Master's Degree in Aerospace Engineering

Master's Degree Thesis

Spacecraft Collision Avoidance: a Transformer-based Reinforcement Learning Approach

Supervisors

Prof. Manuela BATTIPEDE

Prof. Luigi MASCOLO

Candidate

Paolo CIRRINCIONE PAZÉ

ACADEMIC YEAR 2024-2025

"Earth is the cradle of humanity,
but one cannot live in the cradle forever."
— Konstantin Tsiolkovsky

Table of Contents

List of Figures

List of Tables

Acronyms

ADR Active Debris Removal

CAM Collision Avoidance Maneuver

CDM Conjunction Data Message

CNN Convolutional Neural Network

COE Classical Orbital Element

DL Deep Learning

DQN Deep Q-Network

ECI Earth-Centered Inertial

ESA European Space Agency

GEO Geostationary Orbit

IADC Inter-Agency Space Debris Coordination Committee

ISS International Space Station

LEO Low Earth Orbit

LSTM Long Short-Term Memory

MDP Markov Decision Process

ML Machine Learning

MLP Multilayer Perceptron

NASA National Aeronautics and Space Administration

POMDP Partially Observable Markov Decision Process

PPO	Proximal Policy Optimization
RF	Reference Frame
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RTN	Radial–Transverse–Normal
S/C	Spacecraft
SSN	Space Surveillance Network
TCA	Time of Closest Approach

Abstract

The development of the space economy and the ever-growing interest toward space has led to the progressive congestion of the most commercially viable Earth orbits. More and more satellites are launched around our planet each year, increasing the risk of collisions between space objects that have the potential of creating millions of debris and an even more dangerous orbital environment. The necessity to develop collision avoidance tools and techniques has never been more pressing, as spacecraft have to perform avoidance maneuvers with increasing frequency. In this scenario, trajectory optimization becomes of paramount importance, in order to avert collisions in the most effective way. This research proposes an implementation of a Deep Reinforcement Learning framework to optimize the path of a satellite orbiting our planet in a low Earth orbit and confronted with multiple collision warnings. The proposed approach addresses imperfect environmental modeling and measurements by using a Partially Observable Markov Decision Process. To add flexibility to the method, the states of a variable number of space debris are first processed by a Long Short-Term Memory to create a fixed-sized summary of the multiple space objects information, before being concatenated with the observation of the spacecraft state. In this way, the hidden state information is replaced with a belief vector derived from the observation time sequence (history), which is weighted by a Transformer encoder to capture the non-linear dynamics of the signals. The resulting semantic history guides an agent employing Proximal Policy Optimization, an online policy estimation method, which relies on two neural networks: a critic for value estimation and an actor for policy evaluation, implemented as Multi-Layer Perceptrons. The model considers the motion of the satellite and multiple debris in LEO, under the J2 gravitational perturbation and the effect of atmospheric drag. The reward function has been designed to achieve the reduction of the collision probabilities below a critical threshold and minimum fuel consumption. A station-keeping requirement has also been introduced. Significant results obtained from the simulations are presented, highlighting the trends of the most important physical quantities and the progression of the learning of the neural networks. The research concludes by analyzing the implications of the method and its potential applications.

Chapter 1

Introduction

1.1 The issue of space debris

Mankind has been driven to space for all kinds of reason. Initially, military dominance was the primary motivation behind the advancement of space technology. The Cold War ignited the Space Race, beginning with the launch of Sputnik 1 in 1957 and culminating in the Apollo missions and the first Moon landing in 1969. Later, scientists quickly recognized the potential of the space environment for conducting scientific research, as habitable space stations provided the ideal environment for the execution of experiments requiring microgravity. Meanwhile, Earth observation satellites and space telescopes have significantly improved our understanding of both our planet and the broader universe. Space has long been regarded as the final frontier, driving the launch of numerous exploration probes and rovers on interplanetary missions since the 1970s. However, perhaps the most compelling reason behind humanity's continued expansion into space is the vast economic potential it offers. Communication satellites have revolutionized global connectivity, supporting everything from television broadcasts to internet services and GPS navigation, and today they represent the lion's share of the space economy [1]. Figures 1.1 and 1.2, included in the 2024 European Space Agency (ESA) Annual Space Environment Report [2], show the change in both the type and mass of spacecraft that have been launched to orbit since the beginning of the Space Age. Heavier satellites, primarily for imaging, scientific and technological purposes, have progressively, and especially in the last decade, given way to lighter and smaller communication satellites.

The miniaturization of space systems and the rise of large satellite constellations have transformed the landscape of satellite launches, particularly in low Earth orbit (LEO). Traditional large, geostationary satellites are being complemented and, in some cases, replaced by smaller, more cost-effective satellites, highlighting the shift

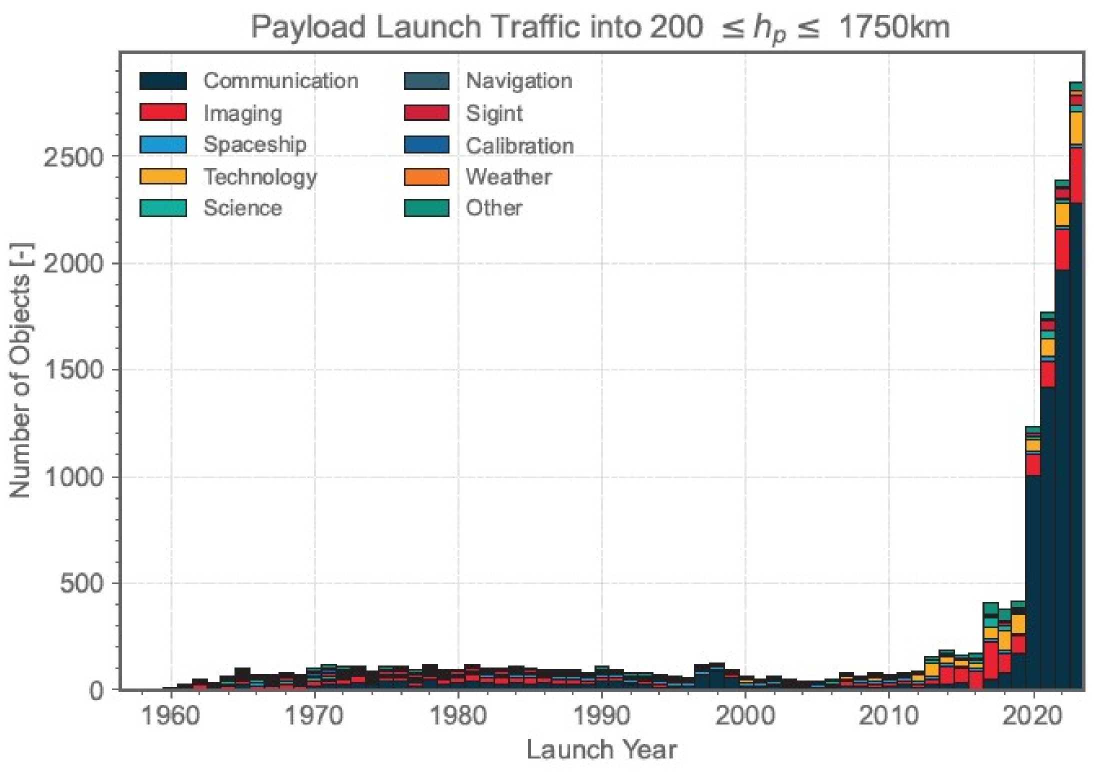

Figure 1.1: Spacecraft by type

toward the New Space Economy, where private companies play an increasingly dominant role in space activities. This transition is characterized by lower launch costs, rapid innovation, and greater accessibility, allowing start-ups and commercial entities to compete alongside traditional space agencies. The main consequence of this transformation, which has been further accelerated by the emergence of reusable rockets, mass satellite production, and flexible launch services, has been the immense increase in the number of space objects. This exponential growth in orbital traffic has brought renewed attention to the issue of space debris, raising concerns about long-term sustainability and the safety of space operations. In this context, the Inter-Agency Space Debris Coordination Committee (IADC) defines space debris as: "all human made objects, including fragments and elements thereof, in Earth orbit or re-entering the atmosphere, that are non-functional. During the operational phases, a spacecraft or orbital stage can be considered as functional" [3].

These objects include various types of debris such as payloads, payload fragmentation debris, rocket bodies and rocket fragmentation debris, predominantly concentrated in two orbital regions: LEO and the geostationary orbit (GEO) ring. Figure 1.3, a computer generated image by the National Aeronautics and Space

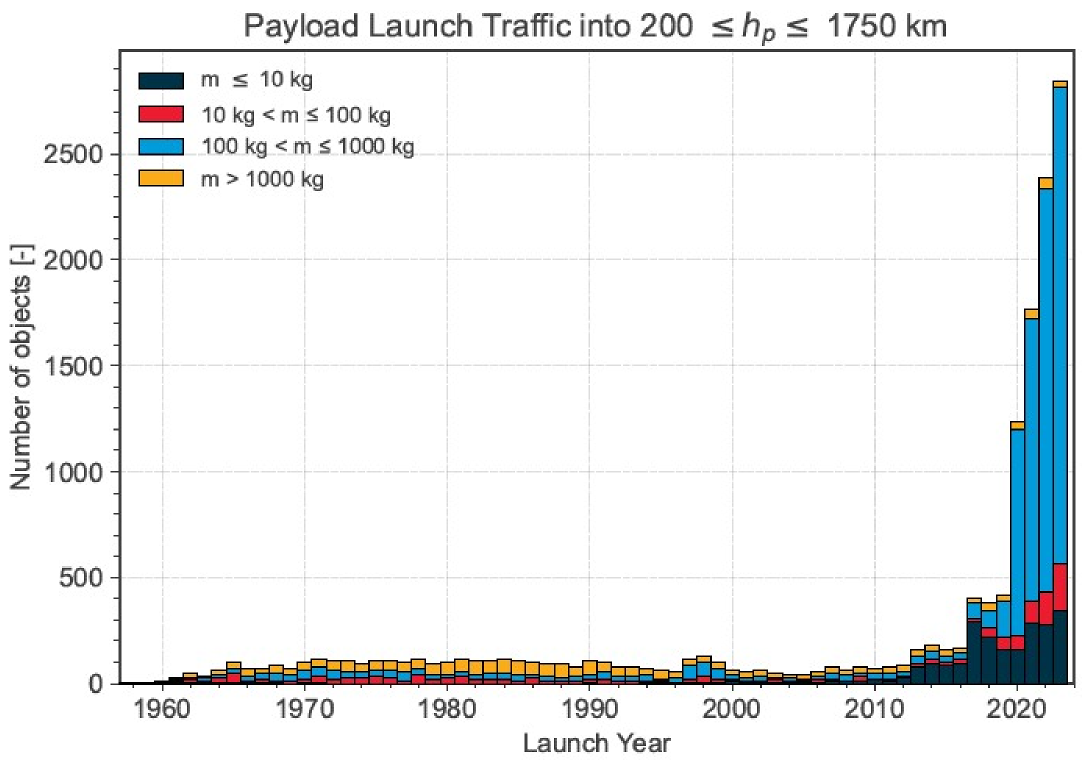

Figure 1.2: Spacecraft by mass

Administration (NASA) Orbital Debris Program Office [4], shows the space objects currently being tracked. LEO, which extends up to approximately 2,000 kilometers above Earth's surface, hosts the highest density of debris due to the large number of active satellites, particularly from commercial constellations, as well as decades of accumulated fragments from past missions and in-orbit breakups. GEO is a vital orbital region due to its unique ability to host satellites that maintain a fixed position relative to the Earth's surface, which makes it ideal for telecommunications, weather monitoring, and broadcasting. As a result of this strategic value, a high concentration of satellites has accumulated in this orbit over the decades, leading to a corresponding build-up of space debris.

This growing congestion paints a very different picture from how we used to perceive space. The orbital environment that surrounds our planet has changed profoundly. The idea that, beyond the atmosphere, empty space begins does not hold true any longer, as more and more man-made objects are launched into orbit. Although space debris has outnumbered operational satellites since the earliest space missions, it is only in recent decades, thanks to continuous advancements in space surveillance technologies, that we have come to fully grasp the scale of the

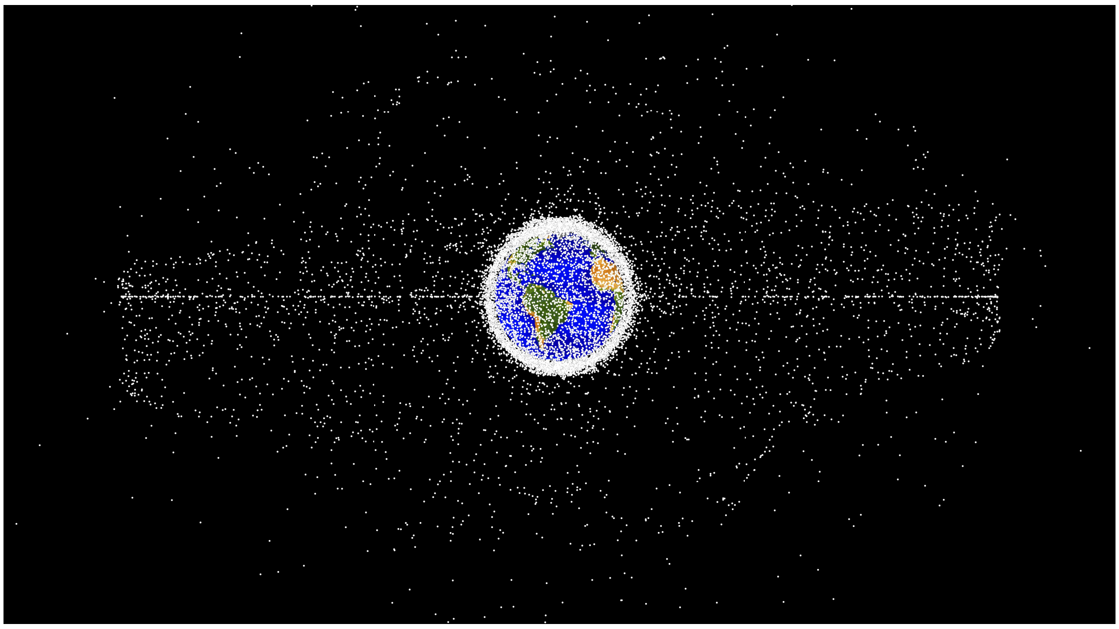

Figure 1.3: Computer generated image of space objects orbiting the Earth

problem. As a result of these improvements, even smaller pieces of debris can now be accurately tracked and recorded. As of March 2025, ESA's Space Debris Office provides updated statistics on the estimated number of debris objects currently orbiting Earth [5]. The United States Space Surveillance Network (SSN), which will be described in more detail in the following section, actively tracks approximately 40,000 objects larger than 10 cm in size. However, smaller debris, which are more difficult to detect, can only be estimated statistically. According to the Space Debris Office, over 1 million objects between 1 cm and 10 cm in size, and around 130 million objects smaller than 1 cm, are currently believed to be orbiting our planet.

Figure 1.4, published in the NASA Orbital Debris Quarterly News from February 2025, illustrates the historical trend of the trackable space debris population in Earth's orbit. A close examination of the figure reveals that the most significant increases in space debris are closely linked to fragmentation events, such as breakups, explosions, collisions, and other anomalous incidents. The study by Braun [6] provides an in-depth analysis of these fragmentation events and investigates the origins of currently tracked debris. The findings confirm that the dominant source of space debris is indeed breakup events, underscoring the critical impact of such incidents on the growing debris population.

Furthermore, specific spikes in the debris trend shown in the figure can be attributed to distinct catastrophic events. One notable example is the 2007 Chinese

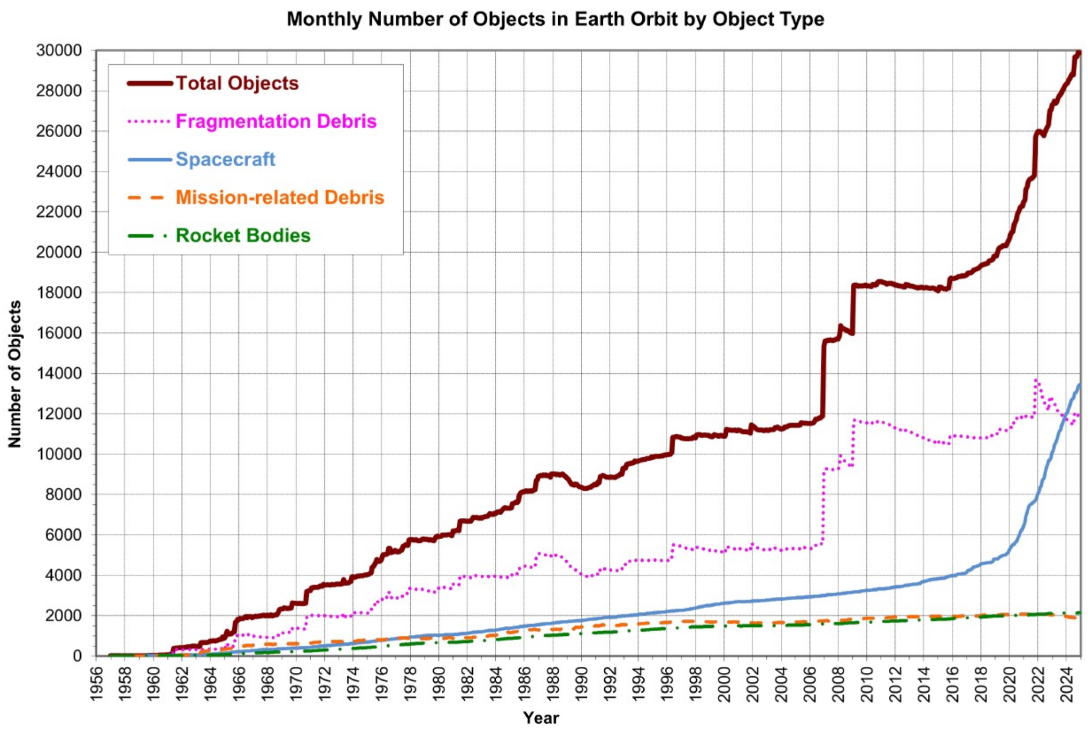

Figure 1.4: Total number of trackable space objects

anti-satellite (ASAT) missile test. On January 11 2007, China successfully carried out a direct-ascent ASAT test targeting one of its own defunct polar-orbiting weather satellites. This test resulted in the creation of at least 2,087 pieces of debris large enough to be tracked by the U.S. SSN. According to the NASA Orbital Debris Program Office, the event generated over 35,000 fragments larger than 1 centimeter in size. The destruction of the Fengyun-1C satellite in this single test increased the population of trackable space objects by approximately 25 percent, making it the largest debris-generating incident ever recorded [7].

Another significant spike observable in the figure, particularly relevant to the focus of this thesis, is the 2009 Iridium–Cosmos collision. Although previous collisions between space objects had occurred, this event marked the first recorded accidental collision between two intact satellites. On 10 February 2009, at 16:56 UTC, Iridium-33, a privately owned American communications satellite, and Kosmos-2251, a defunct Russian military satellite, collided at an altitude of approximately 790 km over Siberia. The satellites were traveling at a relative velocity of 11.6 km/s and were both completely destroyed in the impact. The collision generated over 2,300 trackable fragments, forming two distinct debris clouds in LEO. Initially, each debris cloud followed the orbital path of its respective satellite. In this case, the original orbits were nearly perpendicular, leading to a particularly complex distribution of debris. Over time, the fragments began to disperse along these orbits and eventually evolved into broader shells of debris concentrated around the original collision altitude, posing a long-term global threat to all satellites operating within or crossing that orbital band [8, 9].

A third major event was the 2012 breakup of a Russian Briz-M upper stage. On 16 October 2012, following a failed orbital insertion in August, the fuel-laden stage exploded while stranded in a highly elliptical orbit with a low perigee of around 290 km. The explosion generated a substantial cloud of debris, with over 700 large fragments detected, many of which had perigees below the altitude of the International Space Station (ISS). The low altitude of the breakup, combined with the favorable orbital geometry, allowed for detailed radar observations of the debris, including fragments smaller than those typically detectable by the U.S. SSN [10].

These events collectively highlight the critical importance of preventing catastrophic incidents that significantly increase the debris population in orbit. While large, intact objects pose a manageable risk, the proliferation of smaller fragments greatly complicates tracking and mitigation efforts. In addition to fragmentation debris, several other sources contribute to the growing orbital debris environment. One of the most significant non-fragmentation sources has been over 2400 solid rocket motor firings, which have dispersed aluminum oxide (Al_2O_3) into space as micrometer-sized dust and millimeter- to centimeter-sized slag particles. Furthermore, prolonged exposure to extreme ultraviolet radiation, atomic oxygen, and micrometeoroid impacts gradually erodes spacecraft surfaces. This degradation results in the loss of surface coatings and the release of paint flakes ranging in size from a few micrometers to several millimeters.

It is also worth noting that, as illustrated in Figure 1.4, there are periods during which the total number of tracked debris decreases. This reduction is primarily due to the natural decay of orbital debris, which eventually reenters Earth's atmosphere and burns up. The rate of this decay is influenced by the solar cycle, as increased solar activity heats and expands the upper atmosphere, thereby increasing atmospheric drag, particularly on objects in LEO. However, debris located in higher regions of LEO experience significantly less drag and may remain in orbit for decades or even centuries before reentry occurs.

The generation of debris through the aforementioned mechanisms, combined with the mitigating effects of natural cleansing processes, results in a spatially uneven distribution of debris, both in altitude and latitude. As illustrated in Figure 1.5, extracted from [2], the highest concentrations of debris are observed between 750 and 1000 km, with a secondary peak near 1400 km altitude. In contrast, spatial debris densities in GEO and near the orbits of navigation satellite constellations are two to three orders of magnitude lower. Although the ISS orbits at lower altitudes than the regions of highest debris density, it is still at risk and must regularly perform collision avoidance maneuvers (CAMs) to ensure crew safety.

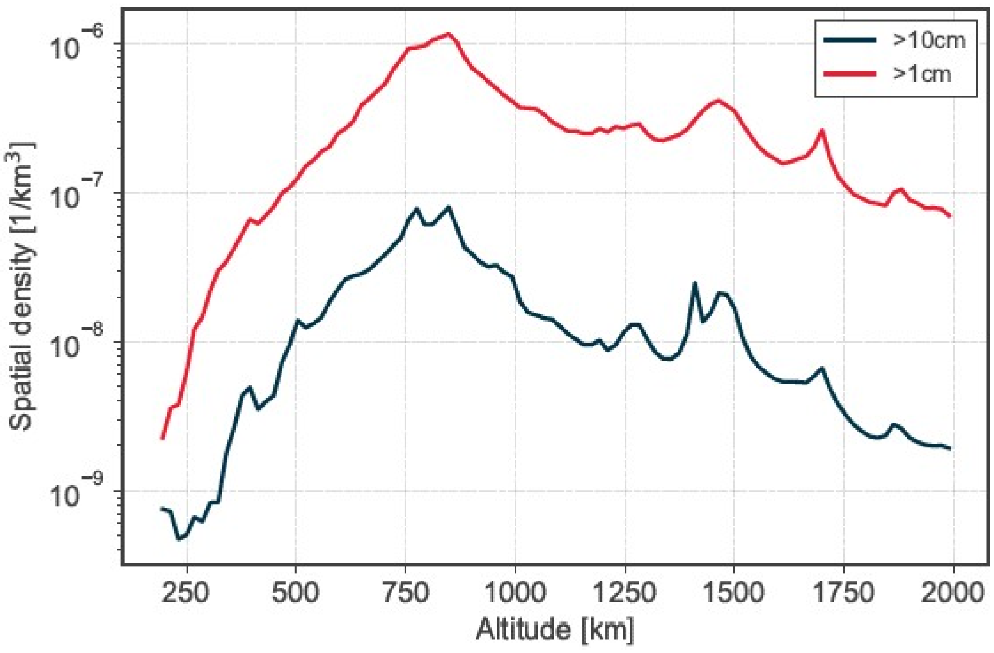

Figure 1.5: Debris density profile in LEO

Over the years, several technological solutions have been proposed and developed to address the growing concern of space debris. These solutions can be broadly classified into mitigation, remediation, and design-oriented strategies.

Mitigation refers to efforts aimed at preventing the creation of new debris and minimizing the risk of collision or fragmentation during and after the operational life of a spacecraft. A cornerstone of space debris mitigation is the implementation of post-mission disposal (PMD) measures, which ensure that spacecraft and launch vehicles are removed from critical orbital regions at the end of their operational lives. As shown in Figure 1.6, taken from [11], PMD compliance significantly reduces the long-term growth of debris. Complementary to PMD is the concept of passivation, which involves depleting all onboard energy sources, such as residual propellants, batteries, or pressurized systems, to prevent explosions or breakups. Effective mitigation also relies on collision avoidance, which has become one of the most important strategies in recent years. By equipping spacecraft with maneuvering capabilities, they are able to perform CAMs upon receiving conjunction warnings, thereby preventing potentially catastrophic collisions. Advancements in propulsion systems further support these efforts, as more efficient engines not only consume less fuel but also extend a satellite's operative life, reducing the frequency of replacements and launches. Solid rocket motors, which release aluminum oxide

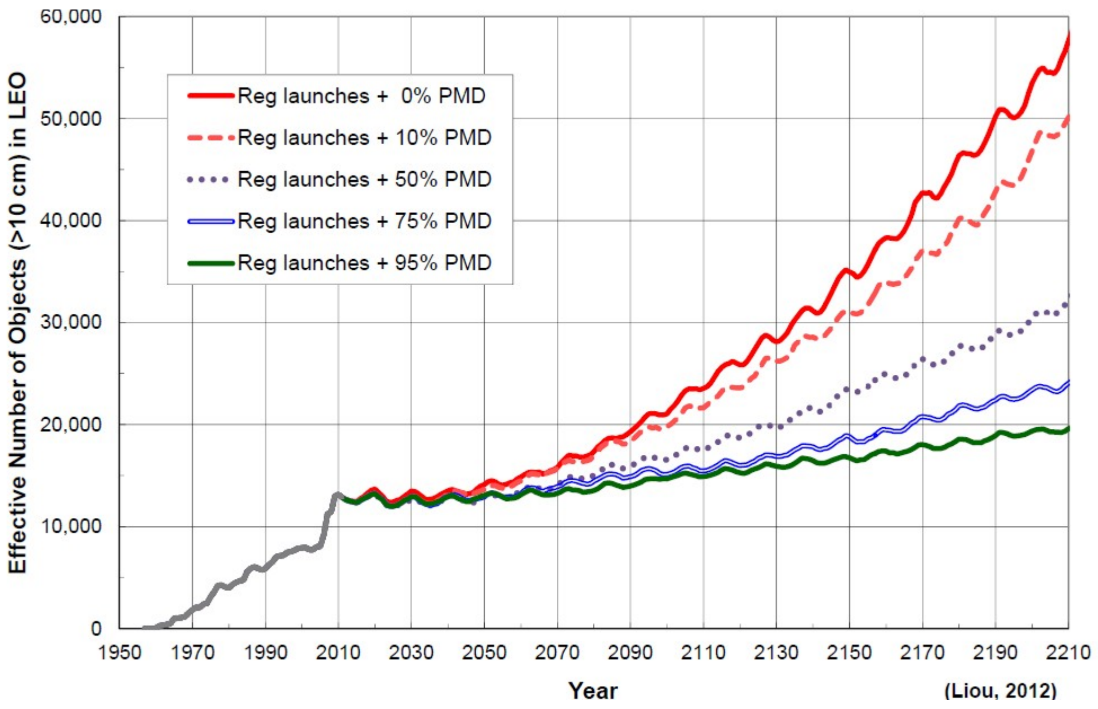

Figure 1.6: Mitigation effect of PMD

particles during burns, must be designed not to emit particles larger than 1 mm, especially in densely populated orbits like LEO and GEO.

Beyond these operational measures, emerging design philosophies play a crucial role in debris prevention. Design for Demise (D4D) involves engineering spacecraft components to fully disintegrate during atmospheric reentry, reducing the chance of debris surviving to the ground. Meanwhile, Design for Removal (D4R) supports future cleanup efforts by allowing spacecraft to be easily captured and deorbited by Active Debris Removal (ADR) systems. These concepts align with the broader shift toward a space circular economy, which mirrors Earth-based sustainability practices by promoting reuse, recycling, and long-term resource efficiency. A successful example of this philosophy is the reusability model adopted by SpaceX for their Falcon 9 boosters, which not only provides economic advantages but also contributes to debris mitigation.

