairly good model could be generated. The downside of this with respect to the initial purpose is that the multi-agent approach could not be treated enough. Hence, no conclusions on its functionality can be made.

Furthermore, several conclusions from different perspectives have arised. On the general thesis project side, it was perceived how a deep understanding of the problem is crucial. Robust knowledge on the matter allows one to be confident enough about how to formulate a problem and why this way to do it is novel. This usually comes with the prerequisite of taking a good and thorough literature review.

Understanding the evolution of the problem is critical too. In this one, it was clearly seen how, chronologically and performance-wise, the initial versions to tackle this problem were fairly different to how it is implemented nowadays. The first approaches relied a lot on dynamic programming, linear programming and graph construction algorithms. Despite these seemed pretty effective, the whole state of the art shifted towards a more resource-efficient solution. The agents tend to need that in real-life situations every time that more reactivity is required, and this generates a demand more performance-based that optimality-based. For this reason, AI models came to the scope of the problem formulation.

On the RL perspective, some highly important points can be made. First of all, there exist plenty of algorithms that have the potential to perform on some domains. However, there is not a perfect one that suits all. The classification of algorithms is an important feature to highlight, given that the diverse set of them come from different families. These families are based on different mathematical principles where the use cases are very sensitive to. For example, trying to implement a PPO algorithm on the Frozen Lake environment form Gymnasium library is highly difficult. It is easily solvable with Q-value iteration or policy iteration (which are primitive forms of RL), but if a PPO faces a 4 by 4 grid Frozen Lake, it is almost impossible to learn from it.[1] And this in only an example of the many particular and tricky situations that RL has to cope with.

In addition, the last stages of the project have shown how a well-studied set of reward metrics is needed. In many cases a more than acceptable performance from PPO was done. However, the resulting policy was far from the intentions. This iterative process to create the appropriate incentives is critical in the model fine-tuning.

From the model perspective, the expectations that transformer-based architectures had a high potential were filled. These proved to be, despite using a high number of parameters, very good for scalability and performance. Yet, Section 5.2.2 showed how tricky and sensitive these architectures are. The main concept is quite simple to figure out, but since most parts collaborate differently, a deep understanding of each one is crucial.

8.1 Distinctive Features of the Formulation

Aside from the conclusions one gets from the work done, there is an interest to see how this particular formulation fits among the state of the art versions of the satellite planning problem.

[1] This is given for the sparse rewards of this setting, where the PPO receives no incentives during all the training and is limited to the capability of a random policy to reach the end at least once. This is very difficult on a 4 by 4 grid without falling into an end-of-trajectory cell.

In this case, what makes this approach have higher potential than the rest is the state and action space definition. The task scheduling formulation has proven to be solvable and functional. Yet, one can not directly tell the agent in a satellite to do a certain task. These require secondary algorithms that turn the task into actual movement, which means that the movement it performs is fixed based on the future tasks to perform. However, this constraints the attitude motion to a certain patterns. Instead, the work of this thesis lets the model directly learn how to do these movement in the continuous spectra, allowing even more freedom to perform any sort of movement. Given that, the model could find nuances in the movement from one observation to another, leading to secondary observation purposes[2]. Here, the other problem formulations would struggle to find an adequate because the problem would be too demanding to assign a whole task for such a minor objective.

Another great deal of this formulation is that a lot of information is given to the agent in the form of augmented state space. That makes it capable enough to make the movement decisions instantly, right before doing them. This gives the model a huge advantage compared to the scheduling formulation in terms of reactivity. Some schedulers make the task allocation to cover the next hour or hour and a half of the satellite's orbit. This means that, on average, it will take the agent half an hour at least to reset its behaviour and accept new data incoming. Hence, it is inherently suboptimal since some unexpected high-interest events could arise in this interval of time. It is true that schedulers have a functionality to cancel their calendar and redo the task allocation, but the algorithm to choose when to and when not to do that is an additional (maybe imperfect) overhead that this formulation does not have to cope with.

Last but not least, the highest potential of this problem formulation lies in the nature of what the model is intended to learn. What the task schedulers have to learn on the big picture is how to make a planning. Despite the constraints added, the interest rates and other parametrization, their job is reduced to task allocation. Instead, the model here must learn how to move around itself. Since the movement taken is very lowly connected to the actual reward received, in contrary to selecting a specific task or another, the model here requires a much better understanding of the Earth's dynamics and its surface characteristics. Now, this has not been proven nor still observed in the scope of this project, but it would make sense to believe that the Earth-focused knowledge would make the model have a predictive ability with respect to upcoming events. A typical example would be a LEO satellite crossing the southern Pacific Ocean from Japan to Chile. With high probability there will be very few points of interest there for it. Yet, a predictive model would know that it is more rewarding to be looking towards Hawaii, Samoa, Cook Islands, French Polynesia or Fiyi rather than anywhere else on its FoR. And it would do this even though no special interest was specified. However, a scheduler would not be capable of creating such a planning if there are not any tasks assigned to these places.

