POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Meta-Reinforcement Learning for Spacecraft Proximity Operations Guidance and Control in Cislunar Space

Tesi di Laurea Magistrale in
Space Engineering - Ingegneria Spaziale

Author: **Giovanni Fereoli**

Student ID: 990939
Advisor: Prof. Pierluigi Di Lizia
Co-advisors: Prof. Hanspeter Schaub
Academic Year: 2022-2023

Abstract

In order to address the challenges of future space exploration, new lightweight and model-free guidance algorithms are necessary to make spacecraft completely autonomous. In recent years, autonomous spacecraft guidance has been a subject of intense research, and in the near future, this technology will be a great advantage for proximity operations in the cislunar space. For instance, NASA's Artemis program plans to establish a lunar Gateway, and this type of autonomous maneuver, besides the nominal Rendezvous and Docking (RVD) ones, is also necessary for the assembly and maintenance procedures.
In this context a Meta-Reinforcement Learning (Meta-RL) algorithm is applied to address the real-time relative optimal guidance problem of a spacecraft in the cislunar environment. Non-Keplerian orbits have more complex dynamics, and classic control theory may be less flexible and more computationally expensive with respect to Machine Learning (ML) methods. Furthermore, Meta-RL is chosen for its peculiar and promising ability of "learning how to learn" through experience. It is an ML approach in which a model is trained on a variety of tasks in such a way that it becomes more efficient and effective at learning new ones.
A stochastic optimal control problem is modeled in the Circular Restricted Three-Body Problem (CRTBP) framework as a discrete time-scale Markov Decision Process (MDP). The agent, an LSTM-based network, is then trained with a state-of-the-art actor-critic algorithm known as Proximal Policy Optimization (PPO). Additionally, operational constraints and stochastic effects are considered to assess policy safety and robustness. An MLP-based agent and an optimal control solution using pseudospectral methods are also evaluated for comparison purposes. The resulting tool is a closed-loop controller able to autonomously guide a spacecraft in the context of cislunar proximity operations. It is able to approximate the optimal control solution with a very general and not hand-crafted algorithmic framework, guaranteeing at the same time high robustness and computational efficiency.

Keywords: Spacecraft Autonomy, Adaptive $G\&C$, Relative Dynamics, Reinforcement Learning, Cislunar Space

Abstract in lingua italiana

Per affrontare le sfide della futura esplorazione spaziale, occorrono nuovi algoritmi di guida computazionalmente efficienti e privi di modelli per rendere i satelliti completamente autonomi. In anni recenti, la guida autonoma dei satelliti è stata oggetto di intense ricerche e nel prossimo futuro questa tecnologia rappresenterà un grande vantaggio per le operazioni di prossimità nello spazio cislunare. Ad esempio, il programma Artemis della NASA prevede di stabilire un Gateway lunare, e questo tipo di manovre autonome, oltre alle manovre nominali Rendezvous and Docking (RVD), è richiesto anche per le procedure di assemblaggio e manutenzione.
In questo contesto, viene applicato un algoritmo Meta-Reinforcement Learning (Meta-RL) per affrontare il problema della guida ottimale relativa in tempo reale di un satellite in ambiente cislunare. Le orbite non-kepleriane hanno dinamiche più complesse, e la teoria del controllo classica può essere meno flessibile e più computazionalmente costosa rispetto ai metodi di Machine Learning (ML). In aggiunta, Meta-RL è scelto per la sua particolare e promettente capacità di "imparare ad imparare" attraverso l'esperienza. È un approccio ML in cui un modello viene allenato su una varietà di compiti in modo tale che diventi più efficiente ed efficace nell'apprendimento di nuovi.
Un problema di controllo ottimale stocastico è modellato nel contesto del Circular Restricted Three-Body Problem (CRTBP) come un Markov Decision Process (MDP) su scala temporale discreta. L'agente, basato su una rete neurale composta da celle LSTM, viene poi addestrato con un algoritmo attore-critico allo stato dell'arte noto come Proximal Policy Optimization (PPO). Inoltre, i vincoli operativi e gli effetti stocastici sono considerati per valutare la sicurezza e la solidità della legge di controllo. Ai fini del confronto vengono valutati anche un agente basato su celle MLP e una soluzione di controllo ottimale che utilizza metodi pseudospettrali diretti. Il risultato è un controllore ad anello chiuso in grado di guidare autonomamente un satellite nell'ambito delle operazioni di prossimità cislunare. È in grado di approssimare la soluzione di controllo ottimale con un algoritmo flessibile e privo di modello, garantendo allo stesso tempo elevata robustezza ed efficienza computazionale.

Parole chiave: Autonomia Satelliti, G&C Adattivo, Dinamica Relativa, Apprendimento per Rinforzo, Spazio Cislunare

Acknowledgements

The stunning autumnal hues of Colorado, with its golden and crimson foliage, are coming to an end. It is time to take a break, reflect on things, and express my gratitude to those who have been a constant source of motivation and inspiration. Soon I will be heading back to my beloved Italy.

My deepest appreciation goes to my family, particularly my parents, Enrico and Alessandra, and my uncle and aunt, Luigi and Annarita. I am deeply grateful for the education I received while growing up, the encouragement, and the emotional support I have had over the past five years. Your boundless love, faith, selflessness, and assistance have been essential for me and my development. Without you, I would not have been able to accomplish my goals and I would not be the same person.

My heartfelt gratitude goes out to my incredible friends who were with me during my exciting college days in Milan, with its highs and lows. I am so thankful to my beloved friend Lorenzo, thank you for all the time we spent together, from our delightful cooking nights to our morning runs at 7 o'clock, and especially for always being there for me; the remarkable duo Luca and Marco, thank you guys for sharing, especially these last two years, this immense academic load, we started together and we are ending together and despite the uncertain future, I am sure we will always be together (we are a trio!). I am also extremely grateful to Matteo, Diego, Annachiara, and Luigia for the wonderful and unforgettable times we shared. Life would not be the same without all of you. I will always remember these years fondly and keep them close to my heart, thank you all.

I am immensely grateful for all the amazing American friends I have made (too many to name!) and all the other incredible people I have met in this country. I hope to see you all again, whether in Europe or the United States. In particular, I am especially grateful to a special individual whose path crossed mine during this American experience. You were only a small part of this extended academic journey, but you endlessly enriched this final rush, and I am certain to say that you have been and will remain one of the most essential and indispensable pieces of the puzzle. Thank you for being the wonderful person you are, Anna.

I am extremely indebted to the AVS laboratory research team for their invaluable aid and, most importantly, their love and friendship throughout my research. I am especially grateful to the machine learning guys John, Adam, and Mark for their generous backing, welcoming me into their group, and teaching me this captivating subject.