Nevertheless, studies have shown that addressing the existing population of space debris will eventually require remediation technologies [11], which although still in early stages, are essential for long-term sustainability. ADR missions have begun to demonstrate feasibility; key initiatives include ELSA-d launched in 2021, ADRAS-J in 2024, and the upcoming ClearSpace-1 mission, all of which aim to capture and deorbit defunct objects. These missions face significant technical and economic hurdles but represent critical progress toward remediation. In parallel, in-orbit servicing technologies are emerging as an alternative to satellite replacement. A notable example is Northrop Grumman's MEV-1, which in 2020 successfully docked with Intelsat 901 to extend its operational life, demonstrating how existing infrastructure can be maintained rather than discarded. A more experimental avenue involves ground-based laser systems, which aim to nudge small debris into lower orbits, accelerating their reentry.

In conclusion, while remediation technologies continue to evolve, mitigation remains the only mature and widely applicable solution to the space debris problem at present. The integration of robust disposal practices, active collision avoidance, and responsible design choices represents the most practical path forward to ensure long-term access to and safety within Earth's orbital environment.

In addition to technological measures, addressing the space debris problem requires a broad and coordinated legal and normative approach. The challenge is inherently global and it affects all countries, making international cooperation essential. The current unregulated and unrestricted access to outer space exacerbates the issue, as technological advancement alone is insufficient to address the growing risks. Existing legal frameworks often fail to comprehensively incorporate debris-related concerns, and only in recent years have global efforts begun to address these shortcomings. The establishment of legal instruments, the development of internationally accepted standards, and the implementation of binding regulations are fundamental to ensuring the long-term sustainability of outer space.

Initiatives such as the creation of the IADC in 1993 have marked crucial milestones in this process. Comprising 13 space agencies as of 2025, the IADC has facilitated international collaboration by sharing research, coordinating efforts, and developing mitigation strategies, most notably through its guidelines, first published in 2002. In their 2025 version, a particular emphasis is put on the prevention of on-orbit collisions, recognizing them as a major driver of future space debris growth. Accordingly, mission planners are encouraged to estimate and minimize the probability of such collisions throughout a spacecraft's orbital lifetime [3].

These guidelines have laid the foundation for further regulatory efforts, such as the United Nations space sustainability guidelines developed by the United Nations Office for Outer Space Affairs (UNOOSA) and the Committee on the Peaceful Uses of Outer Space (COPUOS), which, although non binding, have gained broad international support. The guidelines recommend the registration of all space objects, the sharing of monitoring data, and the performance of conjunction assessments to minimize collision risks [12].

Building on the same principles, ISO began developing technical standards in 2003, leading to the release of ISO 24113 in 2010, with the most recent update issued in 2023. This standard, closely aligned with IADC recommendations, introduces concrete, quantitative mitigation requirements [13].

The European Cooperation for Space Standardization (ECSS) has further refined these efforts by adopting the ISO standard with several targeted modifications, particularly strengthening provisions related to collision avoidance and maneuvering capabilities. It also emphasizes a concept that will be particularly relevant throughout the remainder of this work: the need to maintain the probability of collision below a defined threshold. While some mitigation actions are difficult to express in strictly quantitative terms and remain qualitative in nature, the standard provides guidance for cases where spacecraft have the capability to actively manage collision risks. Specifically, if the assessed collision probability with other space objects exceeds a threshold established by an approving authority, the operator is required to perform avoidance maneuvers to reduce the risk below that threshold [14].

In parallel, ESA has developed a robust policy framework to implement these standards across its missions. The Space Debris Mitigation Requirements [15] set binding internal rules on design, operations, and end-of-life procedures, while its Clean Space initiative introduces an integrated approach that merges environmental responsibility with technological innovation.

Finally, it is important to note that various approaches can be adopted to promote space sustainability. These include hard law measures, such as mandatory regulations exemplified by French law no. 2008-518, and soft law mechanisms, as seen in Japan, where compliance is voluntary and based on guidelines rather than binding legislation.

1.2 Collision avoidance

As discussed in the previous section, one of the key mitigation strategies emphasized by international organizations is the prevention of in-orbit collisions.

While research has shown that objects larger than 10 cm pose a major hazard, capable of causing catastrophic damage upon impact, smaller debris can often be tolerated without compromising spacecraft functionality. For example, Sentinel-1A sustained an impact to one of its solar arrays from a millimeter-sized particle in 2016, yet continued operations without issue. Another notable case is the Hubble Space Telescope, whose solar arrays, when brought back to Earth, were found to have over 3,000 pits from micrometeoroid and debris impacts [16, 17]. Despite this, the telescope remained fully operational while in orbit. Similarly, the Cupola module on the International Space Station has shown surface pitting caused by repeated exposure to small debris over time.

The main concern arises from the extremely high relative velocities in orbit, which result in significant momentum and kinetic energy transfer during collisions.

For particles smaller than a micrometer, the damage is typically superficial, leading mainly to surface degradation, similar to a sandblasting effect.

To mitigate such threats, shielding systems have been shown to have the ability to protect critical spacecraft components from impacts with smaller objects [18]. However, as debris size increases, the potential for catastrophic damage rises sharply. Studies have also suggested that the severity of impact damage scales more directly with the projectile's momentum than with its kinetic energy alone, indicating that mass may play a more significant role than previously assumed [19].

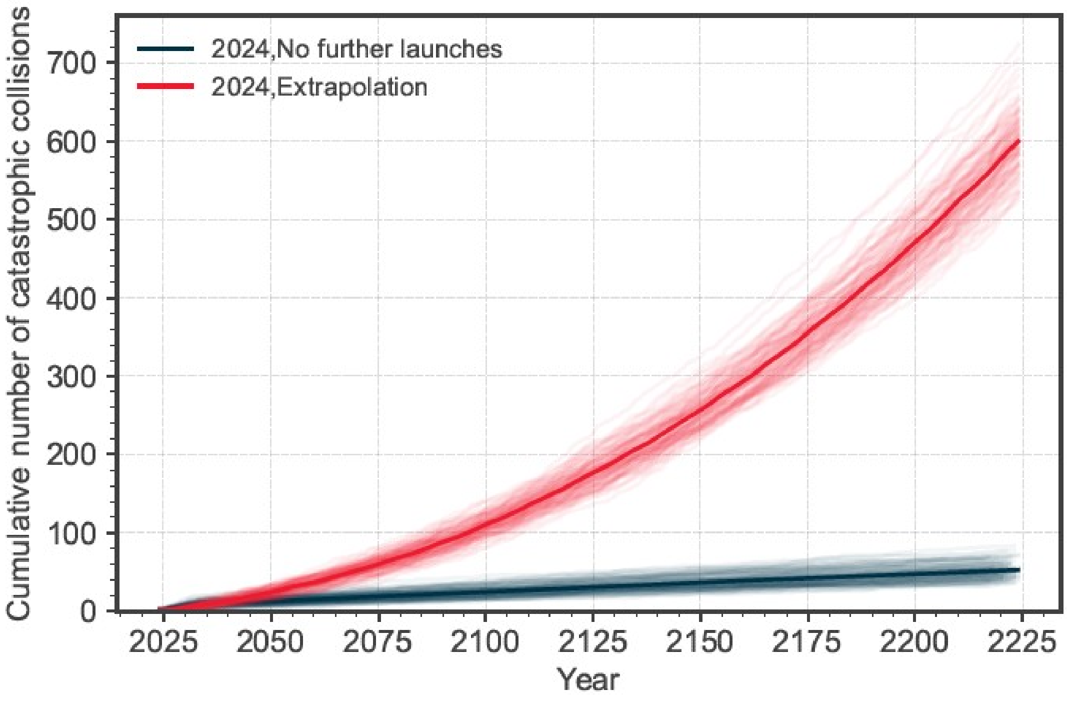

Figure 1.7: Estimated future trend of collisions in LEO

The most severe consequence of failing to address the growing issue of space debris, particularly the threat of in-orbit collisions, is known as Kessler Syndrome.

This phenomenon refers to a self-sustaining cascade of collisions in space. As the number of debris objects increases, so does the probability of catastrophic collisions. Over time, these collisions generate additional fragments, which in turn lead to further collisions. Eventually, collision fragments begin impacting other collision fragments, perpetuating a runaway chain reaction. This positive feedback loop is especially critical in the LEO region, where space traffic is densest. Without effective mitigation, the environment could become so crowded with debris that safe operations would become nearly impossible [20].

Figure 1.7, extracted from [2], suggests that, even in a hypothetical scenario where no new satellites are launched, the existing population of debris in LEO would continue to grow due to ongoing collisions among objects already in orbit. Under a "business-as-usual" scenario, where no significant mitigation measures are implemented, the projected increase in collisions is alarming and aligns closely with the conditions predicted by Kessler.

Figure 1.8, adapted from [21], illustrates the same trend when examining the total number of trackable objects. Even in a scenario with no additional launches, collisions are projected to surpass other types of in-orbit breakups as the dominant source of new debris, leading to a continuous growth in the overall population.

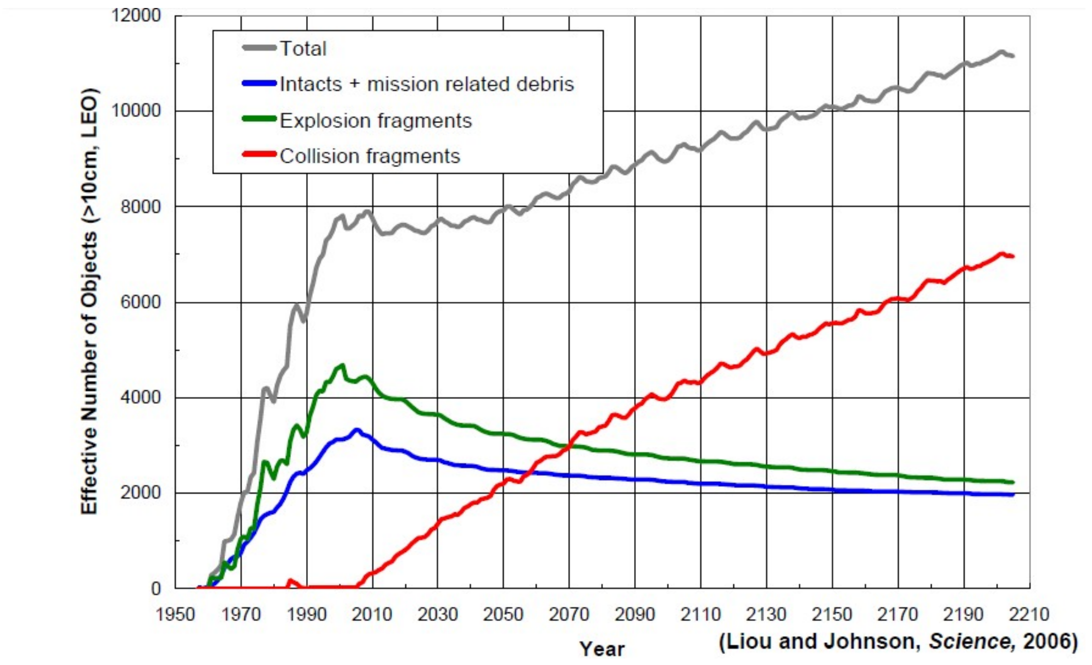

Figure 1.8: Projection of the total number of trackable objects in LEO

Following the discussion about the risk of inaction, we now turn to the operational side of space safety, and specifically how collision avoidance is performed. The process begins with the detection and tracking of objects in orbit, primarily conducted by the United States Space Surveillance Network.

The SSN is a global network composed of numerous ground-based sensors, including radar installations and optical telescopes, distributed across various locations (see Figure 1.9). These sensors continuously monitor the sky to detect, track, and catalog space objects.

The ability to detect and track debris depends heavily on both the size and altitude of the object. Radar systems, typically used for monitoring objects in LEO, are effective at detecting smaller debris thanks to their active signal transmission and their capacity to operate under all lighting conditions. In contrast, optical

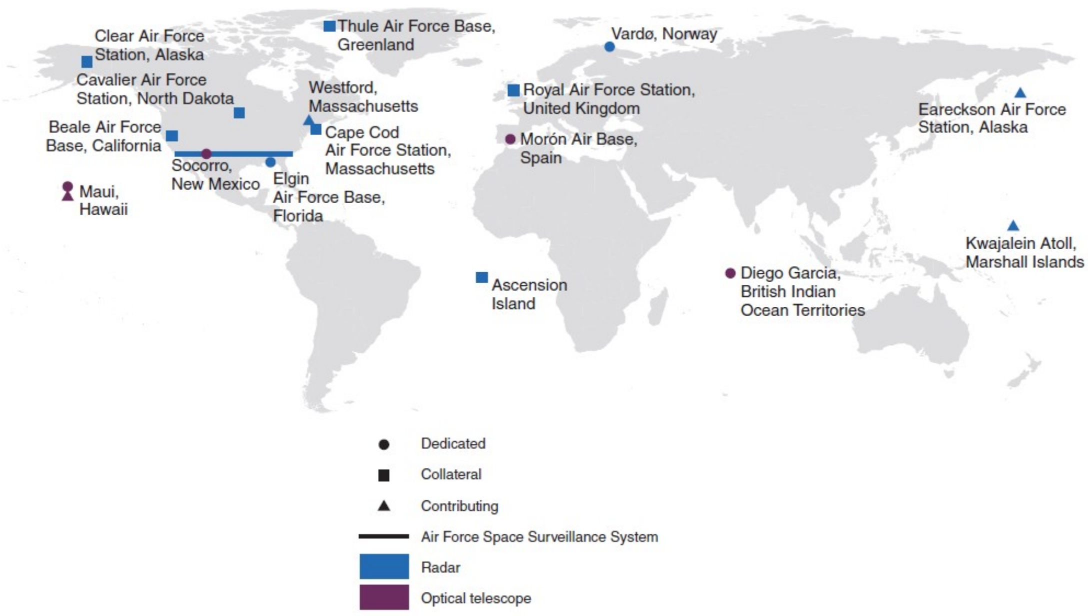
Clear Air Force Station, Alaska
Thule Air Force Base, Greenland
Vardø, Norway
Cavalier Air Force Station, North Dakota
Westford, Massachusetts
Royal Air Force Station, United Kingdom
Eareckson Air Force Station, Alaska
Beale Air Force Base, California
Cape Cod Air Force Station, Massachusetts
Morón Air Base, Spain
Socorro, New Mexico
Elgin Air Force Base, Florida
Maui, Hawaii
Kwajalein Atoll, Marshall Islands
Diego Garcia, British Indian Ocean Territories
Ascension Island
Dedicated
Collateral
Contributing
Air Force Space Surveillance System
Radar
Optical telescope

Figure 1.9: Types and locations of SSN sensors

telescopes rely on passive observation of sunlight reflected off objects. These systems are particularly useful for tracking objects in higher orbits, such as GEO, but are limited by weather conditions and the availability of sunlight.

While radar is more robust against environmental constraints, it tends to have lower angular resolution at greater distances. Optical systems, although more precise in angular tracking, are susceptible to atmospheric interference and cannot operate during daylight or cloudy conditions.

In addition to these ground-based systems, active satellites equipped with onboard GPS receivers can use positioning data to track their own location with high accuracy. However, GPS-based tracking is only applicable to satellites that carry such receivers, and it cannot be used to track uncooperative objects or debris. For these non-cooperative targets, radar and optical telescopes remain the primary tracking methods.

Currently, the US Space Surveillance Network tracks over 40,000 objects larger than 10 cm in diameter. Depending on the type of sensor used, the collected measurements may include parameters such as range, range rate, elevation and azimuth angles, and angular rates. Once these observations are gathered, orbit determination techniques are employed to estimate the trajectory of each object. Two commonly used methods are batch least squares and Kalman filtering. Batch least squares is a retrospective technique that processes a set of measurements collected over a specific time window. It determines the orbit that best fits the data by minimizing the sum of the squared differences between observed and predicted positions. Kalman filtering, on the other hand, is a recursive, real-time estimation method. It continuously updates the object's estimated state whenever new measurements become available [22, 23].

In the United States, the 18th Space Defense Squadron (18 SDS) is responsible for processing observational data from the SSN, performing orbit determination, and propagating the state of resident space objects forward in time. Using automated processes executed multiple times per day, 18 SDS accurately estimates each object's position and velocity. These calculations are then used to maintain and continuously update the High Accuracy Catalog (HAC), a vital resource for tracking and monitoring objects in Earth orbit.

The 19th Space Defense Squadron (19 SDS), meanwhile, is tasked with conducting Conjunction Assessment (CA), a mission that includes both on-orbit and launch-related collision avoidance. This involves identifying potential close approaches between space objects and recommending mitigation strategies to reduce the risk of collision. To carry out this mission, the 19 SDS relies on the HAC maintained by 18 SDS. Leveraging this high-accuracy orbital data, the 19 SDS CA team systematically screens the trajectories of all resident space objects, including active satellites, defunct payloads, rocket bodies, and other debris, to detect and assess potential conjunctions [24].

An important point to understand in orbit determination is the presence of uncertainties and how they are captured through covariance. Every sensor used to track space objects introduces some level of measurement error, such as inaccuracies in range or angle. These sensor-based uncertainties are expressed by the covariance matrix at the time of observation, which quantifies the confidence in our knowledge of the object's state. This covariance is then propagated forward using dynamical models to estimate the object's future covariance, especially at the time of closest approach (TCA). Because of these uncertainties, we cannot predict collisions deterministically; instead, we rely on probabilistic assessments to quantify the risk.

This is where Conjunction Data Messages (CDMs) come into play. Issued by the 18th Squadron to satellite operators, CDMs alert them to potential collisions by providing details about the predicted conjunction. Each CDM corresponds to a close approach between two monitored space objects. The message contains an estimate of the TCA, along with estimated positions and velocities of both objects at that time, and the associated uncertainties represented by the propagated covariance matrices. The closer in time the TCA is, the more accurate these propagated estimates become, since the interval over which uncertainties grow is shorter. The quality of the original sensor data, whether from radar, optical, or other sources, has a direct impact on the initial covariance and, by extension, the final accuracy of the collision risk assessment. All this information is ultimately used to calculate the probability of collision, which forms the basis for deciding whether or not to perform an avoidance maneuver.

The 18th Space Defense Squadron serves as the hub for CDM distribution to space agencies around the world. While this is the common starting point, agencies do not rely solely on external data; instead, they enhance it with their own tracking capabilities, such as on-board GPS data, and propagate orbital states using internally developed high-fidelity orbit models, evaluating the resulting probability of collision. If this probability exceeds a predefined threshold, collision avoidance maneuvers are considered to reduce the risk.

Following the initial CDM, updates are typically issued several times a day, and over the course of a week leading up to the conjunction, a time series of CDMs is built up. As more data becomes available and the predicted encounter nears, the uncertainty in the estimated positions decreases, refining the understanding of the situation. The most recent CDM is considered the best available representation of the event, and if the predicted collision probability approaches or exceeds the agency's operational threshold, often around 1 in 10,000, planning for a potential avoidance maneuver begins. For many missions, especially in LEO, this threshold has become something of a default, though the actual value selected can depend on factors like mission criticality and acceptable risk. Additionally, a lower notification threshold, usually about one order of magnitude below the reaction threshold, is used to flag events early for monitoring. At ESA, the Space Debris Office, responsible for collision avoidance for a number of missions, begins alerting control teams and discussing maneuver options roughly two days before the expected conjunction, with final decisions often made about a day before the event.

CDMs gathered by the Space Debris Office during operations from 2015 to 2019 were compiled into a database of conjunction events, which later served as the foundation for the ESA Collision Avoidance Challenge [25]. The challenge emphasized the critical role of CDMs in understanding conjunction scenarios and illustrated the potential of machine learning (ML) methods to anticipate the evolution of collision probabilities using CDM time series [26].

Finally, we focus more specifically on how collision avoidance is performed at the European Space Agency. Thanks to a data-sharing agreement, ESA has access to CDMs released by the 18th Space Defense Squadron. Once received, these messages are automatically processed using ESA's internal software suite, CORAM. This includes two core tools: CORCOS, which evaluates the probability of collision using several analytical methods, and CAMOS, which helps in planning and optimizing possible avoidance maneuvers. The incoming data, along with results from risk analyses, are stored in a central database, forming the backbone of the overall system.

Another key part of ESA's approach is the creation of a so-called "mini-catalogue", a temporary local subset of orbiting objects that are in proximity to a given spacecraft. This is derived by propagating the state vectors from the CDMs using physical data from ESA's DISCOS object database. This mini-catalogue enables risk assessments to be performed directly against the spacecraft's actual trajectory, including planned maneuvers, without requiring constant re-screening from external sources.

Analysts then use a web-based user interface known as SCARF to monitor conjunctions, review risk metrics and visualize the geometries involved. This tool provides interactive dashboards, email templates for alerts, and 3D visualizations of close approaches. For maneuver planning, CAMOS allows for scenario testing and optimization, balancing objectives such as risk reduction, fuel economy, or required separations. The whole workflow is highly integrated and partially automated, allowing analysts to focus their attention only when truly needed [27].

1.3 Trajectory optimization

The number of active satellites is steadily increasing, and the deployment plans for mega-constellations such as Starlink, Eutelsat OneWeb, Project Kuiper, and others suggest that the satellite population will grow dramatically in the coming decades.

The impact of these mega-constellations is already evident in the number of CDMs generated. As of September 2020, when fewer than 1,000 Starlink satellites had been launched, Starlink alone accounted for 90% of the daily CDMs produced by the 18th Space Defense Squadron, equating to around 180,000 messages per day [28]. Since then, the situation has worsened significantly: as of April 2025, Starlink has 7,247 satellites in orbit [29], with plans to expand to as many as 40,000 satellites in the coming years.

At ESA's Space Debris Office, missions in LEO typically face several hundred potential conjunction events annually. Thanks to successive orbit updates, most of these cases are eventually downgraded and do not require action. Nevertheless, a typical Earth observation mission still needs to conduct one to two collision avoidance maneuvers each year [30].

SpaceX reports significantly higher numbers of CAMs, with the majority involving intra-constellation conjunctions between Starlink satellites themselves. In the first half of 2024, satellites in the Starlink mega-constellation performed nearly 50,000 CAMs, approximately double the number recorded in the previous six-month period. Notably, during the first four years following the initial Starlink launch, the number of evasive maneuvers consistently doubled every six months [31].

Given the growing density of objects in Earth's orbit, collision avoidance is poised to become an increasingly critical aspect of satellite operations. As a result, the development and continual improvement of collision avoidance systems, particularly the push toward achieving full automation, will be essential priorities in the coming decades. Alongside these efforts, advanced trajectory optimization techniques will be indispensable for ensuring that necessary maneuvers are performed efficiently.

One of the key challenges in this context is maintaining energy and fuel efficiency. Optimal fuel usage is crucial in space missions, where propellant is limited. Carefully planning CAMs helps prevent unnecessary fuel expenditure, thus extending the operational lifetime of the spacecraft.

Time constraints introduce even more complexity. In the presence of fast-moving debris or unexpected objects, there may be insufficient time to execute a low-cost, fuel-efficient maneuver. Optimization strategies must therefore accommodate urgent cases, shifting the focus from fuel savings to executing maneuvers within very tight time windows. Real-time decision-making becomes critical in these dynamic environments, requiring systems that can compute safe maneuvers rapidly.

Moreover, since science instruments are often turned off during avoidance maneuvers, leading to data outages, restoring the spacecraft to its operational orbit afterwards becomes a priority. Trajectory optimization algorithms must efficiently plan the return to the original or a sufficiently close orbit to minimize mission disruption and avoid excessive resource consumption.

Trajectory optimization clearly involves balancing multiple competing objectives. Collision avoidance maneuvers must simultaneously minimize fuel usage, ensure a rapid response, and restore the spacecraft's orbit with minimal deviation, all while respecting mission-specific constraints such as maximum allowable fuel expenditure and safe distances from other objects. Handling these multiple goals and constraints simultaneously results in an exceptionally complex optimal control problem.

The next part will review the strategies and approaches that have been explored so far to address these challenges.

1.4 State of the art for collision avoidance

Both classical and modern AI-based methods have been developed to mitigate the risk of collisions. Classical approaches rely on deterministic models and human intervention, while AI methods, particularly Reinforcement Learning (RL), enable real-time autonomous decision-making. This section presents a comprehensive overview of classical methods and AI-driven techniques, with detailed sections on Reinforcement Learning within the Markov Decision Process (MDP) and Partially Observable Markov Decision Process (POMDP) frameworks.

1.4.1 Classical approaches

Traditional approaches in collision avoidance often rely on optimization-based techniques that calculate avoidance trajectories using established mathematical frameworks.

Semi-analytical methods: Some authors have proposed gradient-based optimization. For example, the direction of the avoidance maneuver can be determined by finding the gradient of the collision probability with respect to the maneuver direction. Once the direction is determined, its magnitude can be found using numerical root-finding methods [32]. Such techniques, coupled with parametric searches, have seen successful software applications, such as in the CORAM tool of the ESA/ESOC Space Debris Office [33]. Other semi-analytical methods have involved transforming the problem into the combination of an eigenvalue problem and a non-linear equation [34]. Finally, researchers have tried to reduce the optimization problem to a Quadratically Constrained Quadratic Program, greatly simplifying the task, as in the OCCAM tool [35].

Particle Swarm Optimization (PSO): PSO is a nature-inspired optimization technique based on the collective behavior of groups. It is used to solve complex optimization problems by simulating the movements of a population of particles (potential solutions) within a defined search space. Multi-Objective Particle Swarm Optimizer can provide a set of Pareto-optimal CAMs, enabling operators to select the most appropriate maneuver based on constraints and goals [36].

Genetic Algorithms (GAs): GAs emulate the process of natural selection, iteratively refining candidate solutions. This approach has been applied to both LEO and GEO satellites, with the focus on maintaining satellite positioning within an established operational boundary [37]. Another study tried to deal with the complex scenario of multi-debris collision avoidance, addressing the competing objectives of collision probability and fuel consumption minimization [38]. Additionally, a GA-based optimization technique for the Brazilian Carcará-1 satellite was developed, emphasizing fuel conservation in CAMs, with penalties applied within the GA's fitness function when miss distances between the satellite and debris fell below a specified threshold [39].

Convex Optimization: The optimal control problem with obstacle avoidance constraints is inherently a non-convex problem, given the nonlinearities in the dynamics. A convexification of the optimization problem has been introduced, allowing it to be solved more efficiently and reliably. One method deals with the uncertainties in the knowledge of the positions of the objects, while introducing a collision avoidance constraint [40]. Further work introduced a multi-resolution approach for close-range trajectory planning, adaptively adjusting resolution based on proximity to target or debris to enhance computational efficiency and accuracy [41].

Model Predictive Control (MPC): MPC continuously updates control actions for a satellite based on predicted states over a finite horizon, allowing real-time obstacle avoidance adjustments. The optimization objective minimizes fuel consumption while reducing the arrival time to a target location. This approach has been applied to complex orbital environments with multiple dynamic obstacles [42].

1.4.2 Machine Learning approaches

ML techniques are increasingly pivotal in enhancing space debris avoidance systems by automating detection and decision-making processes. This section explores some ML methods, excluding Reinforcement Learning.

Bayesian Machine Learning: Bayesian ML techniques integrate uncertainty into collision risk assessments, enhancing predictions in the unpredictable context of space debris. Bayesian models, such as Bayesian Hidden Markov Models, can estimate collision probabilities by tracking the sequence of the probabilities in the CDMs over time [43]. An advantage of Bayesian methods is their adaptability. They enable real-time updates as new data from tracking stations or onboard sensors become available. By quantifying the likelihood of various collision scenarios, Bayesian ML supports informed decision-making, helping operators determine when avoidance maneuvers are necessary.

Neural networks: Neural networks, particularly deep learning (DL) models, have become invaluable for collision avoidance due to their ability to process and learn from vast and complex datasets. In collision avoidance, neural networks handle tasks like trajectory prediction, risk assessment, and debris detection by identifying patterns in historical and real-time data that might not be visible with traditional methods. One effective application is in Bayesian neural networks, which incorporate uncertainty estimation directly into their predictions. For example, "Kessler," an ML library for space applications, uses Bayesian neural networks to predict potential conjunction events. By quantifying the confidence in each prediction, Bayesian neural networks help prioritize high-risk events, allowing operators to make more informed, risk-based decisions [44]. Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) architectures are also commonly applied. CNNs are used for processing spatial data, such as images of debris fields captured by satellites, to detect and classify debris objects [45]. Meanwhile, LSTMs handle sequential data and are able to identify temporal patterns, making them suitable for predicting future conjunction events [46]. A recent hybrid approach combines Bi-LSTM and CNN architectures, optimized through Bayesian Optimization. The Bi-LSTM component captures long-range dependencies in sequential data, while CNN layers extract spatial features [47].