[2] A secondary purpose could be scanning certain areas where it is known that an issue can arise. Maybe, there are certain events where these satellites are still not ready to detect because of how quickly they happen. In this case, a frequent focus to these places may be helpful.

Chapter 9

Future Work

The final trainings done, that proved a good agent behaviour, were made with a specific set of variations in model configuration. Despite a few different number of layers were tried, only some qualitative intuition about model sizing was acquired. A full study on this regard would bring more knowledge on which number of parameters for the model fits best for this use case. The proper way to tackle this would be a set of ordered trainings from a full factorial of a few different hyperparameters options. A genetic algorithm could make this process too.

Secondly, an obvious extension of this work is a specific study to tackle the multi-agent problem. This would require multi-agent trainings with models that forcibly receive the orbital coordinates as the state input. A bigger time allocation would be needed, given the number of satellites that would be needed to not let the model overfit on specific orbits. Also, and intuitively speaking, a bigger model might be needed, which adds up to the computational cost.

Finally, and looking beyond the current capabilities, it would be heavily interesting to unleash the predictive potential this model may be able to achieve. As said in Section 8.1, the nature of this problem formulation opens up this possibility, which would increase a few steps the efficiency of the best models available.

Annex

SAC Log-Prob Correction

An unbounded Gaussian distribution is used, but the actual action of the agent will be bounded. For that, a squashing function be needed. In this case, it will be a hyperbolic tangent, where $a_t = tanh(u)$ where u is an unbounded random variable array and $\mu(u|s)$ its probability density. For this, a correction to the log-probability will be needed to ensure consistency.

$$\pi(a|s) = \mu(u|s) \left| det \left(\frac{da}{du} \right) \right|^{-1}$$

The above equation must hold, and in the case of $a_t = tanh(u)$:

$$\frac{da}{du} = diag(1 - tanh^2(u))$$

This leads to the correct logarithmic probability density computation of:

$$log(\pi_\phi^{bound}(a_t|s_t)) = log(\pi_\phi^{unbound}(a_t|s_t)) - \sum_i log(1 - tanh^2(a_i^{unbound}))$$

Overestimation Bias

Q-learning algorithms use a maximization of an estimate as an estimate of the maximization with the Q-values. This leads to positive bias. See the policy:

$$\hat{\pi} = arg\max_a \hat{Q}(s,a), \qquad where\ \hat{V}_{\hat{\pi}} = \max_a \hat{Q}(s,a)$$

Notice now that the expectation of the value in this policy is overestimated:

$$\mathbb{E}[\hat{V}_{\hat{\pi}}] = \mathbb{E}[\max_a \hat{Q}(s,a)] \geq \max_a \mathbb{E}[\hat{Q}(s,a)]$$

This means the policy prioritizes actions with higher estimates rather than higher true values. For instance, let $Q(s, a)$ be the true values of a state and all of them be zero. Their estimates, $\hat{Q}(s, a)$, will be some higher than zero and some lower. Hence, the maximum of the estimates will be equal to or higher than the maximum of the true values (which is zero). Therefore, a positive bias.[29]

This is typically compensated in Q-learning with Double Q-learning. It can be proved that taking the minimum of the two Q-networks generates an underestimation compared to the single Q-learning. In most cases, this effectively compensates the maximization bias.

Bibliography

[1] NASA Headquarters Jenny Mottar. *NASA Earth Science Division Missions*. (Accessed: 23.04.2024). 2013. URL: https://svs.gsfc.nasa.gov/30065.

[2] Nyssa Rayne Kurtis Thome. *Terra*. (Accessed: 24.04.2024). 2025. URL: https://terra.nasa.gov/about/mission.

[3] Paul Przyborski Lazaros Oreopoulos Linette Boisvert. *Aqua*. (Accessed: 24.04.2024). 2024. URL: https://aqua.nasa.gov/.

[4] Kalina Velev. *TOEX/Poseidon*. (Accessed: 24.04.2024). 2017. URL: https://sealevel.jpl.nasa.gov/missions/topex-poseidon/summary/.