I am immensely thankful to Professor Pierluigi Di Lizia for the opportunities he has provided me and for the faith he has placed in me. He is an incredibly inspiring mentor, a devoted professor, and an indefatigable worker. I am sincerely grateful for all the guidance you have given me.

I am deeply grateful to Professor Hanspeter Schaub for giving me the opportunity to be part of his research team and for his numerous advice (e.g., KISS, or Keep It Short Stupid!). I have gained a great deal of invaluable soft skills, and my research abilities have increased significantly under your mentorship.

The last nine months have been some of the most remarkable of my life. My American experience has been an extraordinary educational opportunity, but I have treasured every second of this journey, which has transformed my life. I am overwhelmed by new companions, missing old ones, delighted to learn numerous new things, and serious and reflective about making my next major life choice. I had always envisioned doing a Ph.D., and, fortunately, sometimes life follows the internal flow of emotions and aspirations, granting you what you are striving to accomplish.

Giovanni

Boulder, United States
November 23^{th}, 2023

Contents

Introduction

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

0.1. Autonomous Proximity Operations

Rendezvous and Docking (RVD) is a technology that allows two spacecraft to come together in space with the same velocity and then physically attach. This is an essential capability for complex space missions, such as transferring crew, exchanging cargo, transferring fuel, repairing a satellite or building larger space structures. For the past fifty years, numerous countries have conducted numerous RVD missions [1]. The chaser spacecraft can be divided into more than ten series, including Gemini [2], Apollo [3], Space Shuttle [4], Demonstration of Autonomous Rendezvous Technology (DART) [5], Soyuz and Progress spacecraft from Russia or the former Soviet Union [6], the Cygnus expendable cargo from the United States [7], the automated transfer vehicle (ATV) from the European Space Agency (ESA) [8] and the H-II Transfer Vehicle (HTV) [9] from Japan. The last three spacecrafts mentioned are unmanned, and their RVD missions are completely automated, thus necessitating the development of complex autonomous Guidance, Navigation, and Control (GNC) algorithms.

A Rendezvous and Docking (RVD) process generally consists of four distinct steps: phasing, close-range rendezvous (which is divided into homing and closing), final approach, and docking (as seen in Figure 1). During the phasing phase, the chaser performs maneuvers to adjust the phase angle between the two spacecrafts, reduce the differences in the orbital plane, increase the orbital height, and initiate relative navigation. The homing phase is when the chaser autonomously controls itself and the final position of this phase is P_2, a station-keeping point located a few kilometers from the target. The main goals of this phase are to acquire the target orbit and reduce the relative velocity. The closing phase is when the chaser reduces the relative distance further and its position is transferred to P_3, a station-keeping point located hundreds of meters away from the target.

The final approach phase begins at P_3 and ends when the chaser touches the target and docks. Absolute GNC sensors and algorithms are used mainly during the phasing stage, while relative ones are employed during the close-range rendezvous and subsequent stages.

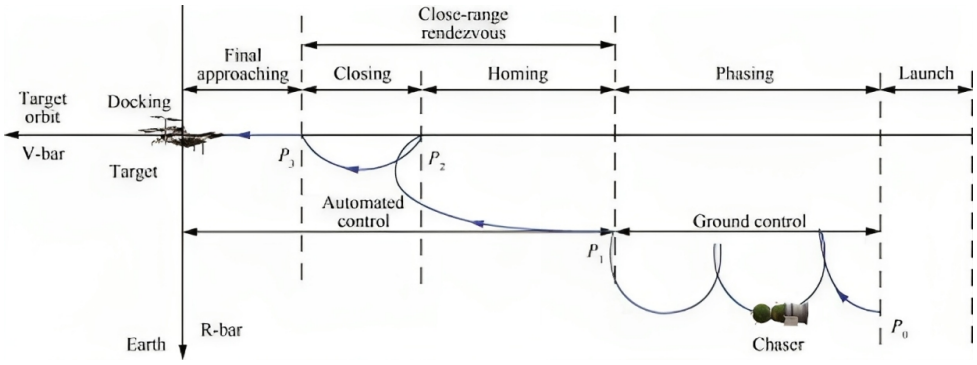


Figure 1: Soyuz spacecraft Rendezvous and Docking (RVD) operational phases with the International Space Station (ISS). Source: Reference [10].

In the past, astronauts had to manually manage the close-range or final approach rendezvous phases. Today, autonomous methods are used to guarantee robustness and safety. Autonomous G&C techniques are essential for the most intricate spacecraft proximity operation missions, such as on-orbit servicing, on-orbit assembly and inspection, active space debris removal, and more. This kind of problem is usually formulated as an Optimal Control Problem (OCP) and solved with active set or interior point techniques after a transcription [10], which are also called OCP direct methods. However, these methods are open-loop, unable to manage unpredictable circumstances without a controller, and too computationally expensive to be executed on-board. Autonomous G&C algorithms require two fundamental ideas: *robustness*, which should be achieved by satisfying numerous requirements, such as closed-loop control, for mission safety, and *computational efficiency* for on-board implementation. Nowadays, to acquire this capability, the high-level task is usually transformed into a precalculated reference trajectory (Guidance) and then tracked by a controller (Control). Several efficient algorithms for on-board implementation have been suggested, including those in Reference [11, 12], as well as the following:

- Guidance algorithms: Artificial Potential Field (APF) [13], Proportional Navigation (PN) [14], Zero-Effort-Miss/Zero-Effort-Velocity (ZEM/ZEV) [15] and more;
- Control algorithms: $\mathbb{H}_\infty$ Control [16], State-Dependent Riccati Equation (SDRE) Control [17], Linear Quadratic Regulators (LQR) [18], Model Predictive Control (MPC) [19], Sliding Mode Control (SMC) [20] and more.

In 1965, the Gemini 4 vehicle attempted the first rendezvous in history, using the Straight-

Line approach $\Delta V = \left(r_f - r_i\right)/\Delta t - v_i$ due to the limited understanding of orbital mechanics. Unfortunately, it was unable to rendezvous with its Titan II launch vehicle's upper stage, as both the target and chaser were still in LEO. Since then, technology has advanced significantly; however, for autonomous spacecraft G&C algorithms to be able to manage more intricate dynamical environments, where engineering modeling would be unfeasible, they must become *model-free* and their *computational efficiency* must be enhanced so they can be executed on smaller, more cost-effective spacecraft, while still being highly dependable and secure. An additional benefit would be to have an *integrated* G&C law, which would not require breaking down the high-level task into reference computation and tracking. Reinforcement Learning (RL) has the qualities mentioned above and, furthermore, since it is able to directly optimize the task-level goal while being able to use domain randomization to handle model uncertainty, it enables the discovery of more empirically reliable control responses, as stated by Song et al. [21].