1.4.3 Reinforcement Learning approaches

Reinforcement learning within the Markov Decision Process and Partially Observable Markov Decision Process frameworks offers some of the most promising solutions for enabling spacecraft to detect and navigate collision threats. These frameworks are especially valuable for real-time decision-making, even in environments with limited observability and high-dimensional state-action spaces. By using MDP and POMDP models, RL can optimize both the safety and resource efficiency of autonomous operations.

The MDP and POMDP approaches differ primarily in their treatment of environmental observability. MDP-based models, which assume full observability, are well suited to scenarios where sensors provide comprehensive coverage. Conversely, POMDP-based models are designed for partially observable environments, where spacecraft must operate with limited or uncertain information, as is often the case in satellite navigation and guidance due to signal delays, sensor inaccuracies and environmental disturbances.

1.4.3.1 MDP-based Reinforcement Learning

MDP frameworks are applied in space debris avoidance by modeling the environment as fully observable, which allows RL agents to make deterministic, real-time decisions based on comprehensive state data. Concretely, this results in the assumption that the exact state of the spacecraft is known at every time step. This approach is particularly effective in applications where good quality data is available from sensors and tracking devices.

Proximal Policy Optimization (PPO): A PPO-based MDP approach has been employed to enable real-time trajectory optimization in dense debris fields [48]. Two key performance objectives are prioritized: minimizing the probability of colliding with multiple debris and conserving fuel. The approach penalizes high collision probabilities, with penalties that increase exponentially as the product of the collision probabilities with the single space debris increases, effectively guiding the spacecraft to maintain safe separation. Fuel use is also carefully managed, with a cost assigned to each thruster activation proportional to the maneuver's intensity. This setup minimizes excessive fuel consumption, making it suited for long missions where fuel must be conserved.

Deep Deterministic Policy Gradient (DDPG): A novel technique, called exploration-adaptive DDPG, has been applied to spacecraft proximity maneuvers, with the purpose of automating the computation of rendezvous trajectories, while considering collision avoidance constraints [49]. The final goal was developing a model for a unified controller, capable of guaranteeing autonomous spacecraft rendezvous together with collision avoidance. Another application has leveraged

DDPG to control a space manipulator for collision-free trajectory planning [50]. The goal was to enable a 3-DOF robotic arm mounted on a spacecraft to capture space objects while avoiding collisions in both *free-flying* and *free-floating* scenarios.

Grid Search and Cross Entropy: RL has also been used with a combination of Grid Search and Cross Entropy methods [51]. The system identifies optimal maneuver timings and directions based on probabilistic models of collision risk, fuel constraints, and mission requirements. Its architecture is designed to be adaptable, accommodating various optimization criteria like minimizing collision probability, fuel use, and trajectory deviation. This flexibility allows it to manage multiple potential collision events simultaneously, which is critical in high-density orbital regions.

1.4.3.2 POMDP-based Reinforcement Learning

In scenarios where partial observability is a factor, POMDP frameworks are employed to handle uncertainties by modeling the environment as belief states, probabilistic representations of the system's true state based on available observations. This is crucial for scenarios in which sensor data is incomplete or inaccurate.

Safe Reinforcement Learning (SRL): SRL, which focuses on optimizing the performance of an agent while ensuring that certain safety constraints are satisfied, has been integrated with POMDP models to manage collision avoidance in scenarios with a variable number of space debris [52]. In this approach, Penalized Proximal Policy Optimization (P3O) is used alongside LSTM networks, which enable the agent to process sequential data without changing the observation vector size, efficiently transforming the variable-length input from multiple space debris into a fixed-size vector representation. While the problem is not explicitly modeled as a POMDP, the approach shares similarities by focusing on managing partial observability using neural networks. The model incorporates constraints such as collision probability and energy efficiency while enabling the spacecraft to autonomously navigate dynamic and uncertain environments.

Deep Q-Network (DQN): Problem uncertainties and imperfect monitoring information have been addressed, in the POMDP framework, using a variation of DQN [53], relying on historical data rather than assuming perfect state observations. An LSTM layer maintains a "belief" of the environment's state by processing sequences of past observations and actions. A multi-objective optimization is sought, attempting to minimize collision probability, fuel consumption and trajectory deviations.

Chapter 2

Fundamentals of orbital mechanics and problem statement

This chapter introduces the fundamental principles of orbital mechanics that form the basis for the simulation framework and analyses developed in this thesis. It begins by outlining the reference frames adopted, along with the orbital elements and state vectors typically used to describe the position and velocity of a space object.

The discussion then proceeds with the classical Keplerian two-body problem, serving as a foundation for understanding orbital motion, before addressing the primary perturbations that influence satellite trajectories. These include gravitational and non-gravitational forces, which are incorporated into the dynamical models through a system of differential equations that govern orbital propagation.

The chapter concludes with a description of the modeling of the spacecraft's propulsion system and control actions, followed by the methodology for generating a synthetic database of close approaches and estimating collision probabilities, critical components in the Reinforcement Learning-based collision avoidance framework explored in the subsequent chapters.

2.1 Reference frames and orbital elements

2.1.1 Earth-Centered Inertial reference frame

One of the most commonly used reference frames (RFs) in spaceflight mechanics is the Earth-Centered Inertial (ECI) frame, illustrated in Figure 2.1 (adapted from [54]).

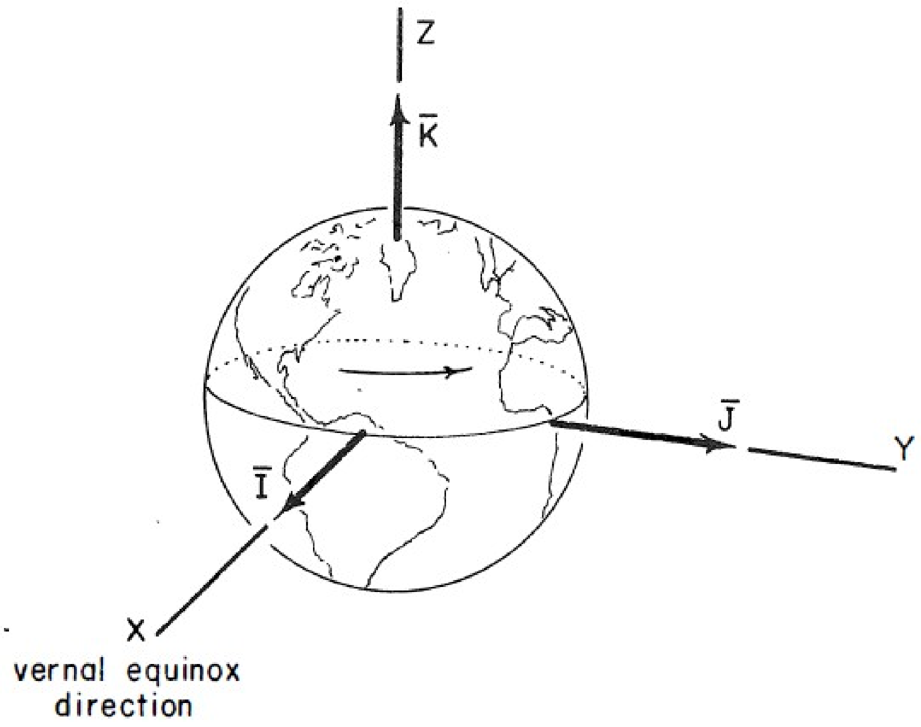

Figure 2.1: Earth-Centered Inertial reference frame

This reference frame is defined as follows:

- Origin is located at the Earth's center of mass.
- The **X-axis** points toward the vernal equinox direction at the epoch J2000 (January 1, 2000). This direction is defined by the intersection of the Earth's equatorial plane and the ecliptic plane (the plane of Earth's orbit around the Sun).
- The **Z-axis** is aligned with Earth's mean rotational axis at J2000, pointing toward the North Celestial Pole.
- The **Y-axis** completes a right-handed orthonormal coordinate system, lying in the equatorial plane and perpendicular to both the X- and Z-axes.

Although the ECI frame is not strictly inertial due to relativistic and precessional effects, it is generally treated as inertial for most practical applications, especially in low Earth orbit. This is because the ECI frame is fixed relative to the "fixed stars" - distant celestial objects whose positions appear constant over short timescales - and does not rotate with the Earth. Therefore, it provides a stable, non-rotating frame suitable for describing satellite motion.

2.1.2 Classical Orbital Elements

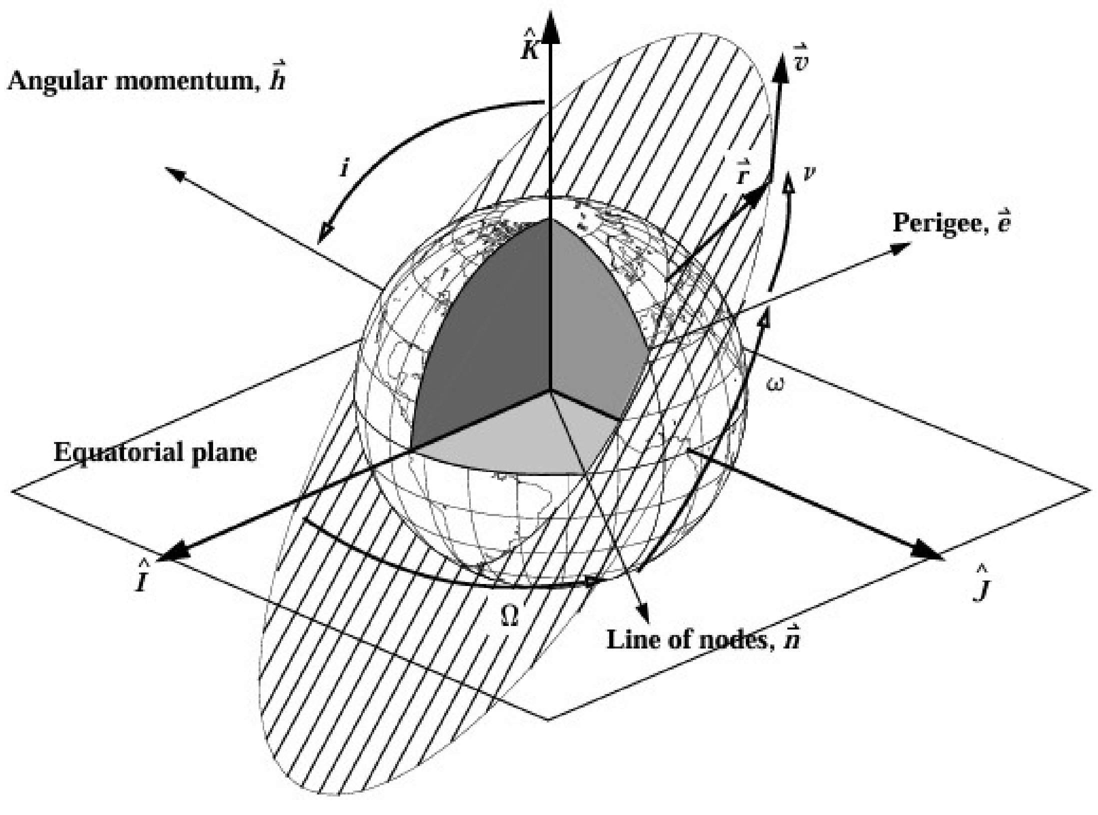

Figure 2.2: Classical Orbital Elements

The instantaneous position and velocity of a satellite in orbit can be fully described using a set of six classical orbital elements (COEs). These elements define the size, shape, and orientation of the orbit, as well as the position of the satellite along the orbit at a given time, and are displayed in Figure 2.2, extracted from [22].

The figure shows the typical configuration of an elliptical orbit of a secondary body (e.g., a spacecraft) around a primary body (the Earth, in this thesis). The primary body is located at one of the foci of the ellipse. The figure highlights both the equatorial plane, defined by the X and Y axes of the ECI frame, and the orbital plane, in which the satellite's motion occurs.

The intersection of the orbital and equatorial planes forms the line of nodes. The point where the satellite crosses the equatorial plane from south to north is the ascending node, while the opposite crossing defines the descending node. The angular momentum vector, obtained by the cross product of the position and velocity vectors in the ECI RF, is perpendicular to the orbital plane.

Also visible in the diagram are the periapsis and apoapsis - the points of minimum and maximum distance, respectively, between the orbiting satellite and the center of the primary body.

The six classical orbital elements are defined as follows:

- a (semi-major axis): Defines the size of the ellipse; half the distance of the major axis.
- e (eccentricity): Describes the shape of the orbit; ranges from 0 (circular) to 1 (parabolic).
- i (inclination): The angle between the orbital plane and the equatorial plane, or equivalently, between the angular momentum vector and the ECI Z-axis.
- Ω (right ascension of the ascending node): The angle from the ECI X-axis to the ascending node, measured in the equatorial plane.
- ω (argument of periapsis): The angle between the ascending node and the periapsis, measured in the orbital plane.
- ν (true anomaly at epoch): The angle between the periapsis and the position of the satellite at a given epoch.

2.1.3 State vector

In orbital mechanics, the state vector of a satellite refers to the set of its position and velocity at a specific moment in time, typically expressed in a chosen reference frame. The position vector defines where the satellite is relative to the center of the primary body, and the velocity vector describes its speed and direction.

Together, these two 3-dimensional vectors form a 6-dimensional state vector.

$$\mathbf{x} = \begin{pmatrix} \mathbf{r} \\ \mathbf{v} \end{pmatrix} \tag{2.1}$$

The six classical orbital elements offer an alternative, yet fully equivalent, representation of an object's orbital state. There exists a one-to-one correspondence between the orbital elements and the state vector (position and velocity). This means:

- Given a specific set of orbital elements, we can uniquely compute the corresponding state vector at any point along the orbit.
- Conversely, a known state vector can be used to uniquely determine the corresponding set of orbital elements that define the orbit.

When modeling a spacecraft, the state vector defined in equation 2.1 is augmented with an additional component: the spacecraft's mass. This is necessary because the spacecraft's mass changes over time due to engine firings, which consume propellant. Accurately tracking mass throughout the trajectory is essential, as it directly affects the equations of motion (see Section 2.2). Moreover, mass can be integrated alongside position and velocity at each time step, making the augmented state vector a natural and efficient choice for this purpose.

$$\mathbf{x} = \begin{pmatrix} \mathbf{r} \\ \mathbf{v} \\ m \end{pmatrix} \tag{2.2}$$

In contrast, the state vectors representing the various debris objects follow the form of equation 2.1 and do not include mass. This is because debris are assumed to be non-maneuvering objects, and their masses remain constant over time.

As a clarification, some ambiguity can arise regarding which object is at risk in a potential collision. In this work, we will use the term protected object or spacecraft (S/C) to refer to the active satellite with maneuvering capabilities. The term debris will be used as a general label for all potential threats, whether they are actual debris fragments or non-cooperative satellites, that pose a collision risk to the protected object.

The state vector can be represented in different reference frames depending on the context and application. In this thesis, the states of all space objects are consistently expressed in the ECI RF, resulting in the following form of the state vector:

$$\mathbf{x} = \begin{pmatrix} X \\ Y \\ Z \\ \dot{X} \\ \dot{Y} \\ \dot{Z} \\ m \end{pmatrix} \tag{2.3}$$

2.1.4 Radial–Transverse–Normal reference frame

Finally, it is necessary to introduce another reference frame, which has been used to express the thrust acceleration acting on the satellite: the Radial–Transverse–Normal (RTN) reference frame.

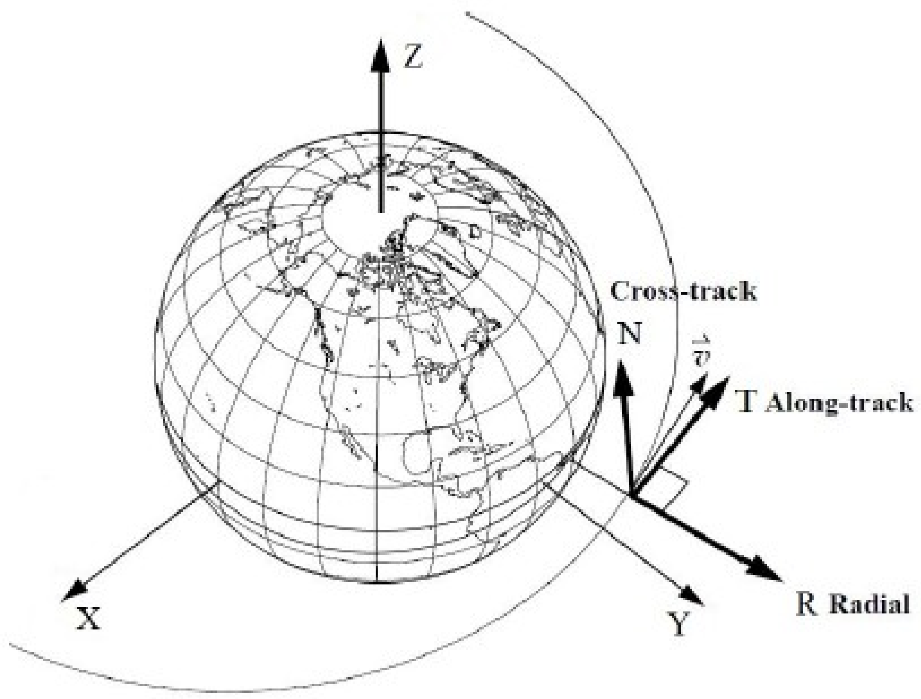

Figure 2.3: Radial-Transverse-Normal reference frame

The RTN frame, shown in Figure 2.3 (adapted from [55]), is a satellite-centered, right-handed orthonormal frame defined as follows:

- The origin is located at the satellite's position.
- The unit vector $\hat{\mathbf{h}}_1$ (radial) points in the direction of the satellite's position vector $\mathbf{r}$, from the center of the Earth to the satellite. It is computed as:

$$\hat{\mathbf{h}}_1 = \frac{\mathbf{r}}{\|\mathbf{r}\|} \tag{2.4}$$

- The unit vector $\hat{\mathbf{h}}_3$ (normal or cross-track) is aligned with the satellite's angular momentum vector $\mathbf{h} = \mathbf{r} \times \mathbf{v}$, and is computed as:

$$\hat{\mathbf{h}}_3 = \frac{\mathbf{r} \times \mathbf{v}}{\|\mathbf{r} \times \mathbf{v}\|} \tag{2.5}$$

- The unit vector $\hat{\mathbf{h}}_2$ (transverse or along-track) lies in the orbital plane and completes the right-handed triad:

$$\hat{\mathbf{h}}_2 = \hat{\mathbf{h}}_3 \times \hat{\mathbf{h}}_1 \tag{2.6}$$

Since thrust acceleration must be combined with other accelerations expressed in the ECI reference frame (as discussed in Section 2.2), it must be transformed from RTN to ECI. This is done via a 3x3 rotation matrix whose columns are the RTN unit vectors expressed in ECI coordinates. The matrix is constructed as:

$$\mathbf{R}_{\text{RTN}\to\text{ECI}} = \begin{bmatrix} \hat{\mathbf{h}}_\mathbf{1} & \hat{\mathbf{h}}_\mathbf{2} & \hat{\mathbf{h}}_\mathbf{3} \end{bmatrix} \tag{2.7}$$

This transformation matrix is time-dependent, and is computed using the satellite's current position and velocity vectors in the ECI frame.

2.2 Equations of motion

2.2.1 Two-body problem

The two-body problem is the simplest dynamical model used to describe the motion of two masses with respect to an inertial reference frame (assumed to be fixed relative to the fixed stars). This model relies on two key assumptions:

- Each body is treated as a point mass, with its entire mass concentrated at a single point (its center).
- The only force acting on the bodies is their mutual gravitational attraction.

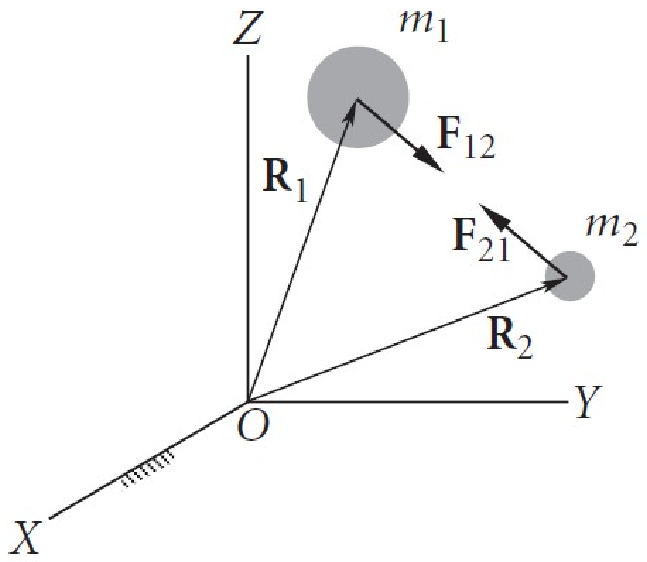

Figure 2.4: Two-body problem

In this scenario, shown in Figure 2.4 (extracted from [56]), the two bodies, of masses m_1 and m_2, are located at positions represented by the vectors R_1 and R_2, respectively. The gravitational force F_{12} is the force exerted on body 1 by body 2, while F_{21} is the force on body 2 due to body 1.

Given the assumptions, and since gravity is a central force (acting along the line connecting the two bodies), Newton's second law leads to the following equations of motion:

$$\mathbf{F_{12}} = \frac{Gm_1m_2}{r^2}\hat{\mathbf{r}} = m_1\ddot{\mathbf{R}}_\mathbf{1} \qquad \mathbf{F_{21}} = -\frac{Gm_1m_2}{r^2}\hat{\mathbf{r}} = m_2\ddot{\mathbf{R}}_\mathbf{2} \tag{2.8}$$

where

$$\mathbf{r} = \mathbf{R_2} - \mathbf{R_1} \qquad \hat{\mathbf{r}} = \frac{\mathbf{r}}{r} \tag{2.9}$$

and $G = 6.6743 \times 10^{-11} m^3 kg^{-1} s^{-2}$ is the universal gravitational constant. These equations also satisfy Newton's third law: the forces are equal in magnitude and opposite in direction ($\mathbf{F_{12}} = -\mathbf{F_{21}}$).

A further simplification of the two-body equations can be obtained by manipulating the expressions in equation 2.8). Specifically, multiplying the first equation by m_2 and the second by m_1, then subtracting the first from the second, leads to:

$$\begin{gathered} \frac{Gm_1m_2{}^2}{r^2}\hat{\mathbf{r}} = m_1m_2\ddot{\mathbf{R}}_\mathbf{1} \qquad -\frac{Gm_1{}^2m_2}{r^2}\hat{\mathbf{r}} = m_1m_2\ddot{\mathbf{R}}_\mathbf{2} \\ m_1m_2\left(\ddot{\mathbf{R}}_\mathbf{2} - \ddot{\mathbf{R}}_\mathbf{1}\right) = -G\frac{m_1m_2}{r^2}(m1 + m2)\hat{\mathbf{r}} \end{gathered} \tag{2.10}$$

Canceling out the common factor m_1m_2, and recalling the definition of the relative position vector $\mathbf{r} = \mathbf{R_2} - \mathbf{R_1}$ from equation 2.9, we obtain the simplified second-order differential equation:

$$\ddot{\mathbf{r}} = -\frac{G\left(m_1 + m_2\right)}{r^2}\hat{\mathbf{r}} \tag{2.11}$$

In the case examined in this thesis, a satellite orbiting the Earth, it is reasonable to assume that the inertial reference frame is Earth-centered (ECI), as introduced in Section 2.1.1. This assumption implies that $\mathbf{r} \equiv \mathbf{R_2}$. Additionally, because the mass of the Earth (m_1) is significantly greater than the mass of the satellite (m_2), we can approximate the total mass as m_1 alone. This leads to:

$$\ddot{\mathbf{r}} = -\frac{\mu}{r^2}\hat{\mathbf{r}} = -\frac{\mu}{r^3}\mathbf{r} \tag{2.12}$$

where

$$\mu = Gm_1 \simeq 3.986 \times 10^{14}\ m^3 s^{-2} \tag{2.13}$$

Equation 2.12 represents the fundamental expression for the gravitational acceleration experienced by a much smaller body (e.g., a satellite) under the influence of a significantly larger body (e.g., the Earth), in the context of the simplified two-body problem. The equation confirms that the resulting acceleration is purely radial, consistent with the nature of gravity as a central force.

In this thesis, the two-body acceleration is expressed in the ECI reference frame. The vector equation 2.12 can be expanded into its Cartesian components as follows:

$$\mathbf{a}_{2b} = \begin{pmatrix} -\frac{\mu X}{r^3} \\ -\frac{\mu Y}{r^3} \\ -\frac{\mu Z}{r^3} \end{pmatrix} \tag{2.14}$$

where $r = \sqrt{X^2 + Y^2 + Z^2}$.

2.2.2 Perturbations

In spaceflight mechanics, the two-body problem constitutes an idealized scenario in which an orbiting body moves solely under the influence of the gravitational attraction of a much larger central body. This model assumes that both bodies are point masses and that no other accelerations are present. Under these conditions, the motion is perfectly Keplerian, meaning that the first five orbital elements defined is 2.1.2 remain constant over time, while only the true anomaly evolves to describe the position along the orbit.

However, in reality, this idealization breaks down due to the presence of additional accelerations known as perturbations, which cause all the orbital elements to vary over time. Such deviations must be accounted for in order to accurately propagate the states of space objects over long time intervals. In this thesis, we focus on the primary perturbations relevant to low Earth orbit: gravitational effects due to the Earth's oblateness (J_2) and atmospheric drag. Other perturbing influences, such as higher-order gravitational harmonics, third-body effects and solar radiation pressure are not considered in the current analysis.

2.2.2.1 J_2 gravitational perturbation

Gravitational perturbations arise from the fact that the Earth is not a perfect sphere, but rather an oblate body with an uneven mass distribution. Specifically, the Earth's equatorial radius is slightly larger than its polar radius due to its rotation, resulting in a noticeable equatorial bulge. These irregularities in the Earth's shape and internal structure cause its gravitational field to deviate from the ideal point-mass model assumed in the two-body problem. The resulting perturbing forces can be modeled through a series of spherical harmonics, with each term representing a different aspect of the Earth's mass distribution. Among these, the J_2 term, associated with the Earth's equatorial bulge, is by far the most significant, especially for satellites in low Earth orbit.

This oblateness leads to perturbations in the orbit of a satellite, primarily affecting the orbital elements related to the orbit's orientation: the right ascension of the ascending node and the argument of periapsis. The J_2 perturbation induces a precession of the orbital plane and a rotation of the orbit on its plane, causing secular (long-term) variations in these elements, while the semi-major axis and eccentricity experience only small periodic changes.

The acceleration due to the Earth's oblateness can be derived from the gravitational potential expanded in spherical harmonics. Truncating the expansion to include only the J_2 term, and taking the negative gradient of the potential with respect to the Cartesian position vector yields the perturbative acceleration in Cartesian coordinates:

$$\mathbf{a}_{J_2} = \begin{pmatrix} -\frac{3}{2} J_2 \frac{\mu R_{eq}^2}{r^5} \left(1 - 5\frac{Z^2}{r^2}\right) X \\ -\frac{3}{2} J_2 \frac{\mu R_{eq}^2}{r^5} \left(1 - 5\frac{Z^2}{r^2}\right) Y \\ -\frac{3}{2} J_2 \frac{\mu R_{eq}^2}{r^5} \left(3 - 5\frac{Z^2}{r^2}\right) Z \end{pmatrix} \tag{2.15}$$

where R_{eq} is the equatorial radius of the Earth and $J_2 = 1.08263 \times 10^{-3}$.

2.2.2.2 Residual atmospheric drag

At low altitudes in low Earth orbit, typically below 800–1000 km, the Earth's residual atmosphere exerts a non-negligible aerodynamic force on orbiting satellites. This force, known as atmospheric drag, arises from collisions between the spacecraft and atmospheric particles, and acts opposite to the direction of motion relative to the atmosphere. Although extremely rarefied, the atmosphere at these altitudes is sufficient to cause a gradual decay in a satellite's orbit over time. The most pronounced effect of atmospheric drag is a secular decrease in the semi-major axis and orbital energy, which leads to orbital decay and ultimately re-entry if no corrective maneuvers are applied. It also tends to circularize the orbit by reducing eccentricity. Accounting for residual drag is essential when modeling long-term orbital dynamics in LEO.