[5] Alex Fukunaga et al. "Aspen: A framework for automated planning and scheduling of spacecraft control and operations". In: *Proc. International Symposium on AI, Robotics and Automation in Space*. Citeseer. 1997, pp. 181–187.

[6] Al Globus et al. "A comparison of techniques for scheduling earth observing satellites". In: *AAAI*. 2004, pp. 836–843.

[7] Shreya Parjan James Mason. *ASPEN*. (Accessed: 24.04.2024). 2024. URL: https://ai.jpl.nasa.gov/public/projects/aspen/.

[8] eoPortal. *EO-1 (Earth Observing-1)*. (Accessed: 24.04.2024). 2024. URL: https://www.eoportal.org/satellite-missions/eo-1#launch.

[9] Hai Li et al. "A Multi-Objective Dynamic Mission-Scheduling Algorithm Considering Perturbations for Earth Observation Satellites". In: *Aerospace* 11.8 (2024). ISSN: 2226-4310. DOI: 10.3390/aerospace11080643. URL: https://www.mdpi.com/2226-4310/11/8/643.

[10] Wei-Cheng Lin et al. "Daily imaging scheduling of an Earth observation satellite". In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 35.2 (2005), pp. 213–223. DOI: 10.1109/TSMCA.2005.843380.

[11] Yuchen She, Shuang Li, and Yanbin Zhao. "Onboard mission planning for agile satellite using modified mixed-integer linear programming". In: *Aerospace Science and Technology* 72 (2018), pp. 204–216. ISSN: 1270-9638. DOI: https://doi.org/10.1016/j.ast.2017.11.009. URL: https://www.sciencedirect.com/science/article/pii/S1270963817311926.

[12] Qingyu Qu et al. "Satellite Observation and Data-Transmission Scheduling Using Imitation Learning Based on Mixed Integer Linear Programming". In: *IEEE Transactions on Aerospace and Electronic Systems* 59.2 (2023), pp. 1989–2001. DOI: 10.1109/TAES.2022.3210073.

[13] Adam Herrmann and Hanspeter Schaub. "Reinforcement Learning for the Agile Earth-Observing Satellite Scheduling Problem". In: *IEEE Transactions on Aerospace and Electronic Systems* 59.5 (2023), pp. 5235–5247. DOI: 10.1109/TAES.2023.3251307.

[14] Autonomous Vehicle Systems (AVS) Laboratory. *Welcome to Basilisk*. (Accessed: 24.04.2024). 2024. URL: https://avslab.github.io/basilisk/.

[15] Ming Chen et al. "Deep Reinforcement Learning for Agile Satellite Scheduling Problem". In: *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2019, pp. 126–132. DOI: 10.1109/SSCI44817.2019.9002957.

[16] Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[17] Alan Aguilar Jaramillo et al. “Decentralized Consensus-Based Algorithms for Satellite Observation Reactive Planning With Complex Dependencies”. In: *AIAA SCITECH 2025 Forum*. 2025, p. 1148.

[18] OpenAI. *Kinds of Algorithms*. 2018. URL: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html.

[19] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *nature* 518.7540 (2015), pp. 529–533.

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

[21] John Schulman et al. “Proximal Policy Optimization Algorithms”. In: *CoRR* abs/1707.06347 (2017). arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347.

[22] Colah’s Blog. *Understanding LSTMs*. 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs.

[23] Kemal Erdem. “Understanding Positional Encoding in Transformers”. In: *https://erdem.pl* (May 2021). URL: https://erdem.pl/2021/05/understanding-positional-encoding-in-transformers.

[24] Farama Foundation. *Gymnasium Documentation*. 2025. URL: https://gymnasium.farama.org/.

[25] OpenAI. *Soft Actor-Critic*. 2025. URL: https://spinningup.openai.com/en/latest/algorithms/sac.html.

[26] Arsenii Kuznetsov et al. “Controlling Overestimation Bias with Truncated Mixture of Continuous Distributional Quantile Critics”. In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 13–18 Jul 2020, pp. 5556–5566. URL: https://proceedings.mlr.press/v119/kuznetsov20a.html.

[27] Tuomas Haarnoja et al. “Soft actor-critic algorithms and applications”. In: *arXiv preprint arXiv:1812.05905* (2018).

[28] Meta. *TorchRL*. 2022. URL: https://docs.pytorch.org/rl/stable/index.html.

[29] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2nd. MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html.

[30] The UN agency for digital technologies. *UN agencies warn of satellite navigation jamming and spoofing*. 2025. URL: https://www.itu.int/hub/2025/03/un-agencies-warn-of-satellite-navigation-jamming-and-spoofing.