0.2. Cislunar Space and Near-Rectilinear Orbits

On the road to a manned exploration of Mars, NASA is aiming to build a space station in cislunar space as part of the ARTEMIS project in cooperation with multiple international organizations and private companies in the next decade [22]. The station, also known as the Lunar Gateway, will be located on the *Southern 9:2 Resonant Near-Rectilinear Halo Orbit (NRHO)* around the Lagrangian Point L_2 of the Earth-Moon system. The station will be built gradually, and a series of rendezvous and docking missions will be necessary to carry out the actual construction, as well as to refurnish it once operational. *Autonomous Rendezvous, Proximity Operations and Docking (ARPOD)* capabilities in cislunar space would be a great advantage for the deployment and the overall functioning of the Lunar Gateway.

The use of a Southern L_2 9:2 Resonant NRHO offers a variety of advantages. According to Lee [22], these include:

1. Low orbit maintenance costs due to neutral stability characteristics;
2. The ability to reach the lunar surface within half a day from the orbit perilune;
3. Four-body Ballistic Lunar Transfers (BLTs) can deliver cargo to the NRHO with low fuel requirements, and it can also be used to depart for interplanetary destinations;
4. Enhanced visibility on the lunar far side, enabling continuous or nearly continuous communication, which significantly improves the overall infrastructure for lunar missions;

5. Low fuel requirements for a splashdown in the northern emisphere of Earth;
6. Absence of Sun eclipses (Figure 2), which can be advantageous for mission planning and operation.

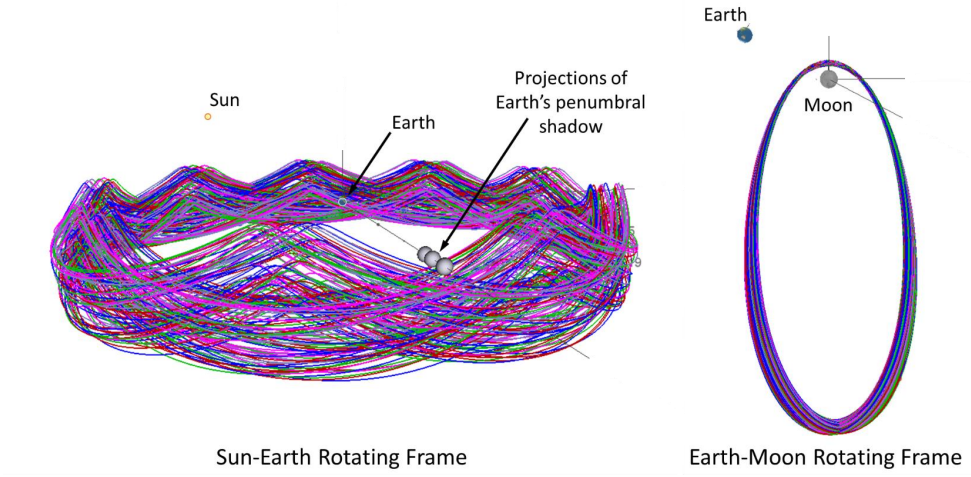


Figure 2: Southern L_2 9:2 Resonant Near-Rectilinear Halo Orbit (NRHO) of the Earth-Moon system eclipse avoidance property seen by the Earth-centered Sun-Earth synodic frame and by Moon-centered Earth-Moon synodic frame. Source: Reference [22].

A comparison between different lunar orbits can be seen in Figure 3 for completeness.

Orbit Type	Earth Access (Orion)	Lunar Access (to Polar LLO)	Crewed Spacecraft Stationkeeping	Crewed Spacecraft Communication	Crewed Spacecraft Thermal
Low Lunar Orbit (LLO)	Infeasible	ΔV = 0 m/s, ΔT = 0	50 m/s + per year	50% Occulted	Radiators Insufficient
Prograde Circular Orbit (PCO)	Marginally Feasible	ΔV < 700 m/s, ΔT <1 day	0 m/s for 3 years	Unknown	Unknown
Frozen Lunar Orbit	Marginally Feasible	ΔV = 808 m/s, ΔT <1 day	0 m/s	Frequent Occulation	Unknown
Elliptical Lunar Orbit (ELO)	Marginally Feasible	ΔV = 953 m/s, ΔT <1 day	>300 m/s	Frequent Occulation	Unknown
Near Rectilinear Orbit (NRO)	Feasible	ΔV = 730 m/s, ΔT = .5 day	<10 m/s per year	No Occulation	Radiators Sufficient
Earth-Moon L2 Halo	Feasible	ΔV = 800 m/s, ΔT = 3 days	<10 m/s per year	No Occulation	Radiators Sufficient
Distant Retrograde Orbit (DRO)	Feasible	ΔV = 830 m/s, ΔT = 4 days	0 m/s	Infrequent Occulation	Radiators Sufficient

Legend	Favorable	Marginal	Unfavorable

Figure 3: Comparison of various cislunar space orbits properties. Source: Reference [23].

In order to understand the behavior of rendezvous and docking operations in this circumstance, a comprehensive examination is necessary, as studied by Reference [24, 25].

Vehicles will maneuver in a non-Keplerian environment, making the dynamics highly nonlinear and chaotic. The methods and protocols used in the Apollo/Shuttle programs and ATV/HTV automated ISS operations, which are based on the assumption of a strong central gravitational field and linearized dynamics, are not applicable, as stated by Lizy-Destrez et al. [25]. The complexity of rendezvous is further compounded by the numerous conditions and constraints that must be met. The target station may require safety zones, approach-trajectory corridors, and hold points along the way to check the accuracy of the chaser's trajectory and switch between suitable sensor suites. Any state of the chaser vehicle outside the nominal limits of the approach trajectory could lead to a collision with the target, which would be hazardous to the crew and vehicle. These restrictions are already difficult to manage in a strong central body gravity field, so in the context of the cislunar space, new RVD algorithms must be studied and taken into account.

Lizy-Destrez et al. [25] mainly analyzed, in addition to proposing a simple guidance strategy, all the possible safety constraints that can be applied in this context, such as a keep-out sphere and an approach corridor aligned with the docking port. The article also pointed out that from a dynamical point of view linearized solutions, due to the three-body effect, are not reliable and, in order to design properly G&C solutions, a nonlinear equation of motion should be exploited. Colombi et al. [24] pointed out the same consideration, also claiming the possibility of reaching the target orbit by exploiting the stable/unstable manifolds (i.e., the typical dynamical structure of a multibody system) to have safe and fuel-optimal maneuvers. The works of Reference [26, 27] delve further into these concepts by proposing a fully safe RVD strategy in the cislunar space. This approach involves the use of manifold exploitation and the selection of optimal holding points for passive collision avoidance during the far-range rendezvous, as well as a State Dependant Riccati Equation (SDRE) control strategy during the final approach for active collision avoidance. A comparable study was conducted by Lorenzo Bucci and Lavagna [28]. Generally, there is a lack of research on RVD strategies and GNC algorithms for multibody dynamics. Very few papers and studies have been published on this topic.