The magnitude of the drag acceleration is strongly dependent on the ballistic coefficient, defined as:

$$B = \frac{C_D A}{m_0} \tag{2.16}$$

Here, C_D is the drag coefficient, which reflects the aerodynamic properties of the object and varies based on its shape and surface material, while A denotes the cross-sectional area exposed to the atmospheric flow. In this thesis, space objects are modeled as spheres. The cross-sectional area is estimated based on the known initial mass m_0, assuming an average internal density and accounting for the contribution of solar arrays.

Another key component in calculating the residual atmospheric drag is the velocity of the atmosphere, which is typically assumed to corotate with the Earth. Under this assumption, the velocity of the atmosphere at a given point is given by:

$$\mathbf{v}_{\text{atm}} = \boldsymbol{\omega}_{\text{Earth}} \times \mathbf{r} \qquad \mathbf{v}_{\text{rel}} = \mathbf{v} - \mathbf{v}_{\text{atm}} \tag{2.17}$$

where $\boldsymbol{\omega}_{\text{Earth}}$ is the angular velocity vector of the Earth, aligned with the planet's rotation axis and pointing toward the North Pole. This leads to the definition of the relative velocity $\mathbf{v}_{\text{rel}}$, which is the velocity of the spacecraft with respect to the surrounding atmosphere.

To compute the drag force, it is also essential to estimate the atmospheric density at the spacecraft's position. This can be modeled using an exponential density profile:

$$\rho = \rho_0 \exp\left(-\frac{r - R}{H_0}\right) \tag{2.18}$$

Here, R is the Earth's mean radius, ρ_0 is a reference density (typically at sea level), and H_0 is the scale height, representing the characteristic altitude over which atmospheric density decreases by a factor of e.

With these definitions, the perturbative acceleration due to atmospheric drag is given by:

$$\mathbf{a}_{\text{drag}} = -\frac{1}{2}\rho B \|\mathbf{v}_{\text{rel}}\| \mathbf{v}_{\text{rel}} \tag{2.19}$$

As with the two-body acceleration and the J_2 perturbation, the drag acceleration can be fully expressed in the ECI frame. By expanding the vector expression component-wise, we obtain:

$$\mathbf{a}_{\text{drag}} = \begin{pmatrix} -\frac{1}{2}\rho B v_{\text{rel}}(\dot{X} + \omega_{\text{Earth}} Y) \\ -\frac{1}{2}\rho B v_{\text{rel}}(\dot{Y} - \omega_{\text{Earth}} X) \\ -\frac{1}{2}\rho B v_{\text{rel}} \dot{Z} \end{pmatrix} \tag{2.20}$$

where $v_{\text{rel}} = \|\mathbf{v}_{\text{rel}}\|$ is the magnitude of the spacecraft's velocity relative to the rotating atmosphere.

2.2.2.3 Neglected perturbations

In addition to the dominant perturbations modeled in this thesis, there exist several other perturbative effects that can influence satellite motion. These include higher-order gravitational harmonics, which represent increasingly detailed deviations of Earth's gravity field from spherical symmetry. While the J_2 term accounts for the primary equatorial bulge, higher-degree terms (such as J_3, J_4, etc.) describe more subtle variations due to irregularities in Earth's mass distribution.

Other significant perturbations arise from third-body effects, primarily due to the gravitational influence of the Moon and the Sun, and from solar radiation pressure, which results from the momentum exchange between solar photons and the spacecraft surface. These forces can lead to long-term orbital changes, particularly in higher orbits.

However, in the context of low Earth orbit, the influence of these additional perturbations is generally negligible compared to the dominant effects of J_2 and atmospheric drag. For this reason, and to maintain a focus on the most relevant dynamics, these secondary perturbations are not considered in the present analysis.

2.3 Propulsion system and control variables

2.3.1 Propulsion system

One of the central components of this thesis is the modeling of the propulsion system and the thrust acceleration it generates. The objective is to replicate real-world behavior as accurately as possible, while maintaining a balance between fidelity and model simplicity.

Modeling a propulsion system requires the definition of several key parameters. First, the thrust F, which is proportional to the thrust acceleration and determines how quickly the spacecraft is able to change its velocity, and, consequently, its trajectory.

Other fundamental quantities are the effective exhaust velocity, denoted c, and the specific impulse I_{sp}, proportional to c. Both parameters represent a measure of engine efficiency.

$$c = \frac{F}{\dot{m}_P} \qquad I_{sp} = \frac{F}{\dot{m}_P g_0} \tag{2.21}$$

where $\dot{m}_P$ is the propellant mass flow rate.

Typically, engines that deliver high thrust exhibit low specific impulse, resulting in greater propellant consumption. Conversely, engines with high specific impulse tend to produce lower thrust.

In this thesis, the protected object is equipped with an ion thruster. An ion thruster operates by ionizing a neutral gas, typically xenon, extracting positively charged ions from the propellant through an electric field. These ions are then accelerated to high velocities using electrostatic or electromagnetic forces and expelled through a nozzle, producing thrust. Due to the low thrust levels, such propulsion systems require long firing durations to achieve significant ΔVs, making the assumption of impulsive maneuvers inapplicable.

Table 2.1 summarizes the parameters adopted for the propulsion system. The maximum thrust has been estimated based on the assumption that the spacecraft requires 100 mN of thrust for every 100 kg of mass.

Parameter	**Value**
m_0 [kg]	500
$F_{\max}$ [N]	0.5
I_{sp} [s]	1000
c [m/s]	9810

Table 2.1: S/C initial mass and engine performance

2.3.2 Control variables

The thrust vector is governed by three control parameters, which are outputs of the actor network described in Chapter 4. These parameters define both the direction and magnitude of the applied thrust.

The direction is specified by two angles: the elevation angle ϕ and the azimuth angle θ. Together, they determine the orientation of the thrust vector in the RTN frame introduced in Section 2.1.4. The magnitude of the thrust is regulated by the variable *engine*, which acts as a throttle by representing the fraction of maximum thrust F_{max} being applied.

These control variables are visualized in Figure 2.5, where the applied thrust magnitude is given by $F = engine \cdot F_{max}$.

The thrust components along the RTN axes can be computed by projecting the vector according to the angles ϕ and θ:

$$\begin{aligned} F_R &= engine \cdot F_{max} \cdot \cos(\phi) \cdot \cos(\theta) \\ F_T &= engine \cdot F_{max} \cdot \cos(\phi) \cdot \sin(\theta) \\ F_N &= engine \cdot F_{max} \cdot \sin(\phi) \end{aligned} \tag{2.22}$$

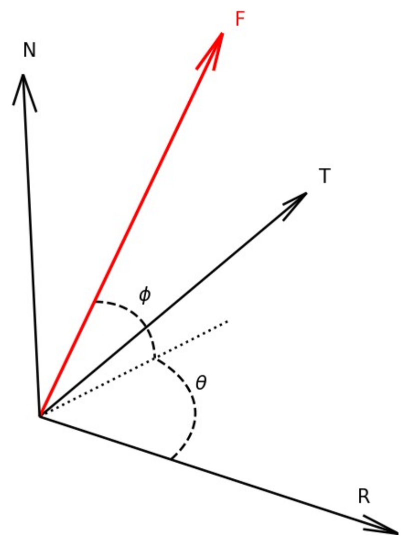

Figure 2.5: Thrust vector in the RTN frame

The corresponding acceleration components are obtained by dividing each thrust component by the spacecraft's instantaneous mass m:

$$\mathbf{a}_{\text{thrust}_{\text{RTN}}} = \begin{pmatrix} \frac{F_R}{m} \\ \frac{F_T}{m} \\ \frac{F_N}{m} \end{pmatrix} \tag{2.23}$$

Since the thrust acceleration must be combined with the gravitational and perturbative accelerations discussed in the previous section, the acceleration vector in RTN must still be rotated into the ECI frame using the rotation matrix defined in Section 2.1.4.

$$\mathbf{a}_{\text{thrust}_{\text{ECI}}} = \mathbf{R}_{\text{RTN}\to\text{ECI}}\ \mathbf{a}_{\text{thrust}_{\text{RTN}}} \tag{2.24}$$

In addition to determining the spacecraft's performance, the expressions in this section allow for the computation of the spacecraft's mass variation over time. Since the only source of mass loss is the expulsion of propellant, the propellant mass flow rate $\dot{m}_P$ is related to the time derivative of the spacecraft's mass m by:

$$\dot{m}_P = -\frac{dm}{dt} \tag{2.25}$$

Recalling the definition of the effective exhaust velocity, the derivative of the mass can be expressed as:

$$\frac{dm}{dt} = -\frac{F_{max} \cdot engine}{c} \tag{2.26}$$

This equation is integrated at each time step to update the spacecraft's mass throughout the trajectory.

2.3.3 Engine constraints

To more accurately simulate the behavior of a real ion thruster, an additional operational constraint has been introduced. As previously discussed, ion propulsion systems require extended firing durations to produce significant velocity changes due to their low thrust. However, prolonged operation can lead to overheating.

To account for this thermal constraint, the thruster in the simulation is subject to a duty cycle. Specifically, if the engine has been active for more than 20 minutes, it is assumed to have reached its thermal limit and is consequently shut down to prevent overheating. Once this threshold is crossed, the thruster must undergo a cooldown phase during which it remains inactive for a full orbital period. Only after this cooldown can the engine resume operation.

2.4 State transition and integration

The 7-dimensional state vector of the protected object is propagated at each time step using a state transition function, which integrates the time derivative of the state vector as defined in equation (2.27). This function takes as input the current state, the thrust vector (both its magnitude and direction), and the time increment. It then computes the updated state by numerically integrating the governing dynamics over the specified interval.

$$\dot{\mathbf{x}} = \begin{pmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \\ \ddot{X} \\ \ddot{Y} \\ \ddot{Z} \\ \dot{m} \end{pmatrix} = \begin{pmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \\ [\mathbf{a}_{2b} + \mathbf{a}_{J_2} + \mathbf{a}_{\text{drag}} + \mathbf{a}_{\text{thrust}_{\text{ECI}}}]_X \\ [\mathbf{a}_{2b} + \mathbf{a}_{J_2} + \mathbf{a}_{\text{drag}} + \mathbf{a}_{\text{thrust}_{\text{ECI}}}]_Y \\ [\mathbf{a}_{2b} + \mathbf{a}_{J_2} + \mathbf{a}_{\text{drag}} + \mathbf{a}_{\text{thrust}_{\text{ECI}}}]_Z \\ -\frac{F_{max} \cdot engine}{c} \end{pmatrix} \tag{2.27}$$

2.5 Database generation

While the use of a real database, such as Two-Line Element (TLE) data from real satellites tracked by the Space Surveillance Network and accessible via platforms like Space-Track, was considered, it was ultimately discarded due to the limited number of conjunction events available. This scarcity would hinder meaningful learning. Consequently, a synthetic database was generated to train the RL model, with an emphasis on ensuring a sufficient number of high-risk conjunction events characterized by small miss distances and non-negligible collision probabilities.

The approach is inspired by Bourriez et al. [53], with several modifications, and is detailed below:

1. **S/C initialization:** The spacecraft, serving as the protected object, is initialized using its classical orbital elements. These elements are subsequently converted into the Cartesian state vector in the ECI frame. An initial mass is also assigned to the spacecraft, which remains constant throughout the simulation, as no thrust is applied.

2. **Scenario definition:** A simulation time interval is established, along with a target number N of conjunction events. Each conjunction corresponds to a debris object whose TCA is defined within a window early in the simulation. This design ensures the S/C is undisturbed during the latter part of the simulation, allowing it time to return to its nominal orbit post-avoidance.

3. **S/C forward propagation:** The spacecraft is propagated forward over the full simulation interval by numerically integrating its equations of motion. Its state at each predefined TCA is recorded.

4. **Debris generation at TCA:** At each TCA, a debris object with random mass is generated by perturbing the S/C's position by 10 m in a random direction, while retaining its velocity. This introduces an initial layer of stochasticity to the database generation process.

5. **Debris backpropagation:** Each debris is backpropagated from its respective TCA to the initial time, and its initial state is recorded.

6. **Velocity perturbation:** To introduce a second layer of stochasticity, the velocity of each debris at the initial time is perturbed by 10^{-2} m/s in a random direction, while the position remains unchanged.

7. **Debris forward propagation:** The debris are then propagated forward over the entire simulation interval, and their state vectors are stored at each time step.

8. **Conjunction reassessment:** The relative position between the S/C and the N debris is computed at every time step. The minimum distances for each debris represent the miss distances, and the corresponding times lead to the definition of the new TCAs, typically different from the initial assigned ones because of the perturbations that were added.

The final database consists of:

- The initial state vector of the S/C (as subsequent actions, during learning, will cause deviations from the trajectory computed during database generation).

- The state vectors of all debris at every time step throughout the simulation duration.

Since debris are non-maneuverable and passive, their positions, velocities, and masses are assumed to be fully known over the entire duration of the simulation.

This generation procedure ensures the presence of N distinct conjunction scenarios, each with varying collision probabilities. These events are critical for training the RL model to learn avoidance maneuvers by maximizing the reward associated with reducing collision probability. The next section outlines the method used to compute this probability.

2.6 Collision probability estimation

The goal of this work is for a satellite to find the optimal trajectory to avoid space debris or other spacecraft. In order to do so, the probability of collision between two orbiting objects must be computed. This probability will then be a key term in the reward function of the RL agent, with the aim of minimizing the risk of collision. Due to the inaccuracies in the knowledge of the states of spacecraft and debris, which arise from instrument limitations as well as modeling errors (e.g., higher order gravitational perturbations, solar radiation pressure, and third-body effects have been neglected), the risk of collision can never be effectively zero and it is referred to as a probability. NASA guidelines, which have been applied, for instance, to the ISS and Space Shuttle programs, mandate that yellow warnings are raised for collision probabilities higher than 10^{-5} and red warnings are raised for probabilities higher than 10^{-4} [57]. The latter value has been considered as the reference threshold in this research, in order to ensure safe conjunction events. The computation of the collision probability between two objects in a conjunction event, hereinafter also referred to as P_c, is of primary importance, and depends on the position and velocity vectors of the objects and on the associated error covariance at TCA.

The position uncertainty is normally represented by a three-dimensional Gaussian distribution, and it has a graphical representation as an ellipsoid centered at the object. The shape of such ellipsoid is defined by a diagonal covariance matrix (assumption of independency), that gives an indication of the error associated with the measurement (x, y and z are local axes centered at the object).

$$\begin{pmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_z^2 \end{pmatrix}$$

Conjunction events are normally divided into two categories [58]:

- Linear relative motion (short-term conjunction): the relative velocity is considered sufficiently high to ensure a short-term encounter between the 2 objects. Therefore, the relative motion is linear and the position error, represented by the covariance matrix, can be considered constant.
- Non-linear relative motion (long-term conjunction): the relative velocity is low, leading to a long-term encounter. The magnitude and direction of the relative velocity vector both change over time, and the motion cannot be considered linear, leading to a more complex collision probability evaluation. Examples are formation flight and rendezvous.

Before digging into the specifics of the computation of the collision probability, the following assumptions, which lead to a simplification of the problem, have been made:

- All space objects are modeled as spheres, thus eliminating the need to consider the effect of their attitude.
- Their relative velocity at TCA is considered sufficiently high, leading to a short encounter, static covariance and linear relative motion (first category of conjunction event presented before).
- The positional errors of the objects are constant, independent of each other, have mean zero and are represented by Gaussian distributions.

Since the error covariance matrices are independent, they can be summed in order to obtain a single, large covariance ellipsoid centered on the primary object (the S/C in this case). In order to add flexibility to the procedure, the user, based on the expected uncertainties of the case, can define the size of the ellipsoid by specifying the axes in terms of $n\sigma$, with σ being the standard deviation in the Gaussian distribution of the position error and n being usually in the range 3-8. A sphere of radius R, representing the combined radii of the two objects, passes in the vicinity of the ellipsoid, and intersects it creating a “collision tube”. A collision happens if the relative distance between the two objects is smaller than their combined radii. The collision probability P_c is then equal to the volume integral of the three-dimensional probability density function (PDF) within the collision tube [59]. Thus, larger objects, as well as more uncertainties in the estimation of

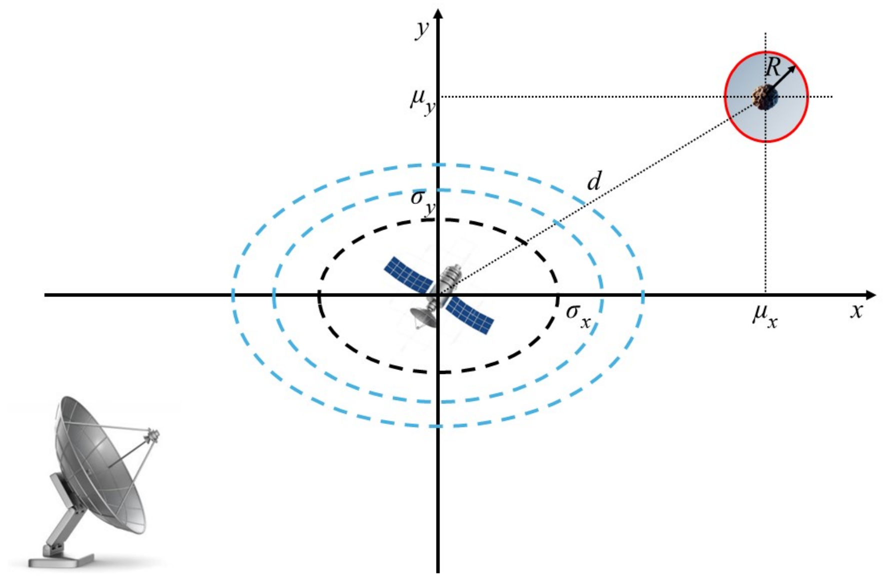

Figure 2.6: Collision probability estimation on the encounter plane

positions and velocities (which result in a larger ellipsoid), lead to higher collision probabilities.

It can be demonstrated that the same result can be obtained by taking the intersection of the ellipsoid and of the collision tube with the plane perpendicular to the relative velocity vector at the TCA, called encounter plane. On this plane, the ellipsoid becomes an ellipse, defined by the combined standard deviations σ_x and σ_y, and the collision tube becomes a circle of radius R, with center (μ_x, μ_y). Figure 2.6 provides the graphical depiction of this simplification on the encounter plane. Thus, the volume integral becomes the surface integral of the two-dimensional PDF on the area of the circle, as expressed by equation 2.28.

$$P_c = \iint_{(x-\mu_x)^2+(y-\mu_y)^2 \leq R^2} \frac{1}{2\pi\sigma_x\sigma_y} e^{-\frac{1}{2}\left(\frac{x^2}{\sigma_x^2}+\frac{y^2}{\sigma_y^2}\right)} dxdy \tag{2.28}$$

Different authors have tried to further simplify the computation. Foster [60], Patera [61, 62] and Alfano [63] give a numerical approximation of equation 2.28, while Chan [64] attempts to give an analytical approximation.

In this work, Chan's method has been employed. Under the assumption of two spherical objects and isotropic Gaussian position errors (i.e., the covariance matrix is proportional to the identity), Chan's method simplifies the two-dimensional Gaussian collision integral to a one-dimensional Rician integral.

Assuming equal positional uncertainty along the x and y axes (i.e., $\sigma_x = \sigma_y = \sigma$), the resulting formula becomes:

$$P_c = e^{-v/2} \sum_{m=0}^{M} \frac{v^m}{2^m m!} \left(1 - e^{-u/2} \sum_{k=0}^{m} \frac{u^k}{2^k k!}\right) \tag{2.29}$$

where

$$u \equiv \left(\frac{R}{\sigma}\right)^2, \quad v \equiv \left(\frac{d}{\sigma}\right)^2 \tag{2.30}$$

Equation 2.29, which converges after just a few terms, provides a way of computing the collision probability between two space objects as a function of only three terms: the combined radii R, the combined variance σ^2, and the miss distance d. The first two parameters are user-defined and depend on the physical size of the objects and the expected uncertainty in their positional estimates.

In this work, the standard deviation σ is estimated to emulate the actual process of covariance propagation, where the associated covariances increase over time. In particular, σ is modeled to vary linearly with the time to TCA, reflecting the fact that prediction uncertainty grows the further in time the event is projected.

The miss distance d has been found from the knowledge of the relative position and relative velocity vectors, exploiting the formula to find the shortest distance between 2 straight lines (thanks to the assumption of linear relative motion).

$$d = \left\| (\vec{r_1} - \vec{r_2}) \times \frac{\vec{v_1} - \vec{v_2}}{\|\vec{v_1} - \vec{v_2}\|} \right\| \tag{2.31}$$

with $\vec{r_1} - \vec{r_2}$ being the relative position vector and $\vec{v_1} - \vec{v_2}$ the relative velocity vector. In simpler terms, equation 2.31 expresses the shortest perpendicular distance from an object's current position to the line along which the other object is moving (defined by the relative velocity vector).

Figures 2.7 and 2.8 illustrate the behavior of equation 2.29, specifically showing how the collision probability varies with the miss distance. The plots are parameterized by either the combined radii or the combined variance.

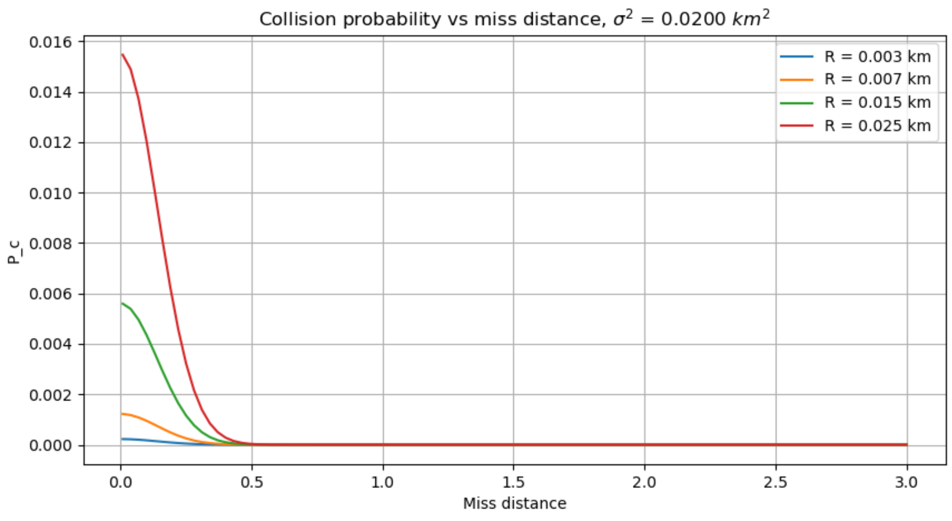

Figure 2.7: P_c as a function of d, parametrized by R

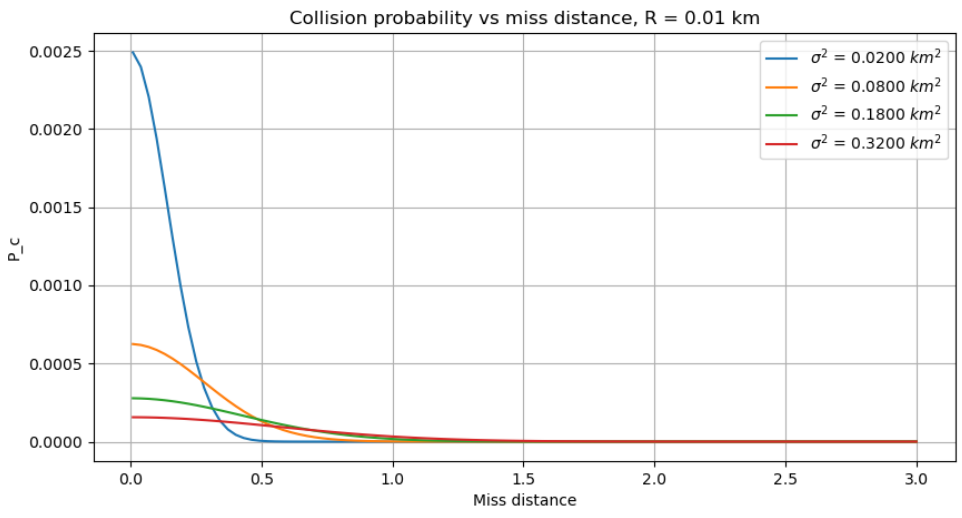

Figure 2.8: P_c as a function of d, parametrized by σ^2

The computation of the collision probability presented in this section represents the foundation for how the collision probability term will be computed in the reward function.

Chapter 3

Deep Learning techniques

The Reinforcement Learning architecture proposed in this thesis is based on various neural networks - Multilayer Perceptron, Long Short-Term Memory, and Transformer - each chosen for its suitability to different aspects of the learning task. This chapter provides a quick description of these neural networks.

3.1 Multilayer Perceptrons

3.1.1 Introduction

A Multilayer Perceptron (MLP) is one of the foundational architectures in neural networks, often used for tasks that require complex pattern recognition, such as classification and regression. MLPs consist of multiple layers of neurons connected in a feedforward manner, where each layer's neurons receive inputs from the previous layer and pass their outputs to the next layer. This layered structure enables MLPs to learn hierarchical representations of data, making them effective in capturing non-linear relationships that simpler models might miss [65]. An MLP is trained using supervised learning techniques, where the network learns a mapping from input features to a target output. This learning is achieved through a process known as backpropagation [66], a gradient-based optimization technique that minimizes the error by adjusting weights in the network.

3.1.2 Architecture of the MLP

An MLP (see Figure 3.1) consists of:

- Input layer: receives the input features.
- Hidden layer(s): contains neurons that perform computations on the inputs to extract features.

- Output layer: produces the final output of the network.

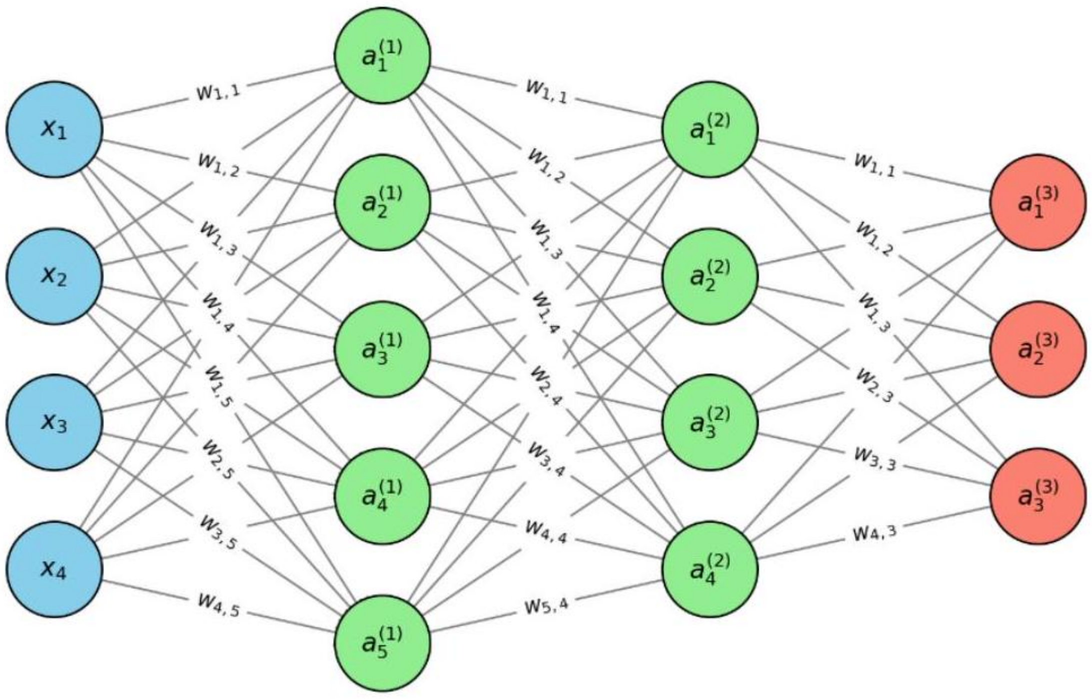

Figure 3.1: Example of MLP architecture

Each neuron j in the network computes an activation based on the weighted sum of its inputs. The output of neuron j in layer l is given by the following formula:

$$z_j^{(l)} = \sum_i w_{ji}^{(l)} a_i^{(l-1)} + b_j^{(l)} \tag{3.1}$$

where:

- $w_{ji}^{(l)}$ represents the weight between neuron i in layer $l-1$ and neuron j in layer l.
- $a_i^{(l-1)}$ is the activation of neuron i in the previous layer.
- $b_j^{(l)}$ is the bias term for neuron j in layer l.