0.3. Why Meta-Reinforcement Learning?

In various control research areas, such as robotics [29], automotive [30] and aerospace [31], *Deep-Reinforcement Learning (Deep-RL)* has been successful. This branch of machine learning uses *Artificial Neural Networks (ANNs)* that are trained through the interaction of the environment to tackle complex sequential decision-making problems known as *Markov Decision Process (MDPs)*. Neural networks have been shown to be capable of

accurately estimating any given function [32], making them a great choice to learn (not designing as traditional controllers) an optimal closed-loop G&C policy by estimating the solution of the Hamilton-Jacobi-Bellman (HJB) equations [33]. Before the training phase, the engineer focuses only on defining the MDP and adjusting the training hyperparameters. Once the agent has been trained on-ground and the optimal policy has been identified, its execution on-board requires minimal computing power. This process is illustrated in Figure 4. In the field of spacecraft G&C, these techniques have been used for a variety of purposes, such as orbital transfer [34–36], pinpoint planetary landing [37], hovering around small bodies in unknown gravity [38], rendezvous and proximity operations [39–43], impact time control [44], and planning images of small bodies [45, 46].

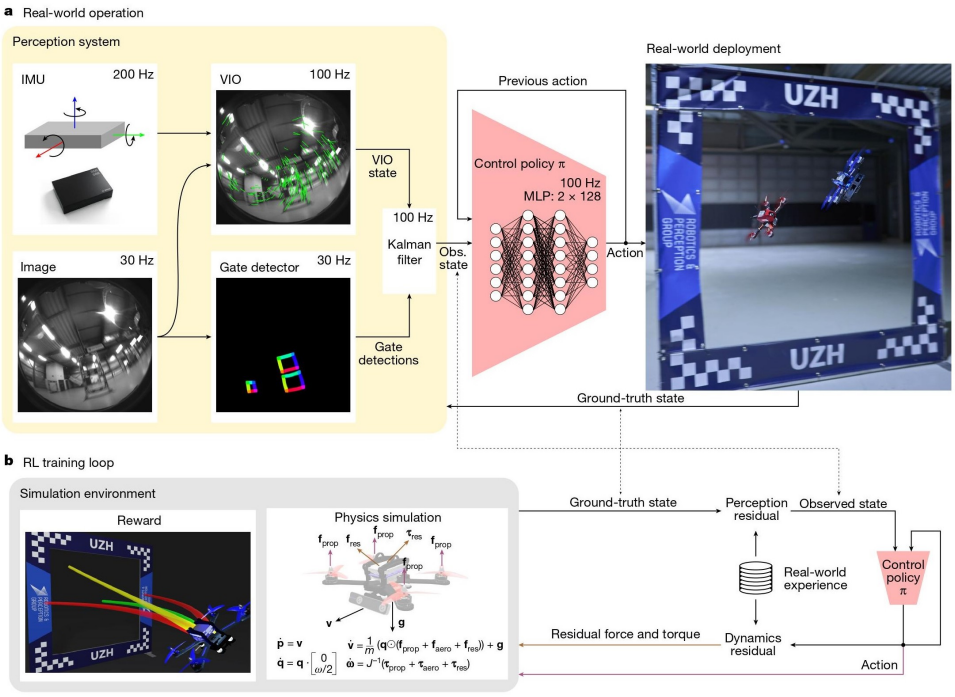


Figure 4: SWIFT system developed by Robotics and Perception Group, University of Zurich. Source: Reference [21].

Reinforcement Learning (RL) addresses control challenges without requiring exact mathematical equations of dynamics or stringent assumptions. It can benefit from domain randomization to address model uncertainty; agents can be trained with high-fidelity simulators that have stochastic models and real-time telemetry data [21]. Although these techniques are closed-loop and provide a great deal of flexibility, they are still not considered as reliable or safe as traditional methods. Neural networks are capable of learning within the boundaries of training distributions, but they tend to be unsuccessful when trying to extrapolate beyond these distributions. This could potentially lead to an unstable

guidance law if the spacecraft encounters states that are not within the training distribution range. Traditional reinforcement learning has another major drawback in that it requires a large amount of experience to learn even the most basic tasks. The integration of *Long Short-Term Memories (LSTMs)* into a Reinforcement Learning (RL) agent, as proposed by Hochreiter et al. [47], can be used to tackle these problems. This is called *Meta-Recurrent Neural Network (Meta-RNN)*, a specific instance of *Meta-Reinforcement Learning (Meta-RL)* or "learning to learn" [48, 49]. Recurrent agents differ from traditional *Multi-Layer Perceptrons (MLPs)* in that they can store temporal data in their hidden states, which act as internal memory. This leads to increased robustness and improved sample efficiency when learning tasks with randomized parameters. The policy obtained will be adaptive as the hidden states of the recurrent layers will be altered as observations are made during the trajectory.

Meta-RL has been shown to be more successful than traditional RL in tackling unmodeled dynamics and actuator failures [50], multi-task scenarios with uncertain initial conditions [35], and partially observable dynamics [42]. This approach has been applied with success to a variety of applications, such as planetary landing [51], control of underactuated cubesats [52], finite-thrust rendezvous missions [35], interplanetary trajectory design [53], and close proximity operations of asteroids [50, 54, 55]. All of these studies employ a particular actor-critic reinforcement learning algorithm, *Proximal Policy Optimization (PPO)*, to optimize Meta-RL policies. PPO has been shown to be highly effective for continuous control applications and is currently the state-of-the-art method [56]. By analyzing the characteristics of Meta-RL and its adaptive G&C, it can be seen that when testing, the agent's next action is based not only on the current state and action, but also on any external factors that were not taken into account during training. This is due to the recurrent layers, which cause the hidden states to vary depending on the observations obtained from the environment during the controlled trajectory. The hidden state is able to store information and learn during training how to modify the internal dynamics of the LSTM to generate the most suitable output for the task-level goal during testing. Therefore, Meta-RL is the ideal solution for situations that involve complex time-varying dynamics, uncertain models, and the possibility of failure.

The latest remarks have focused on the utilization of Meta-RL for a variety of spacecraft G&C applications. However, when all the reinforcement learning solutions in the context of spacecraft proximity operations that are available in the literature are taken into account, there is still a dearth of research. Scorsoglio et al. [57] investigated the potential of using RL to develop an adaptive ZEM/ZEV algorithm with safety guarantees for RVD spacecraft in cislunar space. Federici et al. [40] applied RL to create an autonomous

proximity operations algorithm in the xy-plane with guidance constraints in a Low Earth Orbit (LEO). Hovell and Ulrich [41] used RL to develop a Deep-Guidance strategy (i.e., a reference RL-based trajectory fed into an LQR controller) in LEO. Oestreich et al. [58] employed machine learning techniques with an LQR controller as a reference trajectory, while Broida and Linares [59] used it without optimality guarantees. Up until now, no research has been conducted on the use of RL for spacecraft proximity operations guidance and control that take into account fuel efficiency, three-dimensional motion, complex dynamics such as the Circular Restricted Three-Body Problem (CRTBP), and safety requirements, without relying on any classical optimal control theory.

0.4. Outline and Research Questions

Outline

1. Theoretical Background: Rendezvous and Docking (RVD) operations safety requirements, Cislunar space absolute and relative dynamics, fundamentals of Reinforcement Learning (RL) and Artificial Neural Networks (ANNs), Meta-Reinforcement Learning (Meta-RL);
2. Problem Formulation: RVD as Optimal Control Process (OCP) and Markov Decision Problem (MDP) definition, ANNs and hyperparameters selection;
3. Numerical Results: LSTM-policies training and testing, robustness to disturbances and computational efficiency assessment;
4. Benchmarks: MLP-policies training and testing, optimality assessment through state-of-the-art Optimal Control Problem (OCP) pseudospectral direct solution, and Lyapunov's Direct Method for asymptotic stability evaluation.

Research Questions

This project is centered on two main research questions, as outlined below. Their comprehensive exploration, informed by the acquired results and numerical data, is detailed and discussed in Chapter 5.

1. To what extent can a Meta-Reinforcement Learning algorithm be used as an autonomous spacecraft proximity operations guidance and control algorithm in the cislunar space? Does it meet safety requirements while being computationally efficient for on-board execution?

2. What are the benefits of using Long Short-Term Memories (LSTMs) instead of basic Multi-Layer Perceptrons (MLPs) when it comes to a reinforcement learning agent for spacecraft guidance and control applications?

0.5. Bibliographic Disclaimer

For the past nine months, I have been a Visiting Researcher at the University of Colorado Boulder, working in the Autonomous Vehicle Systems (AVS) laboratory under the guidance of Prof. Hanspeter Schaub. I have had the privilege of presenting the progress of my research on multiple occasions. My findings were featured at the 8th AIAA Intelligent Systems Workshop in Boulder, Colorado, which took place on 24th and 25th July 2023. Furthermore, my abstract has been accepted for the 46th AAS Guidance, Navigation, and Control Conference in Breckenridge, Colorado, from February 1st to 7th 2024.

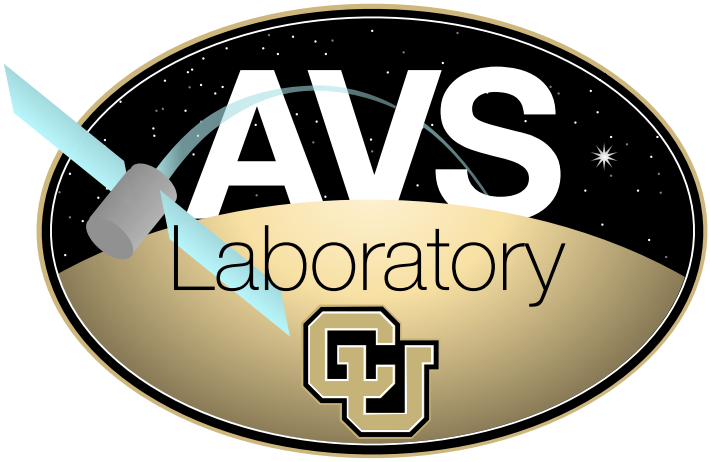


Figure 5: The logo of the Autonomous Vehicle Systems (AVS) laboratory at University of Colorado Boulder.

1 | Theoretical Background

> When a pain stimulus occurs, all tentative entries are canceled, and when a pleasure stimulus occurs, they are all made permanent.
>
> Alan Turing

This chapter provides a comprehensive theoretical foundation for the study of Rendezvous and Docking (RVD) in cislunar space, including safety requirements and dynamics. Additionally, it examines modern Machine Learning (ML) fundamentals such as the Proximal Policy Optimization (PPO) algorithm and Long Short-Term Memory (LSTM) neural networks to introduce the concept of Meta-Reinforcement Learning (Meta-RL). This theoretical foundation is necessary to progress to a systematic examination of implementation and numerical results.

1.1. Safety Requirements in Cislunar Space

NASA and ESA have been conducting manned rendezvous and docking missions since Gemini 6 in 1966, and many of the techniques used in the Apollo and Shuttle programs are still applicable to ATV and HTV rendezvous with the ISS. However, these techniques are based on the assumption of two vehicles operating in a near circular orbit in a strong central gravity field, which is not the case in the cislunar environment. Therefore, the constraints and safety procedures derived from operating in the vicinity of a strong central body are no longer applicable. Additionally, since the gravity field is shallow in cislunar orbits, the relative dynamics of proximity motion is almost straight [25], meaning that the "carving" characteristics of LEO trajectories, which govern the ISS safety standards, cannot be used.

Investigations into 3 Degrees-Of-Freedom (DOF) relative dynamics in non-Keplerian environments are now necessary to identify potential issues in the design of the Lunar Gateway and to adapt the Rendezvous and Docking (RVD) protocols, which have been established in LEO for the International Space Station (ISS), to the new non-Keplerian environment. Here is a helpful collection of definitions [1, 60, 61]:

- Rendezvous: this phase begins when the chaser vehicle is confirmed to be in an orbit established in space relative to the target vehicle, and ends at the docking start;
- Docking: the beginning of this phase is marked by the two vehicles' docking mechanisms coming into contact, and it concludes when the hard-mating hooks/latches are firmly connected. Once the initial contact is established, the rendezvous is finished. The requirements of the terminal relative state depend on the vehicle and the docking mechanism, but usually for a successful maneuver [61], the relative position shall not exceed 1 m and the relative velocity[1] shall be no more than 0.1 m/s;
- Proximity Operations: this stage encompasses the stages mentioned above and many more, such as fly-around, undocking, and departure.