The activation of neuron j in layer l, denoted as $a_j^{(l)}$, is obtained by applying a non-linear activation function f :

$$a_j^{(l)} = f\left(z_j^{(l)}\right) \tag{3.2}$$

Common choices for f include:

- Sigmoid: $f(x) = \frac{1}{1+e^{-x}}$, used to squash outputs to a range between 0 and 1 , often in binary classification tasks.
- Hyperbolic tangent (tanh): $f(x) = \tanh(x)$, which normalizes outputs between -1 and 1, commonly used in hidden layers for faster convergence.
- Rectified Linear Unit (ReLU): $f(x) = \max(0, x)$, commonly used for hidden layers to mitigate the vanishing gradient problem.
- Softmax: $f\left(x_j\right) = \frac{e^{x_j}}{\sum_k e^{x_k}}$, often applied to the output layer in multi-class classification tasks, as it converts outputs into probabilities that sum to 1.

The output layer's activations are then used to compute the network's predictions, either for classification or regression.

During the training of an MLP, an error function (or loss function) measures the discrepancy between the predicted outputs and the true target values. Minimizing this error is central to optimizing the MLP's performance. Two common error functions are:

- Sum of Squares (SSE): often used for regression tasks, the sum of squares error is calculated as:

  $$SSE = \sum_i \left(y_i - \hat{y}_i\right)^2 \tag{3.3}$$

  where y_i is the true target value and $\hat{y}_i$ is the predicted output for sample i. The SSE penalizes large errors more heavily and works well in cases where exact numeric predictions are required.

- Cross-Entropy: commonly used for classification tasks, especially with softmax outputs, cross-entropy error quantifies the difference between two probability distributions-the true distribution p and the predicted distribution q :

  $$H(p, q) = -\sum_i p\left(y_i\right) \log\left(q\left(y_i\right)\right) \tag{3.4}$$

For binary classification, this simplifies to binary cross-entropy, while for multi-class classification, it becomes categorical cross-entropy. Cross-entropy penalizes misclassifications more heavily, helping the model focus on distinguishing between different classes.

These error functions, combined with backpropagation, guide the learning process in an MLP by adjusting weights to minimize the error on training data, ultimately improving the network's predictive accuracy.

Depending on the choice of activation function in the output layer and the corresponding error function [65], the following applications are possible:

- Binary classification: sigmoid + cross-entropy for two classes.
- Multiclass classification: softmax + cross-entropy.
- Regression: linear activation function + sum of squares error function.

3.1.3 Backpropagation

Backpropagation is the key algorithm used to train an MLP [67]. It is based on the chain rule of calculus, which helps compute the gradient of the error with respect to each weight in the network. The steps in backpropagation are as follows:

1. Forward pass: compute the activations from the input layer through each hidden layer up to the output layer, yielding predictions.
2. Compute error: calculate the error E between the network's predictions and the actual target values.
3. Backward pass: propagate the error back through the network, calculating the gradient of the error with respect to each weight w. This involves:
   - Computing the gradient of the error with respect to the output layer.
   - Using the chain rule to backpropagate these gradients through each hidden layer [66].
4. Update weights: update the weights in the network using an optimization algorithm. A basic and widely used method is Stochastic Gradient Descent (SGD), which updates weights by moving them in the direction that reduces the error:

$$w_{ji}^{(l)} \leftarrow w_{ji}^{(l)} - \eta \frac{\partial E}{\partial w_{ji}^{(l)}} \tag{3.5}$$

where η is the learning rate, and $\frac{\partial E}{\partial w_{ji}^{(l)}}$ is the gradient of the error with respect to the weight.

Through iterative applications of these steps, the MLP learns to minimize its error on the training data.

While SGD is simple and effective, it can suffer from slow convergence and instability, particularly in the presence of noisy or sparse gradients. To address these limitations, this thesis employs the Adam optimizer (Adaptive Moment Estimation), introduced by Kingma and Ba [68]. Adam improves upon SGD by computing adaptive learning rates for each parameter using estimates of the first moment (the mean of the gradients) and the second moment (the uncentered variance of the gradients). Adam often leads to faster convergence and more stable training dynamics, especially in complex or high-dimensional optimization landscapes.

3.2 Recurrent Neural Networks

3.2.1 Introduction

Recurrent Neural Networks (RNNs) are designed for handling sequential data, where information at each time step is contextually linked to previous and/or future elements in the sequence. By retaining information across time steps, RNNs can capture these temporal dependencies, making them ideal for applications such as speech recognition, language processing, and time-series analysis. Unlike feedforward neural networks, RNNs have connections that form cycles, allowing information to persist through iterations. This persistence is maintained through a hidden state, an internal representation that is updated at each time step, carrying forward essential information about the sequence. The hidden state serves as a form of memory, storing knowledge about prior inputs, which is crucial for understanding and predicting sequential data.

3.2.2 Architecture of RNNs

In a simple RNN, see Figure 3.2, the architecture is structured so that each input at time t (denoted as x_t) affects not only the immediate output but also the next hidden state h_t. The hidden state h_t is updated at each time step using both the new input and the previous hidden state h_{t-1}:

$$h_t = f\left(W_x x_t + W_h h_{t-1} + b_h\right) \tag{3.6}$$

where:

- W_x and W_h are weight matrices associated with the input and hidden state, respectively.
- b_h is a bias term.
- f is typically a non-linear activation function, such as the hyperbolic tangent or ReLU.

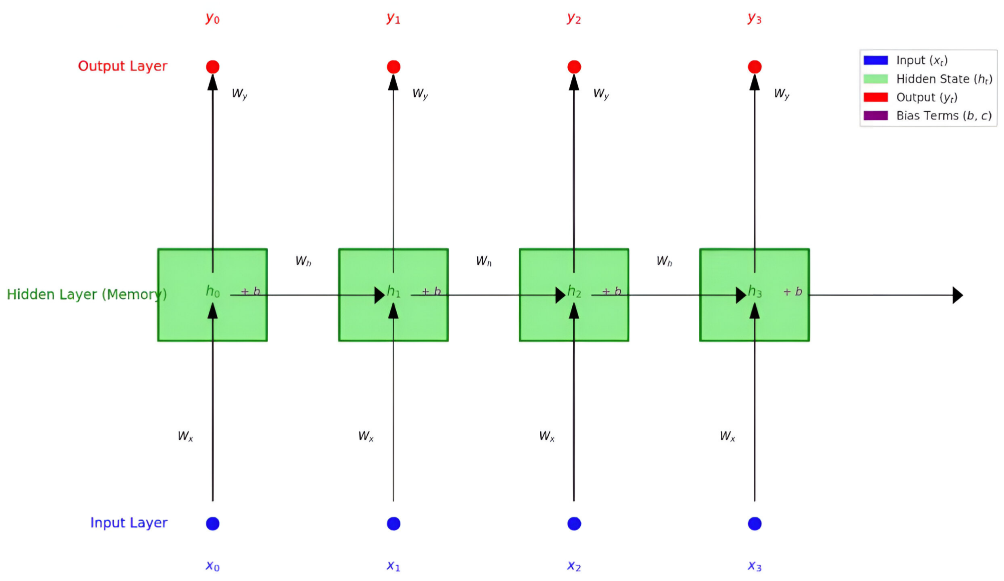

Figure 3.2: Vanilla RNN (spread in time)

The output y_t at each time step t is generally computed as:

$$y_t = g\left(W_y h_t + b_y\right) \tag{3.7}$$

where:

- W_y is the weight matrix that maps the hidden state to the output.
- b_y is the output bias term.
- g is often a softmax function for classification tasks.

The power of RNNs lies in this recursive structure, which enables them to learn context-dependent patterns in data. For instance, in language processing, each word's meaning is influenced by preceding words, a dependency that RNNs are well suited to model.

3.2.3 The vanishing gradient problem

Despite their utility, traditional RNNs face a major challenge known as the vanishing gradient problem. During training, the network's parameters are updated using backpropagation through time (BPTT), an extension of standard backpropagation applied to sequence models. BPTT calculates gradients for each parameter by recursively applying the chain rule over each time step in the sequence. However, as gradients are propagated back through many time steps, they can either shrink (vanish) or grow (explode) exponentially, depending on the eigenvalues of the weight matrices involved.

The vanishing gradient problem was highlighted by Bengio et al. [69], who showed that RNNs often struggle to learn patterns that require long-term dependency, thus limiting their applicability to short sequences. To address this, several approaches have been proposed, such as gradient clipping to control exploding gradients [70] and, more significantly, the development of specialized RNN architectures like Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs) [71].

3.2.4 Long Short-Term Memory

Long Short-Term Memory (LSTM) networks were introduced by Hochreiter and Schmidhuber in 1997 [72] specifically to tackle the vanishing gradient problem. LSTMs use a more complex cell structure than traditional RNNs, with a set of gates that regulate information flow, allowing them to retain long-term dependencies effectively. This architecture is especially beneficial for tasks that require the network to remember information over long sequences, such as machine translation, music generation, and time-series forecasting. The LSTM cell consists of three primary gates that control the flow of information into and out of the cell state (see Figure 3.3).

Forget gate: controls which parts of the previous cell state should be retained or discarded. This gate is crucial for "forgetting" irrelevant information, allowing the cell to focus on more pertinent aspects of the sequence.

$$\mathrm{F}_t = \sigma\left(W_f \mathrm{X}_t + U_f \mathrm{H}_{t-1} + b_f\right) \tag{3.8}$$

where σ is the sigmoid activation function, W_f and U_f are weights, and b_f is a bias term. The forget gate ensures that unnecessary information is not carried forward, thereby preserving essential patterns.

Input gate: controls how much of the new input is stored in the cell state. The input gate helps incorporate new information relevant to the sequence, helping the network learn contextually significant details.

$$\begin{aligned} \mathrm{I}_t &= \sigma\left(W_i \mathrm{X}_t + U_i \mathrm{H}_{t-1} + b_i\right) \\ \tilde{\mathrm{C}}_t &= \tanh\left(W_C \mathrm{X}_t + U_C \mathrm{H}_{t-1} + b_C\right) \end{aligned} \tag{3.9}$$

Here, I_t determines the amount of new information added to the cell state, and $\tilde{\mathrm{C}}_t$ is the candidate cell state that includes the new data to be remembered.

Cell state update: the cell state C_t at each time step combines retained and newly added information.

$$\mathrm{C}_t = \mathrm{F}_t \odot \mathrm{C}_{t-1} + \mathrm{I}_t \odot \tilde{\mathrm{C}}_t \tag{3.10}$$

where the symbol $\odot$ represents the Hadamard (elementwise) product. This mechanism allows LSTMs to maintain long-term information that is critical for learning sequential dependencies, even over lengthy input sequences.

Output gate: determines the final output of the LSTM cell, which is influenced by the updated cell state.

$$\begin{aligned} \mathrm{O}_t &= \sigma\left(W_o \mathrm{X}_t + U_o \mathrm{H}_{t-1} + b_o\right) \\ \mathrm{H}_t &= \mathrm{O}_t \odot \tanh\left(\mathrm{C}_t\right) \end{aligned} \tag{3.11}$$

The output gate controls which part of the cell state should contribute to the hidden state H_t, ensuring that the LSTM's output reflects relevant long-term and short-term information from the sequence.

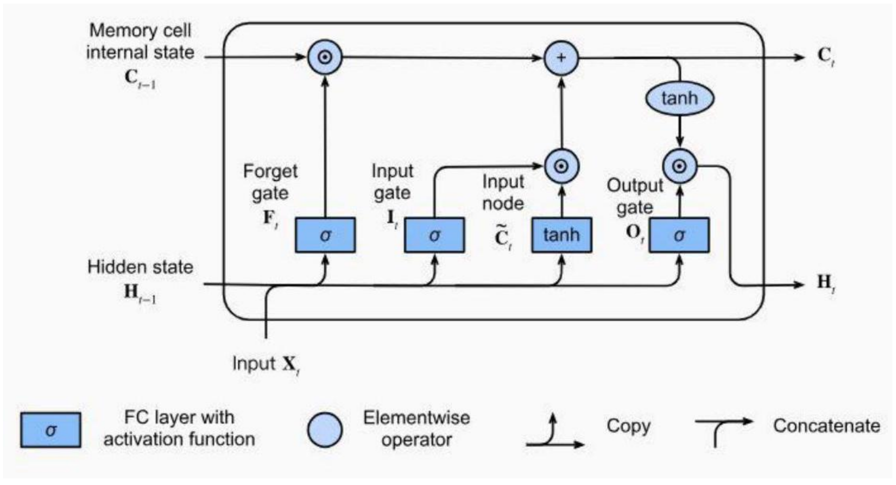
Memory cell internal state $\mathbf{C}_{t-1}$
$\mathbf{C}_t$
tanh
Forget gate $\mathbf{F}_t$
Input gate $\mathbf{I}_t$
Input node $\tilde{\mathbf{C}}_t$
Output gate $\mathbf{O}_t$
σ
tanh
Hidden state $\mathbf{H}_{t-1}$
$\mathbf{H}_t$
Input $\mathbf{X}_t$
FC layer with activation function
Elementwise operator
Copy
Concatenate

Figure 3.3: Architecture of an LSTM unit

Due to these gates, LSTMs can effectively handle long sequences without suffering from the vanishing gradient problem, making them highly successful in various sequential modeling tasks. Graves et al. [73] demonstrated the power of LSTMs for speech recognition tasks, while Sutskever et al. [74] showcased their effectiveness in machine translation.

3.2.5 LSTM for collision avoidance

Long Short-Term Memory networks are particularly effective for tasks involving sequential data with long-term dependencies, making them suitable for space debris avoidance applications where the trajectories and relative positions of debris over time are critical. In space debris avoidance, LSTMs can be used to process sequences of positional data, enabling autonomous spacecraft systems to predict and react to collision threats dynamically. This capability is especially valuable in complex orbital environments, where the number of potential debris threats may vary over time. In this case, an LSTM encoder can handle the variable-length sequence and produce a fixed-length summary vector, which encapsulates information about many debris [52].

Another important use of LSTMs is in feature extraction: they provide a structured representation of time series data by capturing temporal dependencies and maintaining a belief over the true state of the system through the integration of historical observations [53]. This concept will be explored further in the next chapter.

3.3 The Transformer Encoder (BERT)

3.3.1 Introduction

The Transformer architecture, introduced by Vaswani et al. [75], represents a breakthrough in deep learning for natural language processing (NLP) and other sequence-based tasks. The original Transformer model comprises two main components: the encoder and the decoder. The encoder is responsible for processing the input sequence and generating a set of context-aware representations, while the decoder uses these representations to produce the output sequence, typically in tasks such as machine translation.

In the encoder, a series of self-attention and feedforward layers transform the input into a sequence of embeddings that capture the relationships between tokens across the entire sequence. The decoder, meanwhile, uses these embeddings and applies an additional attention mechanism, called encoder-decoder attention, which allows it to focus on relevant parts of the encoder's output when generating each token in the output sequence. This encoder-decoder structure enables the Transformer to model both context within the input and dependencies between the input and output sequences, making it highly effective for translation, summarization, and other sequence generation tasks.

BERT (Bidirectional Encoder Representations from Transformers) focuses exclusively on the encoder part of the Transformer architecture, leveraging its powerful bidirectional self-attention mechanisms to understand contextual information within a sequence of text. By pretraining on large amounts of text data, BERT learns to generate rich, context-aware embeddings, which can be fine-tuned for various NLP tasks.

3.3.2 Self-attention mechanisms

At the core of the Transformer model is the self-attention mechanism, which enables each position in the input sequence to attend to (or focus on) other positions, regardless of their distance. This mechanism overcomes the limitations of sequential processing in RNNs by enabling the model to capture context across the entire sequence at once, allowing it to handle long sequences without losing information.

The self-attention mechanism operates through the following steps:

1. *Query, Key, and Value computations*: for each input token x_i in the sequence, the model computes three vectors: a query vector Q_i, a key vector K_i, and a value vector V_i. These vectors are derived through linear transformations applied to the input:

$$Q_i = x_i W^Q, K_i = x_i W^K, V_i = x_i W^V \tag{3.12}$$

   where W^Q, W^K, and W^V are learned weight matrices for the query, key, and value projections, computed through back-propagation (supervised learning).

2. *Attention scores*: the attention score between two tokens i and j is calculated by taking the dot product of Q_i and K_j, then scaling by the square root of the dimension d_k of the key vectors to avoid extremely large values. These dot products are then passed through a softmax function to generate attention weights, effectively turning them into probabilities.

$$\text{Attention } (Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.13}$$

   where $\frac{QK^T}{\sqrt{d_k}}$ represents the scaled dot-product attention [75, 76].

3. *Weighted sum of values*: the output for each token is computed by taking the weighted sum of all value vectors V, using the attention weights computed in the previous step. This process allows each token to incorporate contextual information from other tokens in the sequence.

   To further enhance the model's ability to capture different types of relationships between tokens, the Transformer uses multi-head attention. In multi-head attention, multiple attention heads are computed in parallel, each with its own set of learned weight matrices. The outputs from all heads are concatenated and linearly transformed:

$$\text{MultiHead}\,(Q, K, V) = \text{ Concatenate }\,(\text{head}_1, \ldots, \text{head}_h)\, W^O \tag{3.14}$$

   where each $head_i = \text{Attention}\,\left(QW_i^Q, KW_i^K, VW_i^V\right)$, and W^O is a learned weight matrix [75].

3.3.3 BERT architecture

BERT [77], developed by Devlin et al., builds upon the Transformer encoder architecture and applies it in a bidirectional manner to better understand language context. BERT is pretrained on large-scale text corpora using two main tasks: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). This bidirectional pretraining allows BERT to generate rich contextual embeddings for words by considering both left and right context.

The architecture of BERT consists of a stack of Transformer encoder layers, each comprising the following components:

1. *Input embedding*: the input tokens are first mapped to embeddings, which represent each word in a continuous vector space. Additionally, BERT adds positional encodings to these embeddings to inject information about the order of words, since Transformers lack inherent sequentiality:

$$\text{Embedding }\,(x_i) + \text{ PositionalEncoding }\,(i) \tag{3.15}$$

   The positional encoding for each position i in the sequence is given by:

$$PE(i, 2k) = \sin\left(\frac{i}{10000^{2k/d_{\text{model}}}}\right), PE(i, 2k+1) = \cos\left(\frac{i}{10000^{2k/d_{\text{model}}}}\right) \tag{3.16}$$

where d_{model} is the embedding dimension.

2. *Multi-head self-attention*: each token in the sequence attends to other tokens using the multi-head self-attention mechanism, as described above. This mechanism enables BERT to capture complex, bidirectional dependencies between tokens.

3. *Add & Norm*: the output of the attention mechanism is combined with the original input via a residual connection and then normalized to stabilize training:

$$\text{Add\& Norm}(X) = \text{ LayerNorm } (X + \text{Attention}(Q, K, V)) \tag{3.17}$$

   This normalization helps mitigate issues related to gradient vanishing/explosion [78].

4. *Feedforward Neural Network (FFN)*: the output from the Add & Norm layer is passed through a feedforward neural network, which typically consists of two linear transformations separated by a ReLU activation:

$$\text{FFN}(x) = \max\left(0, xW_1 + b_1\right) W_2 + b_2 \tag{3.18}$$

   This network enables the model to learn complex patterns in the input. The FFN is applied independently to each position in the sequence matrix.

5. *Add & Norm*: another residual connection and normalization are applied after the feedforward layer:

$$\text{Add \& Norm}(X) = \text{ LayerNorm } (X + \text{FFN}(X)) \tag{3.19}$$

   This structure is repeated across N layers, and each layer allows the model to progressively refine its understanding of the input sequence.

   Figure 3.4 illustrates the structure of each Transformer Encoder layer used in BERT. The model's ability to learn bidirectional relationships and capture complex dependencies has led to state-of-the-art performance in numerous NLP tasks.

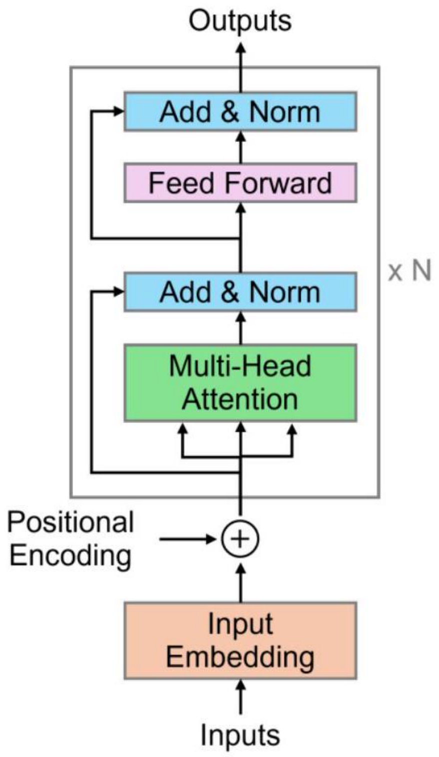

Figure 3.4: Architecture of the Transformer Encoder

3.3.4 Transformers for Reinforcement Learning

The Transformer architecture has gained traction in RL as a tool to tackle challenges such as long-term dependencies, high-dimensional state spaces, and partial observability. Traditional neural network architectures, such as CNNs and RNNs, while successful, often struggle with the complexity of RL tasks, particularly in capturing long-range dependencies. Transformers, with their self-attention mechanism and capacity for parallel processing of sequence data, are proving to be powerful in overcoming these limitations.

A Transformer expressively tailored for this problem is the Decision Transformer (DT) [79], which reframes RL as a conditional sequence modeling task. It predicts actions based on sequences of rewards, observations, and past actions (see Chapter 4). The model is trained using supervised learning, treating trajectories as examples. While DT has demonstrated strong performance on benchmark offline RL tasks, it is fundamentally limited by its reliance on high-quality, diverse offline datasets. Since it does not interact with the environment during training, DT struggles in settings where successful trajectories are rare, exploration is essential, or the dataset is suboptimal. This restricts its application in dynamic or safety-critical domains like autonomous control or space robotics.

Another notable application in the space domain has been the use of Transformers for trajectory optimization in spacecraft rendezvous missions. Guffanti et al. [80] demonstrated how the Transformer architecture can effectively learn policies for complex control tasks using offline Reinforcement Learning. In this context, the Transformer model warm-starts and enhances classical optimization-based methods by leveraging attention over sequences of past states, actions, and rewards, improving both sample efficiency and performance robustness over traditional methods.

3.4 Normalization and regularization in neural networks

Layer normalization normalizes inputs across all features within a single data sample. For each input to a layer, it computes the mean and variance over all the neurons in that layer for the given sample, and uses these statistics to rescale and shift the activations, an approach that has proven especially beneficial in Transformers.

Dropout acts as a regularization technique by randomly deactivating a subset of neurons during training, which prevents overfitting and encourages redundancy and robustness in learned features.

Weight decay, another regularization method, penalizes large weights by adding a term to the loss function proportional to the squared magnitude of the weights, promoting simpler models that generalize better to unseen data.

Together, these techniques contribute to the stability, efficiency, and generalization ability of neural networks. In the Transformer encoder, layer normalization and dropout are applied after the multi-head attention and feedforward sub-layers (see Figure 3.4), while weight decay is employed in the optimization of both the actor and critic MLPs.

Chapter 4

Reinforcement Learning framework

4.1 Fundamentals of Reinforcement Learning

Reinforcement Learning is a class of machine learning algorithms where an agent learns to make decisions by interacting with an environment. The agent's objective is to maximize cumulative rewards over time by taking actions that lead to favorable outcomes. Unlike supervised learning, where models are trained using labeled datasets, RL is characterized by the trial-and-error nature of the agent's learning process. The agent learns the best strategy (policy) through interactions with the environment and feedback in the form of rewards or penalties. The framework of RL is formalized using Markov Decision Processes, which define the environment in terms of states, actions, and rewards. This section provides an in-depth discussion of RL, covering its foundations, key concepts, Bellman equations, and an introduction to Deep Reinforcement Learning.

4.1.1 Story of Reinforcement Learning

The origins of Reinforcement Learning can be traced back to Edward Thorndike in the early 20th century. Thorndike's Law of Effect [81] proposed that responses followed by satisfying consequences are more likely to be repeated, while responses followed by unsatisfactory consequences are less likely to recur. This principle was foundational for developing reward-driven learning, a core idea that would later be formalized in computational models of decision-making.

In the 1980s, RL began to take its modern shape with Richard Sutton and Andrew Barto leading the development of temporal-difference learning. They formalized the RL framework by introducing value functions and Bellman equations, which provided a structured way to solve sequential decision-making problems. Sutton and Barto's 1998 book, Reinforcement Learning: An Introduction [82], formalized the concepts of value iteration, policy iteration, and the exploration-exploitation trade-off.

The most significant advancement in RL came in the 2010s, when Deep Q-Networks combined Deep Learning with RL. Introduced by Mnih et al. in 2015 [83], DQN used deep neural networks to approximate the action-value function, allowing RL to scale to environments with large, high-dimensional state spaces, such as playing Atari games directly from raw pixel inputs. The success of Deep RL has revolutionized the field, with DQN, PPO, and other algorithms being applied to complex tasks, including robotics, autonomous vehicles, and AlphaGo.

4.1.2 Agent-environment interaction

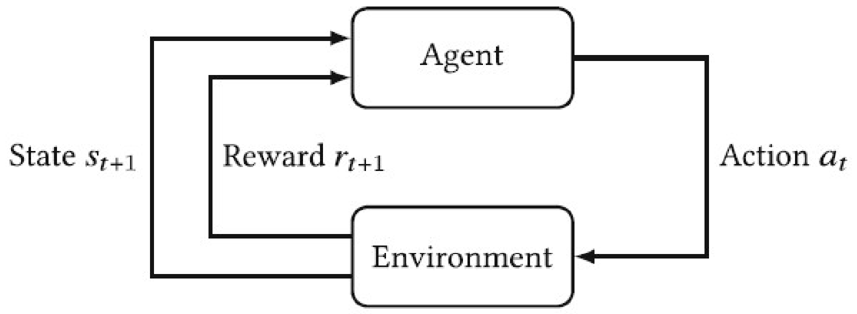

Figure 4.1: The agent-environment interaction

The agent interacts with its environment in a sequential decision-making process, where it must choose actions at each step to maximize its cumulative reward (see Figure 4.1, taken from [84]). This interaction is formalized as a Markov Decision Process, a mathematical framework used to model decision-making in situations where outcomes are partly random and partly under the control of an agent. An MDP is defined by the tuple (s_t, a_t, P, r_{t+1}, s_{t+1}):

- s_t: State of the environment at time t.
- a_t: Action performed by the agent at time t.
- $P(s_{t+1}|s_t, a_t)$: Transition function; probability of moving from state s_t to state s_{t+1} after taking action a_t.
- $r_{t+1}(s_t|a_t)$: Reward received by the agent after taking action a_t in state s_t.

The defining characteristic of an MDP is the Markov property, which states that the next state and reward depend only on the current state and action, not on the full history of previous states and actions.

The agent's goal is to find a policy $\pi(a|s)$, a mapping from states to actions, that maximizes the expected cumulative return. The return G_t is defined as the total discounted reward the agent will receive starting at time t:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{4.1}$$

where γ is the discount factor, controlling the relative importance of immediate versus future rewards. A small γ prioritizes immediate rewards, while a larger γ encourages the agent to consider long-term consequences. By maximizing G_t, the agent learns the optimal policy that leads to the most beneficial actions over time.

4.1.3 Value functions

To evaluate the performance of a policy, we consider the agent's interaction with the environment over time, which results in a trace, a sequence of states, actions, and rewards. Formally, a trace is a sequence of the form $(s_0, a_0, r_1, s_1, a_1, r_2 \ldots)$, generated by following a policy π in the environment. However, since both the environment and the policy can be stochastic, repeating the same policy may result in different traces. Therefore, rather than focusing on the return from a single trace, we are interested in the expected cumulative reward over all possible traces induced by a given policy.