The mission scenario taken into account in this work involves an automated RVD between a chaser and the target orbiting a Near-Rectilinear Halo Orbit (NRHO). Before constructing a secure trajectory, certain safety boundaries and minimum hold points (places where the chaser will stop relatively near the target and decide whether to abort the rendezvous or continue the mission) are established by Gerstenmaier et al. [62]:

- Rendezvous Sphere (RS): the RS is a 10 km radius sphere around the target. Outside of this perimeter, passive safety, defined as the strategy of avoiding collisions by using a collision-free trajectory, shall be guaranteed. Here the spacecraft waits for *GO Rendezvous Entry*;
- Approach Sphere (AS): the AS is a 1 km radius sphere around the target. Active safety, which involves detecting a potential collision and executing a *Collision Avoidance Maneuver (CAM)* by actively using the chaser propulsion system, shall be provided within this region. At this point the spacecraft waits for *GO Approach Initiation*;
- Keep-Out Sphere (KOS): the KOS is a 200 m radius sphere around the target. Here the spacecraft waits for *GO Final Approach*;
- Approach/Departure Corridors: the approach and departure corridors are $\pm 20^{\circ}$ centered to the docking port axis within KOS. The final decision to allow vehicles to have contact is *GO Docking*.

The Figure 1.1 and Table 1.1 give a thorough and formal description of the conditions

[1] In order to accomplish a soft-docking [60], the terminal relative velocity should not exceed 0.03 m/s. Nevertheless, research in the realm of RL has addressed RVD issues with less stringent requirements [58, 59] using the same ones used for the Apollo CSM/LM docking maneuver (`https://ntrs.nasa.gov/citations/19730066752`), which are considered suitable for this work as a preliminary evaluation of the proposed G&C algorithm.

and requirements for a successful RVD operation. However, this work focuses on the final approach phase of rendezvous, and the specific algorithmic requirements (which must comply with those just mentioned) are outlined in Section 2.1.

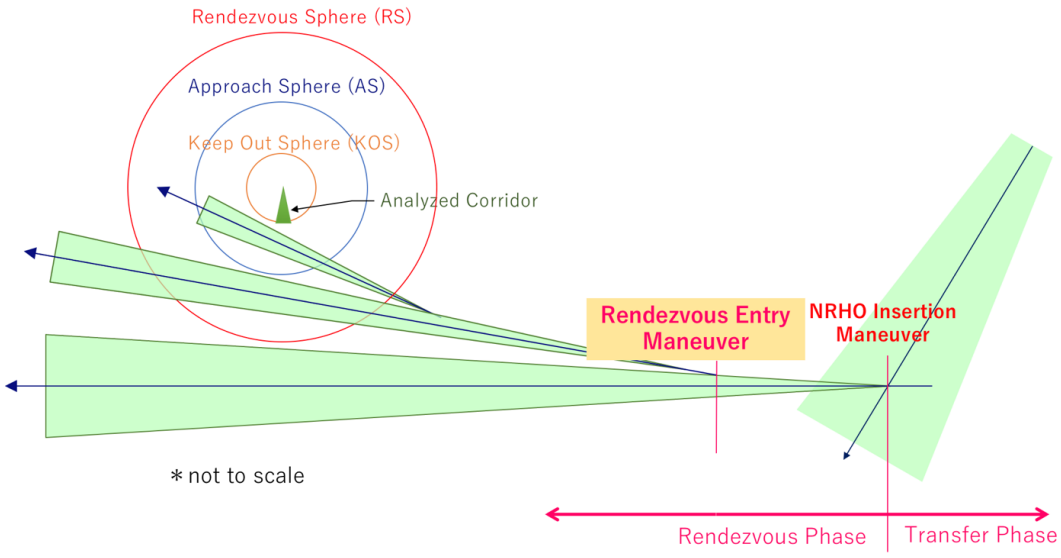


Figure 1.1: Notional concept for a successful Rendezvous and Docking (RVD) in the cislunar space. Source: Reference [62].

Requirement	Description
RVD-001	Rendezvous Sphere (RS) shall have a radius of 10 km.
RVD-002	Approach Sphere (AS) shall have a radius of 1 km.
RVD-003	Keep-Out Sphere (KOS) shall have a radius of 200 m.
RVD-004	Approach Corridor shall have half-angle of $\pm 20°$.
RVD-005	Terminal relative position shall be less than 1 m.
RVD-006	Terminal relative velocity shall be less than 0.1 m/s.

Table 1.1: Minimum Guidance, Navigation and Control (GNC) requirements for Rendezvous and Docking (RVD) maneuvers in cislunar space.

Depending on the mission and vehicle, various holding points can be included; generally, outside of the approach sphere, stable/unstable manifolds [24] should be used to create passive safe trajectories, while inside it closed-loop control and collision avoidance algorithms should be designed. These procedures are necessary due to the presence of various sources of orbital deviation: unmodeled acceleration and disturbances, GNC errors, failures, and so on.

1.2. Cislunar Space Dynamics

1.2.1. Absolute Dynamics

Synodic Reference Frame

The *Circular Restricted Three-Body Problem (CRTBP)* can be described by a synodic reference frame that rotates at a constant angular velocity ω_S in accordance with the motion of two massive bodies around their barycenter. This frame is centered at the barycenter O of the system and is constructed in such a way that the $\hat{\mathbf{x}}$-axis points from the larger body to the smaller one at any given time, the $\hat{\mathbf{z}}$-axis is aligned with the angular momentum of the system, and the $\hat{\mathbf{y}}$-axis completes the right-handed coordinate system, as illustrated in Figure 1.2. This can be condensed to $\mathcal{C}_O : \{\hat{\mathbf{x}},\ \hat{\mathbf{y}},\ \hat{\mathbf{z}}\}$, while $\mathcal{C}_O : \{\hat{\mathbf{X}},\ \hat{\mathbf{Y}},\ \hat{\mathbf{Z}}\}$ is the original non-rotating reference frame.

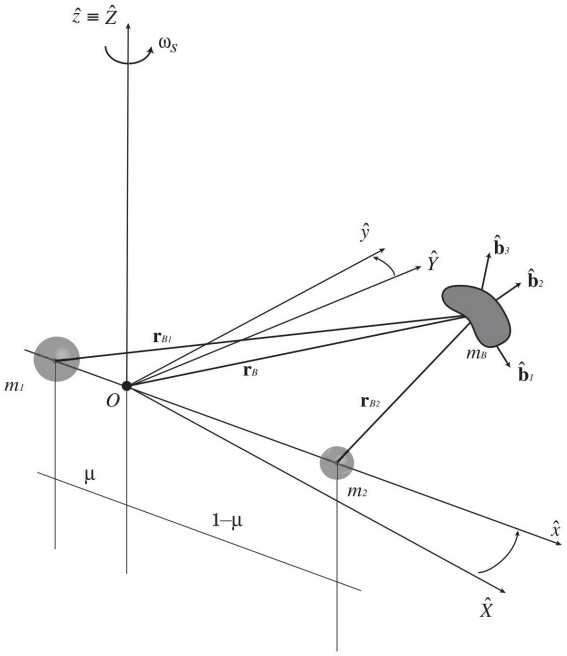


Figure 1.2: Circular Restricted Three-Body Problem (CRTBP) synodic reference frame. Source: Reference [63].

The CRTBP equations of motion become autonomous in this reference frame, since the angular rate of the system $\boldsymbol{\omega} = \omega\hat{\mathbf{z}}$ is a constant.

Circular Restricted Three-Body Problem

To explain the motion of a massless particle, such as a spacecraft, in the presence of two massive bodies, such as Earth and the Moon, the most basic model is the *Restricted Three-Body Problem (RTBP)*. This model takes into account the mass of the body m_B, and the masses of the two massive bodies, m_1 and m_2, with the assumptions that $m_B \ll m_1,\ m_2$ and $m_1 > m_2$. It is possible to simplify the problem and gain useful qualitative insights to help design practical low-energy orbit transfers by introducing some assumptions. These include expressing the equations in a rotating reference frame, called the synodic frame of Subsection 1.2.1, and supposing that the motion of the primary bodies is circular. This leads to the *Circular Restricted Three-Body Problem (CRTBP)*[64], which is usually studied in its normalized form. In this case, the magnitude of the separation between the two massive bodies r_{12}, the angular velocity of the reference frame ω, and the total mass of the system $m_T = m_1 + m_2$ are equal to one. A key definition is the mass parameter μ, which provides a clear definition of the three-body system:

$$\mu = \frac{m_2}{m_1 + m_2} \tag{1.1}$$

The mass parameters and the non-dimensional units typically used for the Earth-Moon system are described in Table 1.2 [65, 66].

Parameter	Value
$\boldsymbol{\mu}$	0.0121505856
Mass Unit (MU)	$6.0458 \cdot 10^{24}\ kg$
Distance Unit (DU)	$3.844 \cdot 10^{8}\ m$
Time Unit (TU)	$375200\ s$

Table 1.2: Characteristic quantities of the Circular Restricted Three-Body Problem (CRTBP) for the Earth-Moon system.

Consequently, due to normalization, the universal gravitational constant has also been set to $G = 1$, and the orbital period of the two massive objects around the barycenter is $T = 2\pi$. The position of m_1 is $\mathbf{x}_1 = -\mu\ \hat{\mathbf{x}}$, and the position of m_2 is $\mathbf{x}_2 = 1 - \mu\ \hat{\mathbf{x}}$. The equations of motion $\dot{\mathbf{x}} = f(\mathbf{x})$ for the CRTBP are given in Equation 1.2, along with the definitions of $\mathbf{r}_1$ and $\mathbf{r}_2$ as the distance of the third body from the two massive ones. The mass parameter μ is the only factor that affects the dynamics.

$$\begin{aligned}
\ddot{x} &= 2\dot{y} + x - (1-\mu)\frac{x+\mu}{\|\mathbf{r}_1\|^3} - \mu\frac{x+\mu-1}{\|\mathbf{r}_2\|^3} \\
\ddot{y} &= -2\dot{x} + y - (1-\mu)\frac{y}{\|\mathbf{r}_1\|^3} - \mu\frac{y}{\|\mathbf{r}_2\|^3} \\
\ddot{z} &= (1-\mu)\frac{z}{\|\mathbf{r}_1\|^3} - \mu\frac{z}{\|\mathbf{r}_2\|^3}
\end{aligned} \tag{1.2}$$

$$\mathbf{r}_1 = [x+\mu,\ y,\ z] \tag{1.3}$$

$$\mathbf{r}_2 = [x+\mu-1,\ y,\ z] \tag{1.4}$$

When considering the Earth-Moon system, particularly in NRHO, two disturbances can be taken into account to improve the accuracy of the model without making it overly complex with ephemeris: Solar Radiation Pressure (SRP) and the gravitational effect of the Sun as a fourth body. To model the first one, the spacecraft can be thought of as a perfect reflective sphere with uniform optical properties, and the SRP can be seen as a continuous and opposing force to the sunlight that is hitting it (Figure 1.3). This is known as the cannonball assumption [67], a first-order approximation used for the LAGEOS mission, and is implemented as follows:

$$\mathbf{a}_{SRP} = -C_r \frac{AP_{SRP}}{m}\left(\frac{1\ AU}{\|\mathbf{r}_4\|}\right)^2 \frac{\mathbf{r}_4}{\|\mathbf{r}_4\|} \tag{1.5}$$

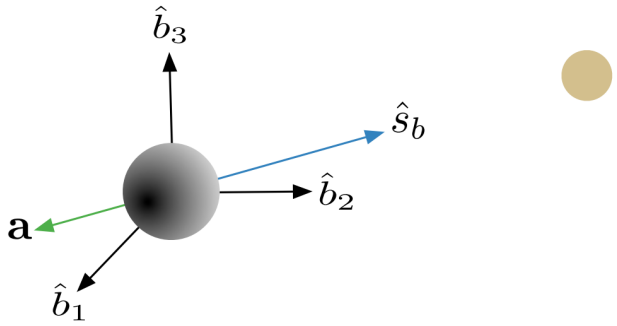


Figure 1.3: Solar Radiation Pressure (SRP) disturbance modeling through the Cannonball assumptions. Source: Reference [67].

Where C_r is the reflectivity coefficient of the spacecraft, $P_{SRP} = 4.56 \cdot 10^{-6}\ Pa$ is the pressure of the photons that impinge on the cross section A of the spacecraft, and $\mathbf{r}_4$ (Equation 1.6) is the position with respect to the Sun. Instead, the gravitational effect of a fourth body, such as the Sun, can be computed considering the *Bicircular Restricted*

Four-Body Problem (BRFBP)[68]. The Sun and the barycenter of the Earth-Moon system O moves in a circular coplanar Keplerian orbit around their shared barycenter $\hat{O}$. This four-body problem is not consistent, meaning that the motion of the Earth and Moon is not affected by the Sun's gravity. The model can be formulated as follows:

$$\mathbf{r}_4\left(x,\ y,\ z,\ t\right) = \left[x - r_s cos\left(\theta_s t\right),\ y - r_s sin\left(\theta_s t\right),\ z\right] \tag{1.6}$$

$$\Omega_{4B}\left(x,\ y,\ z,\ t\right) = \frac{m_s}{\|\mathbf{r}_4\|} - \frac{m_s}{r_s^2}\left[x cos\left(\theta_s t\right) + y sin\left(\theta_s t\right)\right] \tag{1.7}$$

$$\mathbf{a}_{4B} = \boldsymbol{\nabla}\Omega_{4B}\left(x,\ y,\ z,\ t\right) \tag{1.8}$$

Where Ω_{4B} is the BCRFBP pseudopotential in the Earth-Moon synodic frame, m_s is the mass of the Sun, w_s is the angular velocity and r_s is the radius of the circular orbit reflecting the Sun-O motion; thus, the current location of the Sun is $\left(r_s cos\left(\theta_s t\right),\ r_s sin\left(\theta_s t\right),\ 0\right)$. It is important to note that the Sun is not stationary in the chosen reference frame, resulting in a loss of autonomy. This is a great departure from the CRTBP as both the equilibrium points and the Jacobi integral vanish.