The agent evaluates states and actions using value functions:

- The state-value function $V^\pi(s)$ represents the expected return from state s following policy π:
$$V^\pi(s) = \mathbb{E}_\pi[G_t|s_t = s] \tag{4.2}$$
- The action-value function $Q^\pi(s,a)$ represents the expected return from taking action a in state s, then following policy π:
$$Q^\pi(s,a) = \mathbb{E}_\pi[G_t|s_t = s, a_t = a] \tag{4.3}$$

4.1.4 Optimal policies

An optimal policy, denoted by $\pi^\star$, is a policy that yields the highest expected return from every state. More formally, a policy π is considered optimal if its state-value function $V^\pi(s)$ is greater than or equal to that of any other policy π' for all states $s \in \mathcal{S}$. All optimal policies share the same optimal state-value function, denoted by $V^\star$, defined as

$$V^\star(s) \doteq \max_\pi V^\pi(s) \tag{4.4}$$

and the same optimal action-value function, denoted by $Q^\star$, defined as

$$Q^\star(s,a) \doteq \max_\pi Q^\pi(s,a) \tag{4.5}$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$.

The Bellman optimality equations formalize the recursive structure of these optimal value functions. They express the idea that the value of a state, or state-action pair, under an optimal policy equals the expected sum of the immediate reward and of the discounted value of the next state, assuming the best possible action is taken thereafter.

These recursive relationships are central to computing optimal policies and they form the basis of dynamic programming methods such as value iteration and policy iteration.

4.1.5 Taxonomy of Reinforcement Learning methods

Reinforcement Learning methods can be broadly categorized based on their learning paradigms and how they represent and optimize decision-making strategies.

Online vs. offline learning:

- Online RL: The agent learns from interacting directly with the environment, updating its policy as it gains new experiences.
- Offline RL: The agent learns from a fixed dataset of past experiences, with no further interaction during the learning process.

Value-based methods: In value-based RL, the agent learns a value function that estimates the expected return from each state or state-action pair. Q-learning and SARSA are the most common value-based methods. These methods update the action-value function $Q(s,a)$ to find the optimal policy by selecting actions that maximize the value function.

Policy-based methods: In policy-based RL, the agent directly learns the policy function $\pi(a|s)$, which defines the probability of taking action a in state s. An example is the REINFORCE algorithm, which uses policy gradients to adjust the policy based on the observed rewards.

Actor-critic methods: Actor-critic methods combine both value-based and policy-based approaches. The actor updates the policy π, while the critic evaluates the chosen actions by computing the value function $V^\pi(s)$ or $Q^\pi(s,a)$. This combination allows the agent to learn both the optimal policy and the value of states or actions.

Proximal Policy Optimization, used in this thesis, is an online actor-critic algorithm that balances learning stability and policy improvement by constraining updates to the policy. Its robustness and efficiency make it one of the most widely used algorithms in Deep Reinforcement Learning.

4.2 Collision avoidance as an MDP

The problem of spacecraft navigation in a cluttered low Earth orbit-where the agent must simultaneously avoid collisions, restore its nominal orbit, and minimize fuel consumption-can be naturally formulated as a Markov Decision Process. In this setting, the spacecraft acts as the agent, while the environment comprises the dynamically evolving space populated by orbital debris.

At each discrete time step t, the agent observes a state s_t, representing information such as the spacecraft's position, velocity, and the relative configuration of surrounding debris. Based on this state, it selects an action a_t, drawn from a set of control inputs - namely, thrust magnitude (*engine*) and orientation angles (ϕ, θ), as defined in Section 2.3.2. The environment responds via a transition function, previously detailed in Section 2.4, which determines the next state s_{t+1} based on the agent's action and the system dynamics.

The spacecraft receives a scalar reward r_{t+1} from the environment, quantifying the desirability of its action with respect to multiple objectives: minimizing proximity to debris (collision avoidance), conserving fuel, and restoring orbital parameters. These reward signals guide the learning or optimization of a policy that balances safety and efficiency.

4.3 Reward function engineering

4.3.1 Purpose of the reward function

The reward function is a central element in Reinforcement Learning, as it constitutes the sole feedback mechanism available to the agent. It quantifies the desirability of the agent's actions by assigning numerical values based on their impact on the environment. Through the process of maximizing the expected return G_t, the discounted sum of future rewards, the agent is encouraged to learn an optimal policy $\pi^\star$, as discussed in Section 4.1.4.

In the specific case of spacecraft collision avoidance, the primary objective is to prevent collisions between the protected object and surrounding space debris. However, additional performance objectives are also critical, albeit sometimes conflicting with the primary goal. In this thesis, two such secondary objectives are addressed: minimizing fuel consumption and minimizing the deviation from the spacecraft's original operational trajectory.

The subsections below present the implementation of each reward component corresponding to these objectives. Each component has been formulated to produce only negative or zero values. This design choice ensures that the maximum achievable return is zero, which provides a clear and stable target for the learning process and facilitates convergence during training.

4.3.2 Collision probability term

While earlier works have employed simplified collision terms, such as assigning a penalty only upon actual collision [50], or applying a penalty inversely proportional to the miss distance [85], the spacecraft collision avoidance scenario presents additional complexities. As discussed in Section 2.6, it is fundamentally impossible to determine with certainty whether two space objects will collide, due to inherent limitations in sensor accuracy and uncertainties in orbital modeling. This requires the design of a reward function based on the estimated collision probability, which depends on factors such as the miss distance, the physical dimensions of the objects, and their associated covariance matrices (see equation 2.29).

Inspired by prior work on the topic [48, 52], a logarithmic formulation has been adopted for the collision penalty term in the reward function. Each debris object i is included in this term only if its estimated collision probability with the protected object, denoted by P_i, exceeds the threshold of 10^{-4} introduced in Section 2.6.

The collision penalty $r_{collision}$ is thus computed as:

$$r_{\text{collision}} = \begin{cases} \sum_i \left(4 - \log_{10}\left(\frac{P_i}{10^{-8}}\right)\right) & \text{if } P_i > 10^{-4} \\ 0 & \text{otherwise} \end{cases}$$

This formulation ensures that higher penalties are assigned to more threatening debris, while ignoring negligible threats. Figure 4.2 shows the trend of the reward term as a function of P_i.

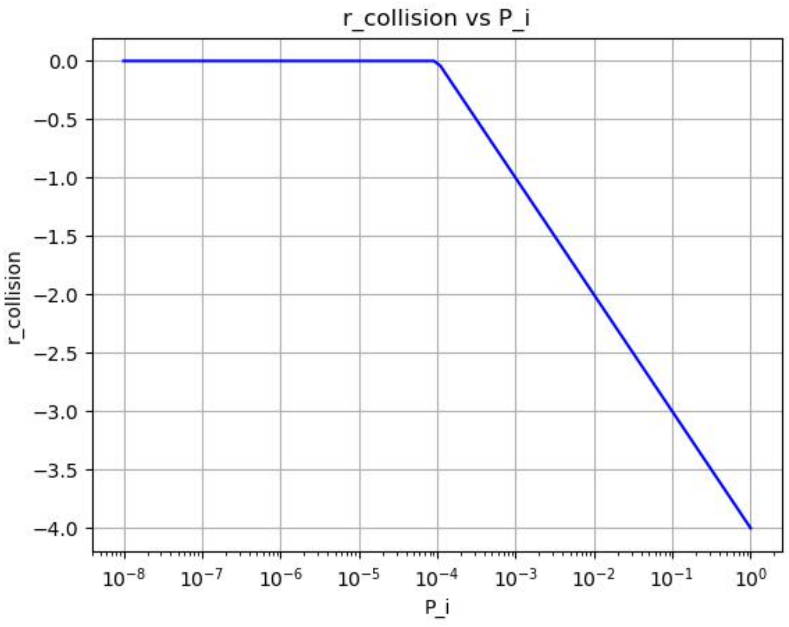

Figure 4.2: $r_{collision}$ as a function of P_i

4.3.3 Fuel efficiency term

Rather than penalizing the absolute magnitude of the thrust vector [86], or the fraction of the total available ΔV expended at each time step t [87], the fuel consumption penalty in this work uses the variable *engine*, which ranges from 0 to 1 and is directly proportional to the magnitude of the applied thrust (as defined in Section 2.3.2).

$$r_{fuel} = -engine \tag{4.6}$$

4.3.4 Trajectory deviation term

Collision avoidance maneuvers, as well as orbital perturbations, inevitably cause the spacecraft to deviate from its nominal operational orbit. Consequently, a restore phase is required to bring the S/C back to its reference trajectory.

Although perturbations induce natural variations in most orbital elements, including changes in the semi-major axis and eccentricity due to residual atmospheric drag, and shifts in the right ascension of the ascending node and argument of periapsis due to the J_2 gravitational effect (as discussed in Section 2.2.2), the restore phase is traditionally concerned only with the semi-major axis a, the eccentricity e, and the inclination i.

If perturbations are neglected, restoring the orbit serves solely to counteract the effects of the avoidance maneuver. However, when perturbations are present, this reward term also encourages the agent to perform station-keeping, thus compensating for the drift caused by the perturbative environment, even in the absence of avoidance events.

To prevent overcorrection, a buffer zone is defined around the nominal orbital parameters. Denoting the initial orbital elements with subscript 0, the trajectory deviation penalty is defined as:

$$\begin{aligned} r_{deviation} = -|a_t - a_0| \cdot 1_{\{|a_t - a_0| > 1\,\mathrm{km}\}} - 1000 \cdot |e_t - e_0| \cdot 1_{\{|e_t - e_0| > 2 \cdot 10^{-4}\}} \\ -10 \cdot |i_t - i_0| \cdot 1_{\{|i_t - i_0| > 10^{-1}\,\mathrm{deg}\}} \end{aligned} \tag{4.7}$$

Here, $1_{\{\cdot\}}$ denotes the indicator function, which equals 1 if the condition inside the braces is satisfied, and 0 otherwise. This ensures that each penalty term is applied only when the corresponding deviation exceeds its predefined threshold, effectively ignoring minor variations that fall within acceptable operational margins.

4.3.5 Total reward

The reward received by the agent at each time step is:

$$r_t = \alpha \cdot r_{collision} + \beta \cdot r_{fuel} + \delta \cdot r_{deviation} \tag{4.8}$$

The three positive constants must be accurately chosen in order to balance the competing objectives of the trajectory optimization scenario.

An additional concept has been introduced in order to achieve the right balance of the goals. The relative weights assigned to the different reward components are adapted dynamically depending on the maneuvering context. Specifically, the coefficient β, which scales the fuel penalty term r_{fuel}, retains its previous value if no maneuver is required (i.e., when both $r_{collision} = 0$ and $r_{deviation} = 0$. Otherwise, it is set to zero to prioritize safety or orbital correction.

$$\beta = \begin{cases} \beta_{\text{old}} & \text{if } r_{\text{collision}} = 0 \text{ and } r_{\text{deviation}} = 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.9}$$

Similarly, the coefficients α and δ, which weight the collision probability and trajectory deviation terms respectively, are adjusted based on whether a critical debris lies ahead. A debris is considered critical if its predicted time of closest approach with the satellite occurs within the next three hours and the associated collision probability is at least 10^{-4}. When no such threat is present, collision avoidance is irrelevant ($\alpha = 0$), and the agent focuses entirely on restoring or maintaining its orbit ($\delta = \delta_{old}$). Conversely, when a critical threat exists, 90% of the weight is assigned to the collision penalty and 10% to orbit restoration, thereby prioritizing safety while still applying a corrective influence to prevent excessive drift from the initial trajectory.

$$(\alpha, \delta) = \begin{cases} (0, \delta_{\text{old}}) & \text{if no critical debris ahead} \\ (0.9\,\alpha_{\text{old}},\ 0.1\,\delta_{\text{old}}) & \text{if critical debris ahead} \end{cases} \tag{4.10}$$

This adaptive scheme enables the agent to shift its focus between collision avoidance, fuel conservation, and trajectory recovery, depending on the evolving threat scenario.

4.4 Proximal Policy Optimization

Proximal Policy Optimization is a family of policy gradient algorithms that aim to optimize a stochastic policy in a manner that is both sample-efficient and stable. Introduced by Schulman et al. [88], PPO is conceptually derived from Trust Region Policy Optimization (TRPO) [89], which introduced the idea of a constrained policy update using a trust region. Instead of enforcing a strict constraint on policy changes as in TRPO, PPO uses a clipped surrogate objective to prevent excessively large updates. This clipping mechanism ensures that the new policy does not deviate too drastically from the old one, promoting stable training while avoiding the complexity of second-order optimization.

4.4.1 Main principles of PPO

At the heart of PPO lies the actor-critic architecture, which uses two separate neural networks: one for the actor (policy network) and one for the critic (value network). The actor is responsible for selecting actions according to a stochastic policy $\pi_\theta(a|s)$. In this thesis, the control parameters *engine*, ϕ and θ are sampled from the Gaussian distributions generated by the policy network at each time step. The critic, on the other hand, evaluates the quality of the current state by estimating the state-value function $V^\pi(s)$. These networks have distinct objectives and loss functions: the actor is trained to maximize the expected advantage-weighted probability ratio (with clipping for stability), and the critic is trained to minimize the mean squared error between predicted and actual returns. This dual-network architecture allows PPO to decouple action selection from value estimation, promoting more stable and robust learning.

PPO builds on the policy gradient framework and addresses the issue of instability in RL caused by excessively large policy updates. These abrupt changes can lead to what is known as "policy collapse", where the new policy becomes drastically different from the previous one, possibly resulting in catastrophic forgetting or convergence to a suboptimal behavior.

4.4.2 Actor loss and critic loss

The actor loss $L^{CLIP}(\theta)$, which depends on the weights θ of the actor MLP, is defined as the negative of the surrogate objective function:

$$L^{CLIP}(\theta) = -E_t\ [min(\rho_t(\theta)\hat{A}_t, clip(\rho_t(\theta),1-\epsilon,1+\epsilon)\hat{A}_t)] \tag{4.11}$$

where $\rho_t(\theta) = \dfrac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the importance sampling ratio between the new and old policies. The clip function bounds the ratio between $1-\epsilon$ and $1+\epsilon$ (with ϵ being called clip ratio), which effectively prevents large policy updates.

$\hat{A}_t$ is the advantage function, which estimates how much better (or worse) taking an action a_t in a state s_t is, compared to the expected value of the state under the current policy. Formally, it is defined as:

$$\hat{A}_t = Q(s_t, a_t) - V(s_t) \tag{4.12}$$

where $Q(s_t, a_t)$ is the action-value function and $V(s_t)$ is the state-value function. Intuitively, if $\hat{A}_t$ is positive, the action a_t was better than expected and should be encouraged; if negative, it was worse than expected and should be discouraged.

The advantage is typically computed using Generalized Advantage Estimation (GAE) [90]:

$$\hat{A}_t = \sum_{l=0}^{\infty}(\gamma\lambda)^l\delta_{t+l} \tag{4.13}$$

where γ is the discount factor, λ is the GAE parameter controlling the bias-variance trade-off, and δ_t is the temporal difference error given by:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{4.14}$$

where r_t is the reward at time step t, $V(s_{t+1})$ is the estimated value of the next state and $V(s_t)$ is the estimated value of the current state.

The critic loss $L^{VF}(w)$, or value function loss, which depends on the weights w of the critic MLP, is defined as:

$$L^{VF}(w) = E_t[(V_w(s_t) - V_t^{target})^2] \tag{4.15}$$

This loss represents the mean squared error between the predicted value $V_w(s_t)$, which is computed by the critic and is also used in the estimation of the advantage, and the target value V_t^{target}, where V_t^{target} is computed using the discounted sum of rewards.

To prevent excessive shifts in the critic's predictions during training-which could destabilize advantage estimation-PPO implementations often apply value function clipping, which constrains the update of $V_w(s_t)$ within a small range relative to its previous value.

4.4.3 Total PPO loss and entropy

In PPO, the total loss combines contributions from the policy (actor), the value function (critic), and the entropy. The total loss is defined as:

$$L_{PPO} = L^{CLIP}(\theta) + c_1 L^{VF}(w) - c_2 E_t[H[\pi_\theta(\cdot|s)]] \tag{4.16}$$

The total loss guides the update of the weights of the MLPs of the actor and critic:

$$\begin{aligned} \theta_{new} &= \theta_{old} - \eta \nabla_\theta \ L_{PPO}(\theta, w) \\ w_{new} &= w_{old} - \eta \nabla_w \ L_{PPO}(\theta, w) \end{aligned} \tag{4.17}$$

where η is the learning rate, in this case the same for both the actor and critic networks.

In continuous action spaces where π_θ is modeled as a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$, the entropy has the analytical form:

$$H[\pi_\theta(\cdot|s)] = \frac{1}{2}\log(2\pi e \sigma^2) \tag{4.18}$$

which depends directly on the mean of the standard deviations of the actions. In the total loss (equation 4.16), this entropy term appears with a negative sign. Therefore, minimizing the total loss implicitly promotes the maximization of entropy. This mechanism encourages the policy to maintain stochasticity during the initial stages of training, mitigating the risk of premature convergence to suboptimal deterministic behaviors. As training progresses, however, the policy becomes increasingly confident in its action choices, which is reflected by a reduction in variance. Consequently, the entropy naturally decreases in the later phases of training.

4.4.4 Rollout and mini-batch construction

Training in PPO relies on collecting a batch of experiences through a process called rollout. A rollout consists of sequences of observations, actions, rewards, dones (episode termination flags), value estimates, and log-probabilities of actions taken under the old policy. These data tuples are stored in a structure commonly referred to as the rollout buffer. Each step during the rollout includes storing the current state s_t, the action a_t sampled from the policy $\pi_{\theta_{old}}$, the log-probability of the action, the received reward r_t, a terminal signal, and the value prediction $V(s_t)$ from the critic network. Once the entire rollout is complete, which typically spans a fixed number of time steps, GAE is applied to compute the advantage estimates $\hat{A}_t$ for each timestep.

The buffer also computes the target values V_t^{target} for critic training using the discounted sum of rewards:

$$V_t^{target} = \sum_{l=0}^{T-t-1} \gamma^l r_{t+l} + \gamma^{T-t} V(s_T) \tag{4.19}$$

where T is the rollout horizon and γ the discount factor. Once all advantages and values are calculated, the entire buffer is shuffled. PPO then performs several epochs of optimization over mini-batches, which are subsets randomly sampled from the full rollout buffer. This approach allows the model to reuse data multiple times, enhancing sample efficiency while keeping the training stable thanks to the clipped objective function. The data are iterated through multiple epochs, and after each mini-batch the weights and biases of the actor and critic networks are updated by backpropagation.

4.5 Deep POMDP

4.5.1 Definition and motivation

In Reinforcement Learning, Markov Decision Processes (MDPs) provide a formal framework where an agent interacts with an environment fully observable at each timestep. An MDP is defined by the tuple (s_t, a_t, p, r_{t+1}, s_{t+1}), as described in Section 4.1.2.

In many real-world applications, however, the agent cannot fully observe the environment. This gives rise to Partially Observable Markov Decision Processes (POMDPs). A POMDP extends an MDP by including a set of observations Ω and an observation function O, forming the tuple (S, A, P, r_{t+1}, Ω, O). The elements of the tuple are:

- S: Set of possible environment states (hidden from the agent).
- A: Set of actions available to the agent.
- $P(s_{t+1}|s_t, a_t)$: Transition function; probability of moving from state s_t to state s_{t+1} after taking action a_t.
- $r_{t+1}(s_t|a_t)$: Reward received by the agent after taking action a_t in state s_t.

- Ω: Set of possible observations the agent can receive.
- $O(o_{t+1}|s_{t+1}, a_t)$: Observation function; probability of observing o_{t+1} after transitioning to state s_{t+1} using action a_t.

The key distinction is that in a POMDP the true state s_t is hidden, and the agent must infer it based on the observation history. This leads to the concept of a belief state $b_t(s_t)$, which is a probability distribution over possible states given the past observations and actions.

4.5.2 Classical belief state estimation

In classical POMDP theory, the belief state is updated using Bayesian filtering:

$$b_{t+1}(s_{t+1}) = \eta \cdot O(o_{t+1}|s_{t+1}, a_t) \sum_{s \in S} P(s_{t+1}|s_t, a_t) \cdot b_t(s_t) \tag{4.20}$$

where η is a normalization constant.

Other classical belief estimation methods include the Kalman filter for linear Gaussian systems, the Extended Kalman filter and Unscented Kalman filter for nonlinear systems, and the Particle filter for highly nonlinear or non-Gaussian models.

These methods are tractable only for small state spaces or simple dynamics. Hence, modern RL employs deep networks for belief estimation.

4.5.3 Deep belief estimation

Before exploring specific architectures, it is worth noting how the transition from the belief as a probability distribution to a vector representation is justified. In classical formulations, the belief state $b_t(s_t)$ is a distribution over the state space S, which can be high-dimensional or continuous. Storing and manipulating such distributions explicitly is intractable in most real-world settings. Therefore, deep learning approaches approximate the belief state using a fixed-dimensional vector $\hat{b}_t$, learned from historical observations and actions. This embedding captures the sufficient statistics of the history relevant to future decision-making and can be trained together with the policy and value networks.

In deep RL, the belief state is often approximated using recurrent or attention-based neural networks.

4.5.3.1 RNNs for deep POMDPs

Recurrent neural networks and their variants such as LSTMs or GRUs are widely used. The hidden state h_t of the RNN implicitly tracks the belief:

$$h_t = RNN(o_t, h_{t-1})$$

The policy and value function are then conditioned on h_t:

$$\pi(a_t|h_t), \quad V(h_t)$$

The policy effectively becomes a mapping from the hidden state of the RNN, best representation of the actual state, to the set of possible actions.

While RNNs handle partial observability well for short temporal contexts, they suffer from vanishing gradients and memory limitations over long sequences.

4.5.3.2 Transformers for deep POMDPs

Transformers have emerged as powerful alternatives due to their ability to capture long-range dependencies via self-attention mechanisms.

In particular, a sliding window of the last L tokens (which derive from the observations) can be fed to the Transformer encoder:

$$[x_{t-L+1}, \dots, x_t]$$

The Transformer produces output embeddings $[z_{t-L+1}, \dots, z_t]$. The belief representation $\hat{b}_t$ can be computed either as the last output token z_t, as a mean or max pooling over all z_i, or as a special learned token (e.g. [CLS] token in BERT).

This $\hat{b}_t$ is then used by the policy and value function:

$$\pi(a_t|\hat{b}_t), \quad V(\hat{b}_t)$$

4.5.4 Proposed architecture

Figure 4.3 illustrates the architecture proposed in this thesis for integrating belief estimation with PPO in a deep POMDP setting.

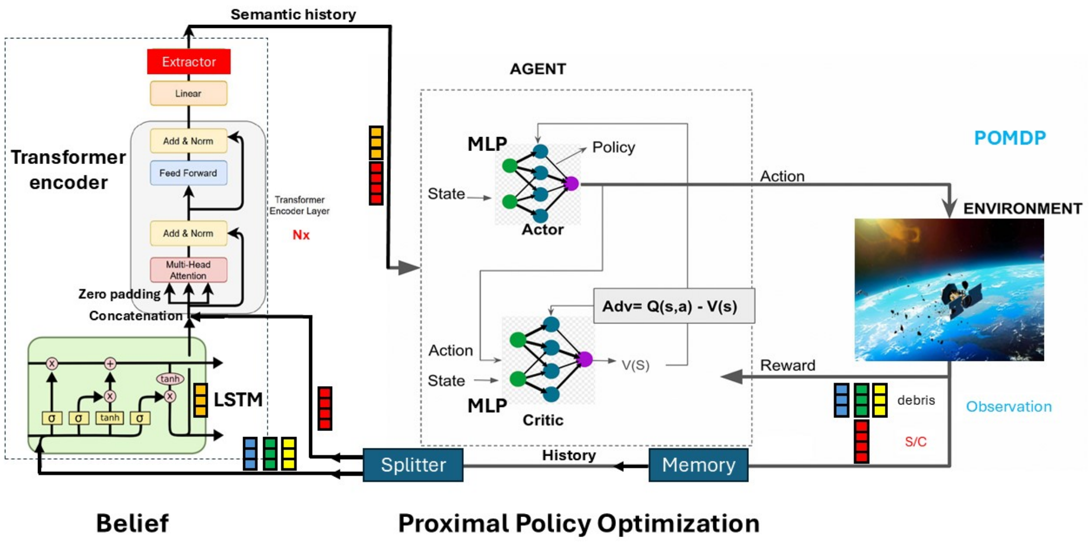

Figure 4.3: Deep POMDP PPO architecture

In this architecture, the agent makes decisions based on partial observations of the environment, such as in spacecraft collision avoidance, where sensing limitations prevent precise knowledge of the position and velocity of space objects. To address this, the architecture maintains a belief over the environment's state, updated using a temporal sequence of observation vectors. After that sequence is memorized, for an entire mini-batch, the procedure is as follows, and it is performed for all the time steps in the mini-batch:

1. The observation vector, of size $7 + 6N$, where N is the number of debris (see Section 2.1.3), is split into the 7×1 vector representing the S/C state and the N separate 6×1 vectors for each debris.

2. The 6×1 debris vectors are processed by an LSTM, which summarizes the set of debris into a single fixed-length vector, represented by the hidden state of the last LSTM layer. The output dimension matches the size of the hidden state.

3. This summary vector is concatenated with the S/C vector to produce a fixed-size representation of the full environment.

4. Before applying positional encoding, each input vector is first projected to a higher-dimensional space to enrich its representational capacity. The resulting sequence is then processed by N_x layers of a Transformer encoder. To handle

variable-length input during inference, particularly at the beginning of each episode, when the mini-batch is gradually assembled, a buffer with zero padding is used to ensure uniform input length. After Transformer encoding, the output is projected back down to a 13-dimensional vector.

The encoded representation of the environment is then passed to the PPO agent. During training, the entire mini-batch of Transformer outputs is used, while during inference only the final output vector, corresponding to the current time step, is retained. This final vector is treated as the approximated belief state $\hat{b}_t$. Based on this belief, the actor MLP produces a probability distribution over the available actions, and the critic MLP estimates the state-value function.

Building on the work of C. Mu et al. [52], the inclusion of the LSTM in the architecture is motivated by the need to produce a fixed-length input for the Transformer. Although LSTMs are typically employed for time series data, their ability to encode sets of variable-length, time-independent vectors makes them suitable here. By leveraging its memory cells and gating mechanisms, the LSTM compresses the variable number of debris vectors into a fixed-size summary. This allows the downstream Transformer to receive a consistent input shape, avoiding architectural modifications when the number of debris changes.

4.5.5 Training considerations

It is important to highlight that training the POMDP architecture involves estimating all network parameters via backpropagation. The total loss, defined in Equation 4.16, is first propagated through the actor and critic networks, and subsequently through the Transformer and LSTM modules. After all gradients have been computed, the network weights are finally updated using the Adam optimizer.

Moreover, another important distinction is that, when the Transformer is included in the architecture, the rollout buffer is not shuffled before training. Indeed, the Transformer needs ordered temporal sequences to learn the dynamics of the simulation scenario.

Chapter 5

Results

5.1 Simulation scenario and initial conditions

The simulation scenario consists of one protected object and ten debris. To introduce variability in the dataset, the physical properties of the debris are randomly sampled from uniform distributions. Table 5.1 summarizes the physical parameters of both the spacecraft and the debris.

The total cross-sectional area, which is necessary for computing the residual drag acceleration, is calculated as the sum of the area of the solar arrays and the projected area of the main body. The latter is derived under the assumption that the object is a uniform-density sphere, using its mass and internal density ρ.

Parameter	**S/C**	**10 debris**
m_0 (initial mass) [kg]	500	$\mathcal{U}(200,\ 500)$
ρ (density) [kg/m^3]	1500	1500
A_{arrays} [m^2]	5	$\mathcal{U}(3,\ 5)$
A_{body} [m^2]	$\pi\left(\frac{3m_0}{4\pi\rho}\right)^{2/3}$	$\pi\left(\frac{3m_0}{4\pi\rho}\right)^{2/3}$
A (total cross-sectional area) [m^2]	$A_{\text{body}} + A_{\text{arrays}}$	$A_{\text{body}} + A_{\text{arrays}}$
C_D (drag coefficient)	2.2	2.2

Table 5.1: Physical properties of the spacecraft and debris

Since collision avoidance is particularly relevant in low Earth orbit, the protected spacecraft is placed in a typical LEO orbit. The initial classical orbital elements of the spacecraft are presented in Table 5.2.

Element	Value
a [km]	7000
e	0.01
i [°]	0.1
Ω [°]	90
ω [°]	90
ν [°]	5

Table 5.2: Initial classical orbital elements of the spacecraft

Two datasets have been generated for training and evaluation purposes. The first includes only the unperturbed dynamics, while the second incorporates full orbital perturbations, specifically the Earth's oblateness (J_2) and residual atmospheric drag. As the debris properties are randomly sampled, their mass and cross-sectional area differ across the two datasets.

Figures 5.1 and 5.2 show the trend of the distance between the spacecraft and the ten debris. In both datasets, TCAs are marked by vertical red dashed lines. The presence of multiple close approaches in each dataset has been deliberately ensured, following the procedure outlined in Section 2.5, to generate non-negligible collision probabilities in the absence of active avoidance maneuvers.

The total simulation time is 30 hours. Every training or evaluation episode starts from the same initial conditions described above. However, the duration of each episode is generally shorter and can be adjusted by modifying the hyperparameters, as explained in the next section.

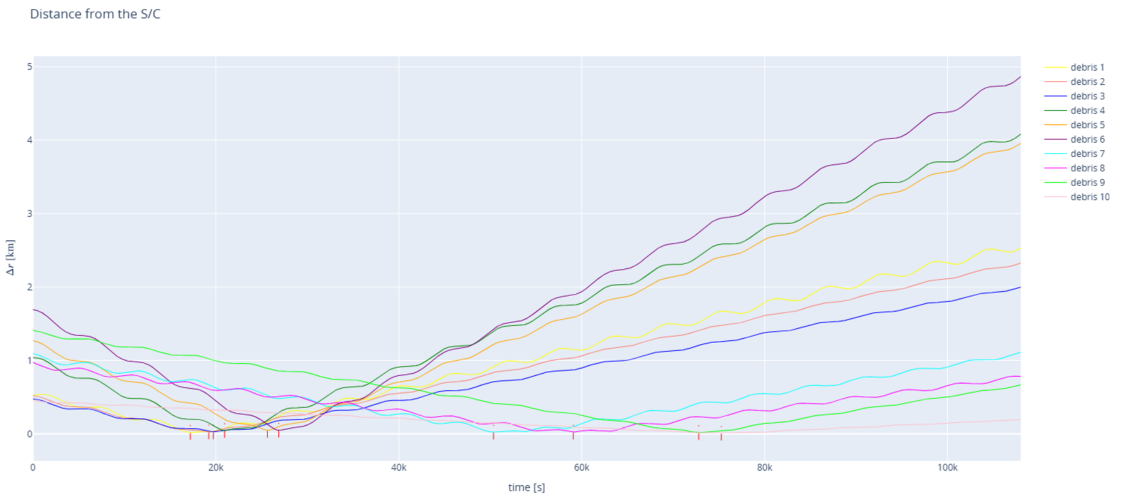

Figure 5.1: Database generated without orbital perturbations

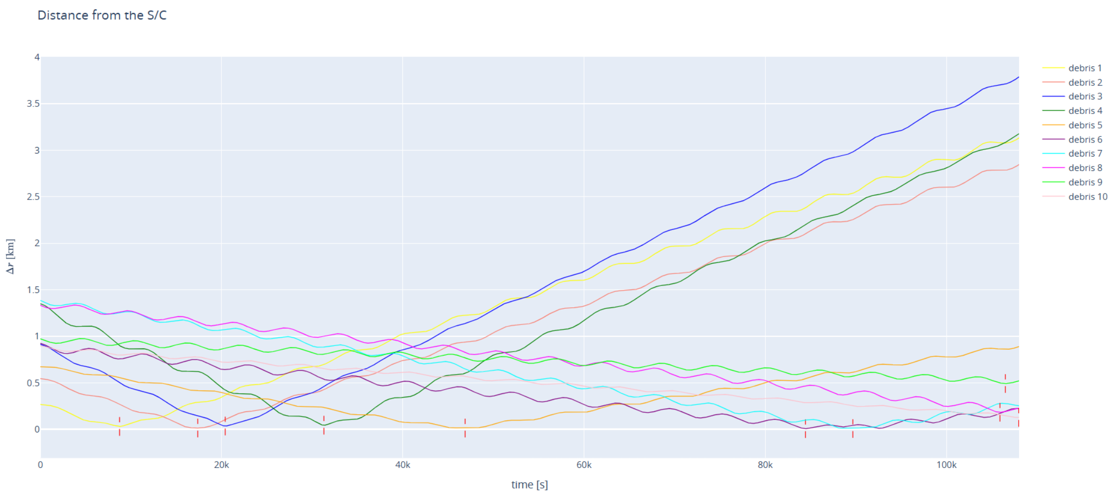

Figure 5.2: Database generated with orbital perturbations

5.2 Policy architecture and hyperparameters

The implementation of PPO used in this thesis is based on the Stable Baselines3 library [91]. This framework provides a convenient and flexible interface for configuring training parameters and supports the integration of custom architectures, including LSTM and Transformer-based feature extractors. The following section details the main hyperparameters employed and provides a rationale for their selection.

5.2.1 Simulation parameters for training

Parameter	**Value**
`max_episodes`	500
`k_ep`	40
`batch_size`	32
`incr_t` [s]	60
`episode_length` [s]	`k_ep` × `batch_size` × `incr_t`
`n_steps`	`k_ep` × `batch_size` × 5
`total_timesteps`	`k_ep` × `batch_size` × `max_episodes`

Table 5.3: Simulation parameters used for training

Table 5.3 summarizes the main parameters governing the training phase of the simulation. The parameter `max_episodes` defines the upper limit on the number of episodes that can be executed during training. After each episode concludes, the spacecraft and debris are reset to the same initial conditions described in the previous section.

The duration of a single episode is determined by the product of three quantities: `k_ep`, the number of mini-batches per episode; `batch_size`, the number of time steps in each mini-batch; and `incr_t`, the time increment per simulation step (in seconds). This product yields the episode length in seconds. It is essential that this duration remains shorter than the total simulation time span defined in the database, to ensure consistency with the generated orbital data.

The choice of `batch_size` significantly impacts the training dynamics, as it influences the value of `k_ep` and ultimately determines the number of weight updates per rollout. A batch size of 32 was selected as a compromise between computational efficiency and learning stability. Smaller values can lead to noisy gradient estimates and unstable updates, while larger batches tend to oversmooth the gradients, potentially slowing down convergence. Empirically, batch sizes in the range of 32–64 are widely used in PPO for their balance between these effects.

The parameter `incr_t` must strike a balance between computational efficiency and control flexibility: a larger value reduces the computational burden but restricts the frequency of thrust updates, effectively "freezing" the control input over longer intervals; a smaller value increases resolution but also the computational load.

The parameter `n_steps` defines the total number of simulation time steps collected in one rollout buffer, as detailed in Section 4.4.4. In the present configuration, a rollout consists of five episodes, which are subsequently divided into mini-batches for gradient updates. Finally, `total_timesteps` specifies the maximum number of time steps allowed during training. This limit is rarely reached, as training is typically terminated earlier by an early stopping criterion.

5.2.2 Simulation parameters for evaluation

Parameter	**Value**
`eval_freq`	`k_ep` × `batch_size` × 5
`n_eval_episodes`	1

Table 5.4: Simulation parameters used for evaluation

Table 5.4 summarizes the parameters used during evaluation (also referred to as testing). The parameter `eval_freq` specifies how often evaluation is triggered, expressed in environment time steps. In this setup, evaluation is conducted every 5 training episodes, allowing the agent sufficient time to learn between evaluations. Although not strictly required, the evaluation frequency has been deliberately chosen to match the length of a rollout buffer. As a result, testing occurs immediately after a complete rollout has been collected and training has been performed on it.

The parameter `n_eval_episodes` defines how many test episodes are executed during each evaluation. This enables a more robust assessment of the policy's performance, typically by averaging the undiscounted return across evaluation episodes.

Running multiple evaluation episodes is especially useful in the presence of stochasticity, either in the policy or the environment. However, in this case, both the policy and the environment are deterministic: actions are selected as the means of the Gaussian distributions (rather than sampled), and the same database is consistently used across episodes. As a result, repeated evaluation episodes yield identical outcomes.

5.2.3 PPO-specific and learning parameters

Parameter	**Value**
`actor_network`	[150, 150]
`critic_network`	[100, 100]
η	10^{-5}
`n_epochs`	4
c_1	0.5
c_2	0.005
γ	0.995
λ	0.95
ϵ	0.25
`std_init`	0.25
`weight_decay`	10^{-5}
`clip_range_vf`	0.2
`max_grad_norm`	0.2

Table 5.5: PPO-specific and learning parameters

Table 5.5 lists the hyperparameters specific to the PPO algorithm and the learning process. Both the actor and critic networks are composed of two hidden layers. However, the actor network is designed with a larger number of neurons, as it outputs the mean and standard deviation for three continuous action dimensions, whereas the critic only produces a single scalar representing the state-value function.

The learning rate η controls the step size during gradient descent and must balance convergence speed with training stability to prevent overly aggressive updates. The `n_epochs` parameter specifies the number of times each mini-batch is iterated over during training, effectively controlling how often the same data is reused in a single update cycle.

The coefficients c_1 and c_2, as defined in equation 4.16, regulate the contributions to the total loss of the value function loss and the entropy term, respectively. While c_1 was set to its default value of 0.5, c_2 was deliberately assigned a nonzero value, departing from its default of zero, in order to promote exploration during training.

The parameters γ, λ, and ϵ correspond to the discount factor, the GAE parameter, and the clip ratio, respectively. While standard values are used in this work, the clip ratio ϵ is set slightly higher than the common default (0.2) to allow for larger policy updates.

Another important parameter is `std_init`, which specifies the initial standard deviation of the Gaussian distributions from which the actor samples actions. This value influences the initial degree of exploration.

Finally, `weight_decay`, `clip_range_vf`, and `max_grad_norm` are stabilization techniques. Weight decay discourages large weights by adding an L_2 penalty to the loss function. Value function clipping introduces a clipping mechanism analogous to that used for the policy update, helping prevent excessive shifts in value estimates. Gradient clipping limits the maximum norm of gradients during backpropagation, further improving training stability.

5.2.4 Reward function constants

Constant	**Value**
α	5×10^{-1}
β	10^{-2}
δ	10^{-2}

Table 5.6: Constants used in the reward function

The selection of appropriate weights for the components of the reward function is arguably one of the most critical and impactful design decisions in the Reinforcement Learning setup. As shown in Table 5.6, the constants α, β, and δ determine the relative importance of the three main objectives encoded in the reward function.

The constant α scales the penalty associated with collision risk and reflects the primary objective of the task: avoiding collisions with debris. The constant β weighs the fuel consumption term, encouraging fuel-efficient maneuvers. Lastly, δ governs the penalty for deviations from the reference trajectory, promoting a return to the nominal orbital path when possible.

These constants must be chosen carefully to keep a good balance between the different goals. The values shown were selected empirically to make sure all three objectives are taken into account during training.

5.2.5 LSTM- and Transformer-specific parameters

Constant	**Value**
`lstm_embed_dim`	6
`tf_dim`	16
`tf_heads`	4
`tf_layers`	2
`tf_ffn`	30
`tf_dropout`	0.1

Table 5.7: LSTM- and Transformer-specific parameters

Table 5.7 presents the hyperparameters used for the LSTM and Transformer components of the architecture.

The parameter `lstm_embed_dim` defines the size of the LSTM's hidden state and was empirically set to 6. Although larger values could also be effective, this choice was found to be sufficient for generating a compact yet informative summary vector representing the 10 debris objects.

This summary vector, after concatenation with the S/C state vector, is projected to a higher-dimensional space of size 16 (`tf_dim`) to increase its expressiveness before being processed by the Transformer. The Transformer encoder applies multi-head attention using 4 parallel attention heads (`tf_heads`), followed by a feedforward neural network with 30 neurons (`tf_ffn`). A dropout rate of 0.1 (`tf_dropout`) is applied after both the attention and feedforward layers to promote training stability. This entire process is repeated for 2 layers (`tf_layers`), allowing the model to refine the representation over multiple Transformer blocks.

The relatively small size of the Transformer is a deliberate design choice that reflects the nature of the problem: the orbital scenarios considered involve structured and moderately dynamic interactions, which do not require deep or high-capacity models.

5.3 Experimental results

From this point onward, the results presented aim to systematically compare the performance of two main model architectures: (1) a baseline architecture combining an LSTM feature extractor with PPO, and (2) an enhanced architecture that integrates a Transformer encoder between the LSTM and PPO modules. These two variants are evaluated under two distinct settings: with and without artificially introduced noise in the state space.

In this context, the term noise refers to bounded multiplicative perturbations applied to the position and velocity components of the debris objects during database generation. Specifically, when noise is enabled, each component of the state vector is scaled by a random factor uniformly drawn from the range $[0.95,\ 1.05]$, introducing independent random variations of up to $\pm 5\%$ in each component of every debris object.

This mechanism simulates realistic inaccuracies such as sensor noise in orbital estimation, effectively replacing real states with observations to simulate a POMDP scenario. The goal is to evaluate whether the Transformer-enhanced policy can learn the underlying dynamics under noisy conditions and exhibit greater robustness and stability compared to the PPO baseline.

The two configurations (LSTM+PPO and LSTM+Transformer+PPO) will be analyzed in two representative scenarios: one with ideal Keplerian motion and no noise, and one with both orbital perturbations and noise.

5.3.1 Comparison under Keplerian dynamics without noise

As a first experiment, we compare the performance of the baseline PPO model with that of the Transformer-enhanced PPO under a simplified scenario. In this setting, the orbital dynamics are purely Keplerian and no artificial noise is applied.

Figures 5.3 and 5.4 illustrate the evolution of the average cumulative reward during training and evaluation, respectively. The average cumulative reward, also referred to as the undiscounted return, is defined as the mean, across a set of episodes, of the total reward accumulated in each episode, without applying a discount factor. This distinguishes it from the discounted return defined in equation 4.1.

During training, the average is computed over the five episodes that constitute a single rollout buffer. In contrast, during evaluation, a single evaluation episode is executed every five training episodes, and the reported average corresponds to the return of that sole episode, as detailed in Table 5.4.

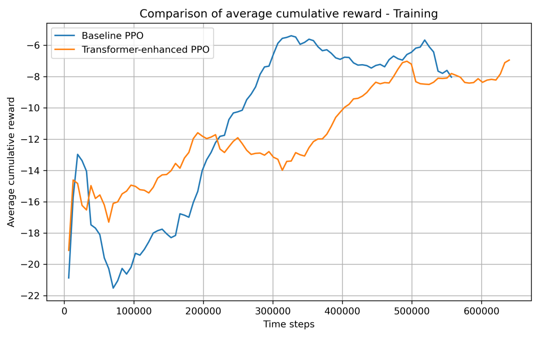

Figure 5.3: Average cumulative reward in training

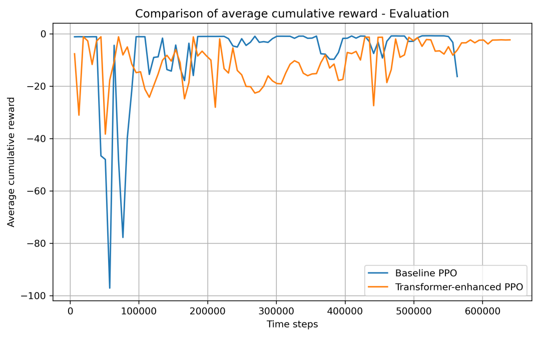

Figure 5.4: Average cumulative reward in evaluation

As it can be noticed from the figures, the behavior of the two models differs across training and evaluation.

In the training phase, both the baseline PPO and the Transformer-enhanced PPO progressively improve their performance over time. The baseline PPO shows a steeper learning curve and reaches higher average cumulative rewards earlier, but its performance plateaus and slightly declines toward the end of training. The Transformer-enhanced PPO, on the other hand, learns more slowly, likely due to the added complexity of the Transformer module, but demonstrates a steadier progression and eventually approaches the performance of the baseline model.

In the evaluation phase, both models exhibit high variance in early episodes, particularly the baseline PPO, which shows sharp drops in performance. These sudden declines correspond to specific evaluation episodes where the agent experienced increased collision risk, but do not concern the enhanced policy, due to the Transformer's ability to contextualize temporal sequences (i.e., the dynamics is taken into account). Over time, the variability decreases and both models stabilize, consistently achieving high performance.

It is also worth noting that the Transformer-enhanced PPO was trained for the full 500 episodes in an effort to explore potential long-term gains, even though its best performance was achieved early. In contrast, the baseline PPO model was terminated earlier due to the absence of further improvements.

Figures 5.5 and 5.6 show the evolution of total loss and value function loss during training for both models.

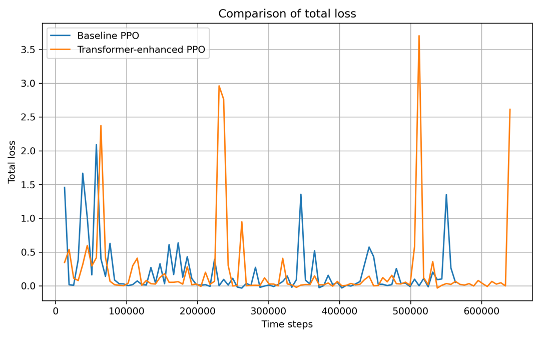

Figure 5.5: Total loss

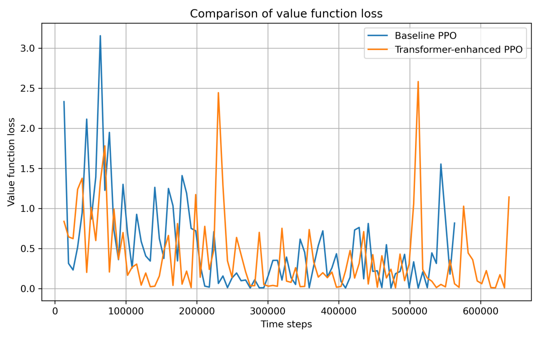

Figure 5.6: Value function loss

In terms of total loss, both algorithms show a general decreasing trend, but the Transformer-enhanced PPO exhibits higher peaks throughout training, indicating greater variability in overall optimization.

For the value function loss, both models display similar behavior: a rapid decline in early training followed by stabilization with comparable fluctuations. No significant difference in overall stability is observed between the two.

These results suggest that while both models converge, the Transformer-enhanced PPO introduces slightly more variability in total loss, potentially due to its increased representational complexity.

From this point onward, we focus on the evaluation episodes that yielded the best performance for each model. For the baseline PPO, the highest evaluation cumulative reward was achieved at episode 85, whereas the Transformer-enhanced PPO reached its peak performance much earlier, at episode 28.

Figures 5.7 and 5.8 present the individual reward components for these respective episodes.

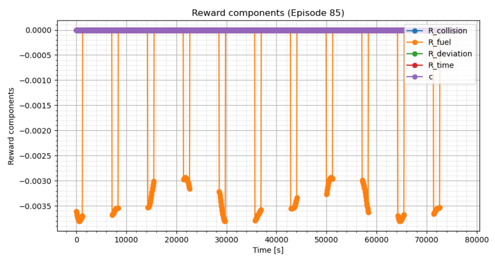

Figure 5.7: Reward terms in the baseline PPO

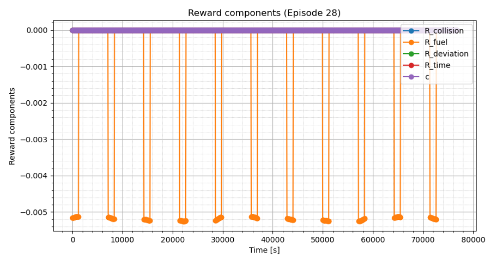

Figure 5.8: Reward terms in the Transformer-enhanced PPO

Both models exhibit desirable outcomes, with the only penalty being related to fuel usage. In that regard, the baseline achieves a slightly better result overall, being characterized by a lower thrust magnitude and therefore less fuel consumption penalty. The on-off behavior of the *engine* control variable is a consequence of the thermal constraint introduced in Section 2.3.3.

The same can be observed when analyzing the actions. Figures 5.9 and 5.10 show the evolution of the control variables ϕ (elevation angle) and θ (azimuth angle) during the best evaluation episodes for the baseline and Transformer-enhanced PPO models.

In both cases, the thrust is applied intermittently, with coasting phases in which the engine is forced to shut down. The horizontal dotted lines indicate the action bounds: $\phi \in [-\pi/2, \pi/2]$ and $\theta \in [0, 2\pi]$.

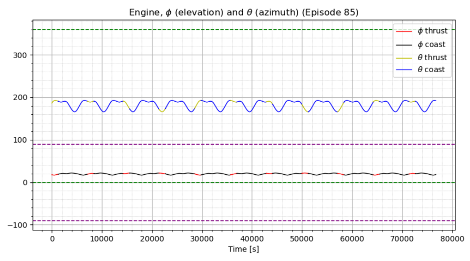

Figure 5.9: S/C actions in the baseline PPO

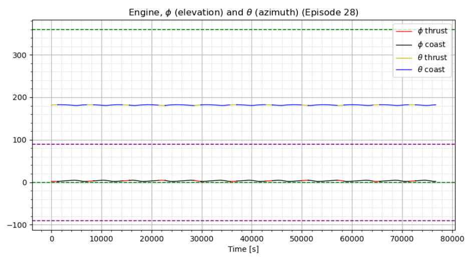

Figure 5.10: S/C actions in the Transformer-enhanced PPO

Although both models achieve comparable outcomes, their control strategies differ noticeably. The baseline PPO tends to achieve greater separation from the debris by applying thrust with a significant out-of-plane component (i.e., with non-zero elevation angles). In contrast, the Transformer-enhanced PPO relies more heavily on in-plane thrusting, keeping the elevation angle closer to zero. In both cases, the azimuth angle θ remains close to π, indicating that the thrust is primarily directed toward the center of the Earth, opposite to the $\hat{\mathbf{h}}_1$ unit vector in the RTN frame.

Although the baseline algorithm commands out-of-plane thruster firings to increase separation from the debris, the satellite's inclination remains within acceptable limits. This is because thrust is applied symmetrically on opposite sides of the orbit, effectively compensating for any inclination change. As a result, the inclination stays near its initial value, and no penalty for trajectory deviation is incurred.

The increased distance from the orbital debris, achieved through these distinct strategies, is illustrated in Figures 5.11 and 5.12.

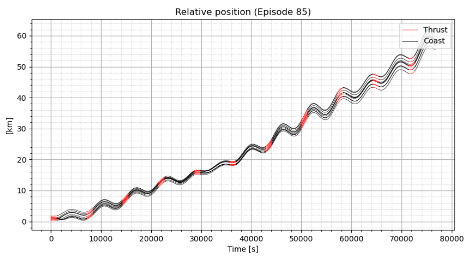

Figure 5.11: Relative position of the S/C in baseline PPO

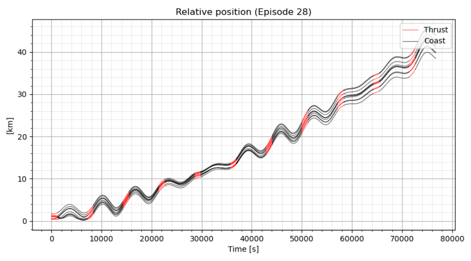

Figure 5.12: Relative position of the S/C in in the Transformer-enhanced PPO

It is evident that the Transformer maintains a smaller final separation from the debris, indicating its ability to avoid collisions while remaining closer to its original nominal orbit. Consequently, this would result in a more fuel-efficient restore phase, should one be required.

As the distance between the spacecraft and the debris grows, the collision probability rapidly drops to zero in both cases, as shown in Figures 5.13 and 5.14.

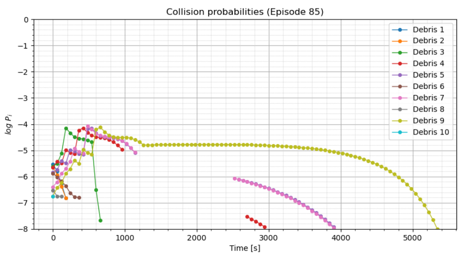

Figure 5.13: Collision probabilities in the baseline PPO

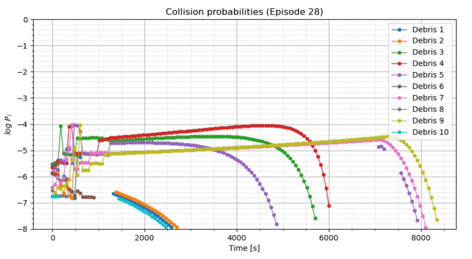

Figure 5.14: Collision probabilities in the Transformer-enhanced PPO

The collision probabilities for all debris objects remain consistently below the critical threshold of 10^{-4} throughout the episode. This ensures that no collision penalties are incurred in either case.

The early suppression of collision risk reflects the effectiveness of both control policies in rapidly maneuvering the spacecraft to safe trajectories. Notably, the baseline PPO appears to neutralize high-risk objects slightly earlier, while the Transformer-enhanced PPO achieves similar safety margins with a more gradual reduction. In both cases, the collision avoidance objective is fully satisfied.

5.3.2 Comparison under perturbed dynamics with noise

As a second experiment, we compare the performance of the two models under a more challenging scenario. In this setting, the database with added noise and orbital perturbations has been used.

Figures 5.15 and 5.16 show the behavior of the average cumulative reward during training and evaluation, respectively.

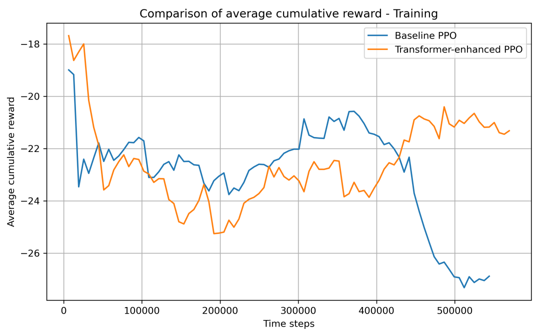

Figure 5.15: Average cumulative reward in training

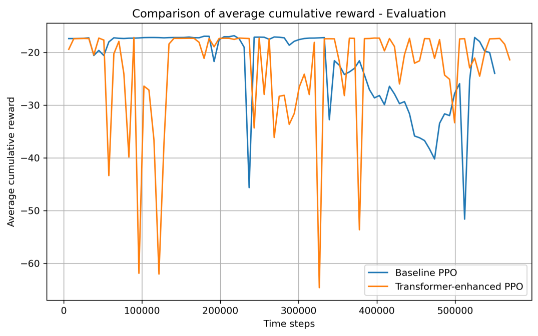

Figure 5.16: Average cumulative reward in evaluation

Although the Transformer-enhanced architecture exhibits slower initial learning, it eventually surpasses the PPO baseline in the later stages of training, with performance continuing to improve. In contrast, the PPO baseline undergoes a sudden drop in undiscounted return, likely due to increased exploration after failing to escape a suboptimal policy.

The slower learning of the enhanced architecture can be attributed to its higher computational complexity. However, this same complexity contributes to greater training stability, making the policy less prone to diverging or regressing once a suboptimal solution has been reached.

During inference, both models clearly fail to surpass a certain performance threshold. This limitation is most likely due to the agent's inability to simultaneously achieve collision avoidance and maintain effective station-keeping, a trade-off that will be further examined in the following analysis.

However, the enhanced policy exhibits more consistent high performance in the final stages, aligning with the trends observed during training.

Both models were terminated based on early stopping criteria, triggered by the lack of further improvements in the evaluation cumulative reward.

Figures 5.17 and 5.18 show the evolution of total loss and value function loss during training for both models.

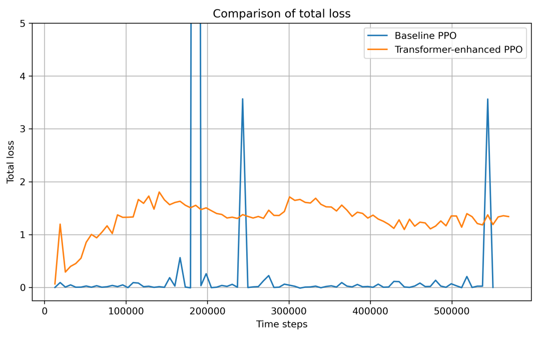

Figure 5.17: Total loss

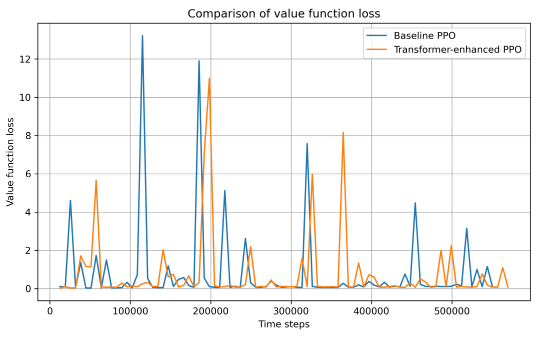

Figure 5.18: Value function loss

Although the baseline PPO exhibits a lower average total loss, it is marked by high and irregular peaks. In contrast, the enhanced PPO shows a smoother and more stable loss curve, suggesting greater training stability and potential for further improvement with extended training.

The value function loss decreases over time for both policies, as expected, but baseline PPO displays more pronounced spikes during the later stages of training.

Like in the previous experiment, we are now interested in analyzing the outcomes of the best evaluation episodes for both models. For the baseline PPO, the highest evaluation cumulative reward was achieved at episode 33, while the Transformer-enhanced PPO reached its best performance at episode 52.

Figures 5.19 and 5.20 present the individual reward components for these respective episodes.

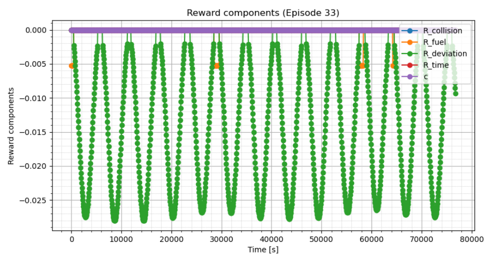

Figure 5.19: Reward terms in the baseline PPO

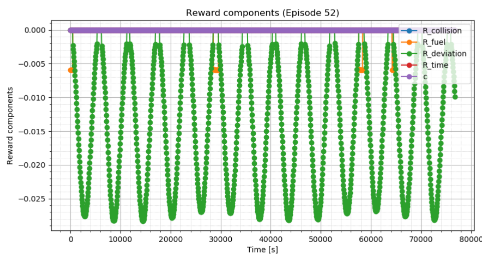

Figure 5.20: Reward terms in the Transformer-enhanced PPO

In both scenarios, the primary objective of collision avoidance is successfully achieved. However, significant penalties are incurred throughout the episode due to deviations from the nominal orbit. These penalties arise from the eccentricity, which varies in a sinusoidal manner with a period corresponding to one orbital revolution of the spacecraft. This oscillatory behavior is induced by the J_2 perturbation and causes the eccentricity to exceed the bounds defined in the reward function.

Additionally, it can be observed that the fuel efficiency term becomes active only when it is the sole penalty being applied. In all other cases, it is deactivated to prioritize collision avoidance and the restoration of the reference orbit.

Figures 5.21 and 5.22 illustrate the actions executed by each algorithm during their respective best-performing test episodes.

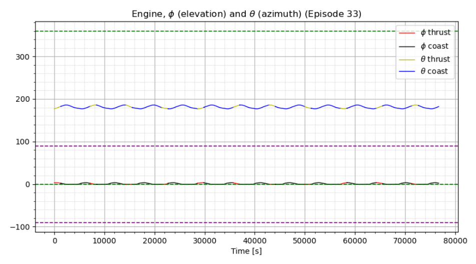

Figure 5.21: S/C actions in the baseline PPO

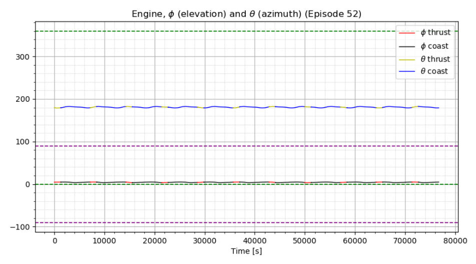

Figure 5.22: S/C actions in the Transformer-enhanced PPO

It is evident that both models adopt nearly identical control policies: thrust firings are primarily directed in-plane, opposite to the $\hat{\mathbf{h}}_1$ unit vector in the RTN frame, effectively pointing towards the center of the Earth. As observed in earlier cases, the thrust sequences are regularly interrupted by cooldown phases, each lasting one orbital period.

Finally, the successful achievement of the primary objective, collision avoidance, is clearly demonstrated in 5.23 and 5.24.

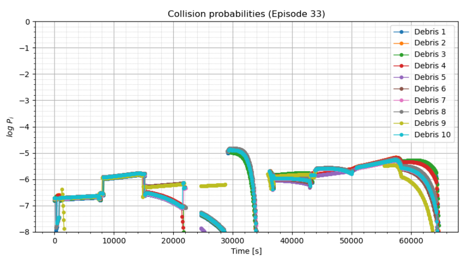

Figure 5.23: Collision probabilities in the baseline PPO

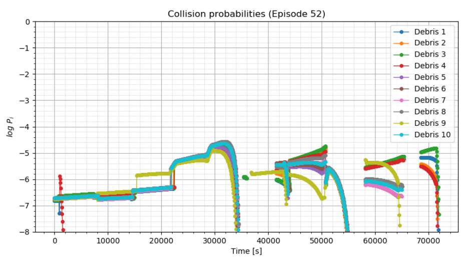

Figure 5.24: Collision probabilities in the Transformer-enhanced PPO

In both cases, the collision probability remains consistently below the 10^{-4} threshold, despite the presence of noisy observations and perturbed dynamics. Unlike the previous experiment, where the spacecraft achieved significant separation from the debris early in the episode, the separation here is more limited. As a result, non-negligible collision probabilities persist until nearly the end of the episode.

Chapter 6

Conclusion and outlook

6.1 Thesis outcomes

This thesis presented a comprehensive investigation into the use of Proximal Policy Optimization algorithms enhanced with Transformer and LSTM-based architectures for autonomous decision-making in spacecraft collision avoidance tasks, under Partially Observable Markov Decision Process conditions and in the presence of multiple debris objects.

The primary objective was to assess whether advanced sequence-processing models such as LSTMs and Transformers could address the challenge posed by partial and noisy observability of the environment. The work extended the PPO framework with an LSTM encoder and a custom Transformer module designed to extract meaningful representations from high-dimensional, sequential input data.

The experimental campaign, detailed in Chapter 5, demonstrated that:

1. **PPO proved robust to environment noise and orbital perturbations.**

   Baseline PPO, coupled with LSTM, worked well in all setups. Even when faced with noisy observations to learn from, and in the more challenging scenario of the full dynamical model, the algorithm fulfilled the main goal of avoiding collisions.

2. **LSTM encoders facilitated scalable representation of multiple debris objects.**

   Leveraging an LSTM, the agent was able to extract a meaningful fixed-size embedding vector from the unordered sequence of debris states. Combined with reward feedback, this enabled the policy to make effective decisions and avoid all debris. Moreover, the recurrent nature of the LSTM architecture provides the flexibility to handle a variable number of debris objects.

3. **The Transformer worked as dynamical context.**

   In contrast to the Transformer-based approaches for RL discussed in Section 3.3.4, this thesis proposes a novel architecture in which a lightweight custom Transformer leverages the history of past observations to capture system dynamics and enhance the agent's belief state.

4. **The Transformer helped regularize training behavior.**

   Both in the absence of noise (MDP framework), and when faced with noisy observations (POMDP framework), the Transformer produced smoother returns and more stable convergence. This can be attributed to the Transformer's ability to contextualize the dynamics hidden behind the observations. However, this enhanced performance comes at the cost of increased computational complexity, resulting in longer settling times.

In conclusion, this thesis makes a strong case for incorporating structured deep learning models, particularly Transformers and LSTMs, within Reinforcement Learning pipelines to tackle POMDPs under challenging conditions. By addressing the learning of robust policies in noisy environments, the proposed architecture paves the way for more intelligent, adaptable, and autonomous decision systems.

6.2 Potential applications

The methods developed and validated in this work have broad implications, also beyond the specific task of spacecraft collision avoidance. Several areas stand to benefit from the proposed approach:

- **Autonomous spacecraft collision avoidance systems:** Since current collision avoidance procedures still largely depend on human intervention, the research presented in this thesis could contribute to the development of future autonomous on-board systems.

- **Active debris removal and in-orbit servicing:** With the growing density of space objects, establishing a circular space economy is becoming increasingly urgent. The need to deorbit debris and extend the operational life of existing satellites will soon be critical. Transformer-enhanced Reinforcement Learning agents could support autonomous on-board decision-making in these scenarios, particularly under conditions of noisy or uncertain measurements.

- **Multi-agent robotics:** Tasks involving a variable number of agents or obstacles (e.g., swarm robotics, underwater drones, autonomous vehicles in urban environments) require policies that generalize across configurations.

The LSTM-based embedding of dynamic entities offers a path forward for such systems.

- **UAV control:** Unmanned Aerial Vehicles (UAVs) operating in crowded airspace can use similar architectures to avoid mid-air collisions or restricted zones. Particularly under low-visibility conditions, the benefits of Transformer-based observation encoding become compelling.
- **Healthcare and wearables:** For real-time decision-making in health monitoring systems, where input data streams are noisy and partially observed (e.g., ECG, PPG signals), attention-based policies may improve estimation of the signal.
- **Noise filtering:** More broadly, any application involving the estimation of a variable observed, either directly or indirectly, under noisy conditions could benefit from this approach. Traditional methods such as Kalman filtering and batch least squares, widely used in orbit determination, may be enhanced by preprocessing temporal observation sequences with a Transformer encoder.

These applications share a need for decision systems that can handle high-dimensional, time-dependent inputs and that are resilient to partial observability and noise, requirements directly addressed by this thesis.

6.3 Future work

Several promising research directions emerge from the findings of this thesis.

1. **Extension to full multi-agent settings**

   While this work focused on a single-agent system navigating through multiple uncooperative objects, a natural next step is the explicit modeling of interactive multi-agent environments, where active spacecraft, rather than passive debris, may also have decision-making capacity. Incorporating decentralized policies with inter-agent communication and coordination could enhance safety and performance.

2. **Integration with realistic orbit determination and sensor models**

   In future work, the observation model can be expanded to simulate the collision avoidance process more realistically. Instead of direct observations of the states of the spacecraft and debris, an orbit determination model could be included, and sensor constraints (e.g., time delay and field-of-view limitations) might be added. Coupling PPO-Transformer architectures with domain-specific sensor models would bring simulations closer to on-board deployment conditions.

3. **Curriculum learning and domain randomization**

   To improve generalization further, training regimes could include curriculum learning, where the agent is first trained on simpler tasks and gradually exposed to harder ones, or domain randomization, which exposes the agent to a wide variety of conditions to promote robust policy acquisition.

4. **Exploitation of LSTM properties**

   LSTMs not only enable the aggregation of information from multiple debris objects into a fixed-size representation, but also provide the flexibility to accommodate the addition or removal of debris, such as when previously critical debris become irrelevant or new objects enter the vicinity. This adaptability arises from the internal gating mechanisms, specifically the input and forget gates, which allow selective integration and discarding of information over time. Future experiments will further investigate these properties.

Bibliography

[1] Satellite Industry Association. *2024 State of the Satellite Industry Report*. Tech. rep. 2024-SSIR. June 2024 (cit. on p. 1).

[2] ESA Space Debris Office. *ESA's Annual Space Environment Report*. Tech. rep. GEN-DB-LOG-00288-OPS-SD. July 2024 (cit. on pp. 1, 6, 11).

[3] Inter-Agency Space Debris Coordination Committee. *IADC Space Debris Mitigation Guidelines*. Tech. rep. IADC-02-01 Rev. 4. Jan. 2025 (cit. on pp. 2, 9).

[4] NASA Orbital Debris Program Office. *Photo Gallery*. URL: `https://orbitaldebris.jsc.nasa.gov/photo-gallery/` (cit. on p. 3).

[5] ESA Space Debris Office. *Space debris by the numbers*. URL: `https://www.esa.int/Space_Safety/Space_Debris/Space_debris_by_the_numbers` (cit. on p. 4).

[6] V. Braun et al. «Analysis of Breakup Events». In: *7th European Conference on Space Debris*. Apr. 2017, pp. 1–4 (cit. on p. 4).

[7] T.S. Kelso. «Analysis of the 2007 Chinese ASAT Test and the Impact of its Debris on the Space Environment». In: *8th Advanced Maui Optical and Space Surveillance Technologies Conference*. Sept. 2007 (cit. on p. 5).

[8] D. Wright. «Colliding satellites: consequences and implications». In: *Union of Concerned Scientists* 26 (Feb. 2009) (cit. on p. 6).

[9] T.S. Kelso. «Analysis of the Iridium 33-Cosmos 2251 collision». In: *Advances in the Astronautical Sciences* 135 (Sept. 2009), pp. 1099–1112 (cit. on p. 6).

[10] M.J. Matney et al. «Characterization of the 2012-044C Briz-M upper stage breakup». In: *6th European Conference on Space Debris*. Apr. 2013 (cit. on p. 6).

[11] J.-C. Liou. *Orbital Debris Modeling and the Future Orbital Debris Environment*. Orbital Debris Lecture. Sept. 2012 (cit. on pp. 7, 8).

[12] United Nations Office for Outer Space Affairs. *Guidelines for the Long-term Sustainability of Outer Space Activities of the Committee on the Peaceful Uses of Outer Space*. Tech. rep. ST/SPACE/79. June 2021 (cit. on p. 9).

[13] *ISO 24113:2023*. International Organization for Standardization, 2023 (cit. on p. 10).

[14] *ECSS-U-AS-10C Rev.2*. European Cooperation for Space Standardization, 2024 (cit. on p. 10).

[15] ESA Space Debris Mitigation Working Group. *ESA Space Debris Mitigation Requirements*. Tech. rep. ESSB-ST-U-007 Issue 1. Oct. 2023 (cit. on p. 10).

[16] G.A. Graham et al. «The chemistry of micrometeoroid and space debris remnants captured on Hubble Space Telescope solar cells». In: *International Journal of Impact Engineering* 26 (Dec. 2001), pp. 263–274 (cit. on p. 10).

[17] A.T. Kearsley et al. «Cosmic dust impacts on the Hubble Space Telescope». In: *Philosophical Transactions of the Royal Society A* 382 (May 2024) (cit. on p. 10).

[18] S. Ryan and E.L. Christiansen. «Hypervelocity impact testing of advanced materials and structures for micrometeoroid and orbital debris shielding». In: *Acta Astronautica* 83 (Mar. 2013), pp. 216–231 (cit. on p. 11).

[19] W.P. Schonberg and J.E. Williamsen. «Modeling Damage in Spacecraft Impacted by Orbital Debris Particles». In: *The Journal of the Astronautical Sciences* 47 (Jan. 1999), pp. 103–115 (cit. on p. 11).

[20] D.J. Kessler and B.G. Cour-Palais. «Collision frequency of artificial satellites: The creation of a debris belt». In: *Journal of Geophysical Research: Space Physics* 83 (June 1978), pp. 2637–2646 (cit. on p. 11).

[21] J.-C. Liou and N. L. Johnson. «Risks in Space from Orbiting Debris». In: *Science* 311 (Jan. 2006), pp. 340–341 (cit. on p. 12).

[22] D.A. Vallado. *Fundamentals of astrodynamics and applications*. Springer Science & Business Media, 2001 (cit. on pp. 14, 25).

[23] B. Schutz et al. *Statistical orbit determination*. Elsevier, 2004 (cit. on p. 14).

[24] *Spaceflight Safety Handbook for Satellite Operators*. 18th and 19th Space Defense Squadron. 2023. URL: `https://www.space-track.org/documents/SFS_Handbook_For_Operators_V1.7.pdf` (cit. on p. 14).

[25] ESA Advanced Concepts Team. *Collision Avoidance Challenge*. URL: `https://kelvins.esa.int/collision-avoidance-challenge/` (cit. on p. 15).

[26] T. Uriot et al. «Spacecraft collision avoidance challenge: Design and results of a machine learning competition». In: *Astrodynamics* 6 (June 2022), pp. 121–140 (cit. on p. 15).

[27] K. Merz et al. «Current Collision Avoidance service by ESA's Space Debris Office». In: *7th European Conference on Space Debris*. Apr. 2017 (cit. on p. 16).

[28] R. Hiles et al. «Report on 2020 Mega-Constellation Deployments and Impacts to Space Domain Awareness». In: *Advanced Maui Optical and Space Surveillance Technologies Conference*. Sept. 2021 (cit. on p. 16).

[29] Jonathan McDowell. *Starlink Statistics*. Apr. 2025. URL: `https://planet4589.org/space/con/star/stats.html` (cit. on p. 16).

[30] ESA. *Anatomy of a debris incident*. URL: `https://www.esa.int/Enabling_Support/Operations/Anatomy_of_a_debris_incident` (cit. on p. 16).

[31] Space.com. *SpaceX Starlink satellites made 50,000 collision-avoidance maneuvers in the past 6 months. What does that mean for space safety?* URL: `https://www.space.com/spacex-starlink-50000-collision-avoidance-maneuvers-space-safety` (cit. on p. 16).

[32] R.P. Patera and G.E. Peterson. «Space Vehicle Maneuver Method to Lower Collision Risk to an Acceptable Level». In: *Journal of guidance, control and dynamics* 26 (Mar. 2003), pp. 233–237 (cit. on p. 18).

[33] I. Grande-Olalla et al. «Collision Risk Assessment and Avoidance Manoeuvres: New Tools CORAM for ESA». In: *6th European Conference on Space Debris*. Apr. 2013 (cit. on p. 18).

[34] J.H. Ayuso. «Collision Avoidance Maneuver Optimization - A Fast and Accurate Semi-Analytical Approach». MA thesis. Universidad Politécnica de Madrid, 2014 (cit. on p. 18).

[35] J.H. Ayuso et al. «OCCAM: Optimal computation of collision avoidance maneuvers». In: *6th International Conference on Astrodynamics Tools and Techniques*. Mar. 2016 (cit. on p. 18).

[36] A. Morselli et al. «Collision avoidance maneuver design based on multi-objective optimization». In: *Advances in the Astronautical Sciences* 152 (Feb. 2014) (cit. on p. 18).

[37] S. Lee et al. «Collision Avoidance Maneuver Planning Using GA for LEO and GEO Satellite Maintained in Keeping Area». In: *International Journal of Aeronautical and Space Sciences* 13 (Dec. 2012), pp. 474–483 (cit. on p. 18).

[38] E. Kim et at. «A Study on the Collision Avoidance Maneuver Optimization with Multiple Space Debris». In: *Journal of Astronomy and Space Sciences* 29 (Mar. 2012), pp. 11–21 (cit. on p. 18).

[39] G.M. Neves and A.F.B.A. Prado. «Collision Avoidance Maneuvers Optimization Using Genetic Algorithm». In: *14th Workshop em Engenharia e Tecnologia Espaciais (WETE)*. Dec. 2023 (cit. on p. 18).

[40] S. Dutta and A.K. Misra. «Convex optimization of collision avoidance maneuvers in the presence of uncertainty». In: *Acta Astronautica* 197 (Aug. 2022), pp. 257–268 (cit. on p. 18).

[41] B. Li et al. «Spacecraft close-range trajectory planning via convex optimization and multi-resolution technique». In: *Acta Astronautica* 175 (Oct. 2020), pp. 421–437 (cit. on p. 18).

[42] Q. Hu et al. «Trajectory optimization for accompanying satellite obstacle avoidance». In: *Aerospace Science and Technology* 82-83 (Nov. 2018), pp. 220–233 (cit. on p. 19).

[43] J.S. Catulo et al. «Predicting the Probability of Collision of a Satellite with Space Debris: A Bayesian Machine Learning Approach». In: *74th International Astronautical Congress, IAC 2023*. Oct. 2023 (cit. on p. 19).

[44] G. Acciarini et al. «Kessler: a Machine Learning Library for Spacecraft Collision Avoidance». In: *8th European Conference on Space Debris*. Apr. 2021 (cit. on p. 19).

[45] J. Tao et al. «SDebrisNet: A Spatial–Temporal Saliency Network for Space Debris Detection». In: *Appl. Sci. 2023* 13 (Apr. 2023) (cit. on p. 19).

[46] R. Abay. «Predicting Collision Risk from Historical Collision Data Messages (CDMs) using Machine Learning». Unpublished (cit. on p. 19).

[47] I. Priyadarshini. «Enhanced Space Debris Detection and Monitoring Using a Hybrid Bi-LSTM-CNN and Bayesian Optimization». In: *Artificial Intelligence and Applications* 3 (Dec. 2024) (cit. on p. 19).

[48] S. Liu et al. «Autonomous Spacecraft Collision Avoidance with Multiple Space Debris Based on Reinforcement Learning». In: *42nd Chinese Control Conference*. July 2023 (cit. on pp. 20, 64).

[49] Q. Qu et al. «Spacecraft Proximity Maneuvering and Rendezvous with Collision Avoidance Based on Reinforcement Learning». In: *IEEE Transactions on Aerospace and Electronic Systems* 58 (Dec. 2022), pp. 5823–5834 (cit. on p. 20).

[50] J. Blaise and M. Bazzocchi. «Space Manipulator Collision Avoidance Using a Deep Reinforcement Learning Control». In: *Aerospace* 10 (Aug. 2023) (cit. on pp. 21, 64).

[51] L. Gremyachikh et al. *Space Navigator: A Tool for the Optimization of Collision Avoidance Maneuvers*. 2019. arXiv: `1902.02095`. URL: `https://arxiv.org/abs/1902.02095` (cit. on p. 21).

[52] C. Mu et al. «Autonomous Spacecraft Collision Avoidance with a Variable Number of Space Debris Based on Safe Reinforcement Learning». In: *Aerospace Science and Technology* 149 (June 2024) (cit. on pp. 21, 53, 64, 74).

[53] N. Bourriez et al. *Spacecraft Autonomous Decision-Planning for Collision Avoidance: a Reinforcement Learning Approach.* 2023. arXiv: `2310.18966`. URL: `https://arxiv.org/abs/2310.18966` (cit. on pp. 21, 38, 53).

[54] R.R. Bate et al. *Fundamentals of astrodynamics.* Courier Dover Publications, 2020 (cit. on p. 23).

[55] S. Metz. «Implementation and comparison of data-based methods for collision avoidance in satellite operations». MA thesis. TU Darmstadt, 2020 (cit. on p. 28).

[56] H.D. Curtis. *Orbital mechanics for engineering students.* Butterworth Heinemann, 2019 (cit. on p. 30).

[57] R.T. Gavin. «NASA's Orbital Debris Conjuction Assessment and Collision Avoidance Strategy». In: *2010 AAS Guidance and Control Conference.* Feb. 2010, p. 16 (cit. on p. 39).

[58] L. Chen et al. *Orbital Data Applications for Space Objects.* Springer, 2017 (cit. on p. 40).

[59] S. Alfano. «Review of conjunction probability methods for short-term encounters». In: *Advances in the Astronautical Sciences* (Jan. 2007), pp. 2–3 (cit. on p. 40).

[60] J.L. Foster (Jr.) and H.S. Estes. *A Parametric Analysis of Orbital Debris Collision Probability and Maneuver Rate for Space Vehicles.* Tech. rep. JSC-25898. NASA, National Aeronautics and Space Administration, Lyndon B. Johnson Space Center, Aug. 1992 (cit. on p. 41).

[61] R.P. Patera. «General Method for Calculating Satellite Collision Probability». In: *Journal of Guidance, Control, and Dynamics* 24 (July 2001) (cit. on p. 41).

[62] R.P. Patera. «Conventional Form of the Collision Probability Integral for Arbitrary Space Vehicle Shape». In: *AIAA/AAS Astrodynamics Specialist Conference and Exhibit.* Aug. 2004 (cit. on p. 41).

[63] S. Alfano. «A Numerical Implementation of Spherical Object Collision Probability». In: *Journal of the Astronautical Sciences* 53 (Mar. 2005) (cit. on p. 41).

[64] K. Chan. «Comparison of Methods for Spacecraft Collision Probability Computations». In: *AIAA/AAS Space Flight Mechanics Meeting.* Apr. 2020, pp. 3–6 (cit. on p. 41).

[65] C.M. Bishop. *Neural networks for pattern recognition.* Oxford university press, 1995 (cit. on pp. 45, 48).

[66] I. Goodfellow et al. *Deep learning.* MIT press, 2016 (cit. on pp. 45, 48).

[67] D.E. Rumelhart et al. «Learning representations by back-propagating errors». In: *nature* 323 (Oct. 1986), pp. 533–536 (cit. on p. 48).

[68] D.P. Kingma and J. Ba. «Adam: A method for stochastic optimization». In: *arXiv preprint arXiv:1412.6980* (Dec. 2014) (cit. on p. 49).

[69] Y. Bengio et al. «Learning long-term dependencies with gradient descent is difficult». In: *IEEE transactions on neural networks* 5 (Mar. 1994), pp. 157–166 (cit. on p. 51).

[70] R. Pascanu et al. «On the difficulty of training recurrent neural networks». In: *International conference on machine learning*. May 2013, pp. 1310–1318 (cit. on p. 51).

[71] K. Cho et al. «Learning phrase representations using RNN encoder-decoder for statistical machine translation». In: *arXiv preprint arXiv:1406.1078* (June 2014) (cit. on p. 51).

[72] S. Hochreiter and J. Schmidhuber. «Long short-term memory». In: *Neural computation* 9 (Nov. 1997), pp. 1735–1780 (cit. on p. 51).

[73] A. Graves et al. «Speech recognition with deep recurrent neural networks». In: *IEEE international conference on acoustics, speech and signal processing*. May 2013, pp. 6645–6649 (cit. on p. 53).

[74] I. Sutskever et al. «Sequence to sequence learning with neural networks». In: *Advances in neural information processing systems* 27 (Sept. 2014) (cit. on p. 53).

[75] A. Vaswani et al. «Attention is all you need». In: *Advances in neural information processing systems* 30 (2017) (cit. on pp. 53–55).

[76] P. Shaw et al. «Self-attention with relative position representations». In: *arXiv preprint arXiv:1803.02155* (Mar. 2018) (cit. on p. 54).

[77] J. Devlin et al. «Bert: Pre-training of deep bidirectional transformers for language understanding». In: *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. June 2019, pp. 4171–4186 (cit. on p. 55).

[78] J.L. Ba et al. «Layer normalization». In: *arXiv preprint arXiv:1607.06450* (July 2016) (cit. on p. 56).

[79] L. Chen et al. «Decision transformer: Reinforcement learning via sequence modeling». In: *Advances in neural information processing systems* 34 (Dec. 2021), pp. 15084–15097 (cit. on p. 57).

[80] T. Guffanti et al. «Transformers for trajectory optimization with application to spacecraft rendezvous». In: *2024 IEEE Aerospace Conference*. Mar. 2024, pp. 1–13 (cit. on p. 58).

[81] E.L. Thorndike. «Animal intelligence: An experimental study of the associative processes in animals». In: *The Psychological Review: Monograph Supplements* 2 (June 1898) (cit. on p. 59).

[82] R.S. Sutton and A.G. Barto. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998 (cit. on p. 60).

[83] V. Mnih et al. «Human-level control through deep reinforcement learning». In: *nature* 518 (Feb. 2015), pp. 529–533 (cit. on p. 60).

[84] A. Plaat. *Deep reinforcement learning*. Springer, 2022 (cit. on p. 60).

[85] S. Song et al. «Smooth trajectory collision avoidance through deep reinforcement learning». In: *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*. Dec. 2022, pp. 914–919 (cit. on p. 64).

[86] C. Wilson and A. Riccardi. «Improving the efficiency of reinforcement learning for a spacecraft powered descent with Q-learning». In: *Optimization and Engineering* 24 (Mar. 2023), pp. 223–255 (cit. on p. 65).

[87] H. Yuan and D. Li. «Deep reinforcement learning for rendezvous guidance with enhanced angles-only observability». In: *Aerospace Science and Technology* 129 (Oct. 2022) (cit. on p. 65).

[88] J. Schulman et al. «Proximal policy optimization algorithms». In: *arXiv preprint arXiv:1707.06347* (July 2017) (cit. on p. 67).

[89] J. Schulman et al. «Trust region policy optimization». In: *International conference on machine learning*. June 2015, pp. 1889–1897 (cit. on p. 67).

[90] J. Schulman et al. «High-dimensional continuous control using generalized advantage estimation». In: *arXiv preprint arXiv:1506.02438* (June 2015) (cit. on p. 68).

[91] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. «Stable-Baselines3: Reliable Reinforcement Learning Implementations». In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: `http://jmlr.org/papers/v22/20-1364.html` (cit. on p. 77).