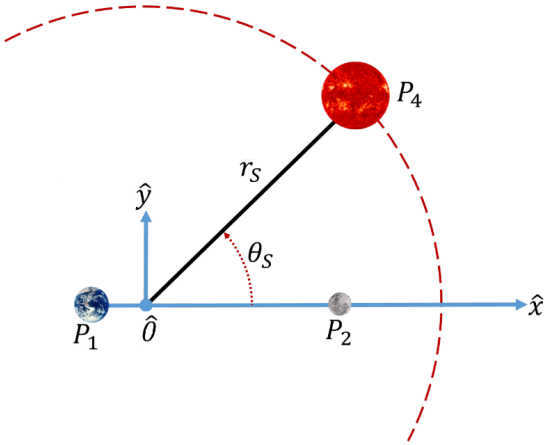


Figure 1.4: Bicircular Restricted Four-Body Problem (BRFBP) coplanar motion of the Sun around $\hat{O}$ with respect to the Earth-Moon synodic frame. Source: Reference [68].

Lagrange Points

A potential function $\Omega\left(\mathbf{x}\right)$ can be established in the CRTBP utilizing non-dimensional normalized synodic coordinates:

$$\Omega\left(\mathbf{x}\right) = \frac{1}{2}\left(x^2 + y^2\right) + \frac{1-\mu}{\|\mathbf{r_1}\|} + \frac{\mu}{\|\mathbf{r_2}\|} \tag{1.9}$$

The CRTBP has a set of autonomous *Ordinary Differential Equations (ODEs)*, thus equilibrium points may be present in the phase space. To discover them, the following equation must be solved:

$$\boldsymbol{\nabla}\Omega\left(\mathbf{x}\right) = \mathbf{0} \tag{1.10}$$

The five solutions of the Circular Restricted Three-Body Problem (CRTBP) are known as the five Lagrangian (or libration) points. All of them are located in xy-plane with $z = 0$. The first three, L_1, L_2, and L_3, are referred to as *collinear* and are found by setting $y = z = 0$. The last two, L_4 and L_5, are called *equilateral* and are located when $r_1 = r_2 = 1$. The Lagrange points of the Earth-Moon system are outlined in Table 1.3.

	x	y	z
L_1	0.8369151324	0	0
L_2	1.1556821603	0	0
L_3	-1.0050626453	0	0
L_4	0.4878494157	0.8660254038	0
L_5	0.4878494157	-0.8660254038	0

Table 1.3: Normalized positions of the five Lagrangian points, comprising three collinear and two equilateral points, in the synodic reference frame of the Earth-Moon system.

The Jacobian matrix of the system dynamics, when computing the eigenvalues of the collinear points, produces an unstable result, whereas the equilateral points are stable. Nevertheless, there are periodic orbits that exist around the unstable collinear points.

State Transition Matrix

The *State Transition Matrix (STM)* $\boldsymbol{\Phi}\left(t,\ t_0\right)$ of a dynamical system is used to monitor the divergence of a trajectory; it predicts how a slight alteration in the initial conditions will propagate along the trajectory. A particular instance of STM is the monodromy matrix, which is expressed as $\mathbf{M} = \boldsymbol{\Phi}\left(t_0 + T,\ t_0\right)$, and contains information on all the areas that a spacecraft would traverse during an orbit of period T. The STM can be determined by taking the partial derivatives of the state vector $\mathbf{x}\left(t\right) = \left(x,\ y,\ z,\ \dot{x},\ \dot{y},\ \dot{z}\right)$ with respect to the initial conditions as follows:

$$\mathbf{\Phi}\left(t,\ t_0\right) = \frac{\partial \mathbf{x}\left(t\right)}{\partial \mathbf{x}\left(t_0\right)} \tag{1.11}$$

The STM is propagated in time from its initial value $\mathbf{\Phi}\left(t_0,\ t_0\right) = \mathbf{I}$ employing the following equation:

$$\dot{\mathbf{\Phi}}\left(t,\ t_0\right) = \mathbf{A}\left(t\right)\mathbf{\Phi}\left(t,\ t_0\right) \tag{1.12}$$

$$\mathbf{A}\left(t\right) = \frac{\partial \dot{\mathbf{x}}\left(t\right)}{\partial \mathbf{x}\left(t\right)} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ \frac{\partial \ddot{x}}{\partial x} & \frac{\partial \ddot{x}}{\partial y} & \frac{\partial \ddot{x}}{\partial z} & 0 & 2 & 0 \\ \frac{\partial \ddot{y}}{\partial x} & \frac{\partial \ddot{y}}{\partial y} & \frac{\partial \ddot{y}}{\partial z} & -2 & 0 & 0 \\ \frac{\partial \ddot{z}}{\partial x} & \frac{\partial \ddot{z}}{\partial y} & \frac{\partial \ddot{z}}{\partial z} & 0 & 0 & 0 \end{bmatrix} \tag{1.13}$$

In this work, the Jacobian matrix $\mathbf{A}\left(t\right)$ of the dynamics of CRTBP (Equation 1.2) is analytically determined and the propagation of the STM is combined with the dynamical state equations, forming a system of 42 *Ordinary Differential Equations (ODEs)*.

Construction of Halo Orbits in the CRTBP

In order to gain insight into the behavior of dynamical systems and identify periodic solutions, three types of techniques can be used in this context [69]: Analytical Expansion Techniques, Poincaré Method, and Shooting Techniques. The latter are especially sensitive to the starting conditions, yet they are highly simple and computationally efficient, and thus they are employed in this study.

In the CRTBP framework, since Lagrange points are the five simplest solutions, it may be argued that the next simple solutions are the periodic *symmetric* orbits. There are other arbitrarily complicated orbits as well, for instance, not symmetric, but this is beyond the scope of this work. This study focuses on the use of a *Southern 9:2 Resonant Near-Rectilinear Halo Orbit (NRHO)* around the L_2 point of the Earth-Moon system, as mentioned in Section 0.2. Halo orbits are a well-known three-dimensional periodic solution to the CRTBP. These orbits are divided into two categories, Northern and Southern, and are symmetric in relation to the xz-plane. Lyapunov orbits, which are restricted to the xy-plane, can be used to generate an initial guess for an orbit through a bifur-

cation technique. Subsequently, a shooting technique (Subsection 1.2.1) is employed to calculate an elementary Halo orbit from the initial guess, and then a continuation scheme (Subsection 1.2.1) is used to obtain the desired one.

Moreover, the CRTBP has certain intriguing symmetries that can be used to create periodic orbits. They can be summarized as:

1. If $(x,\ y,\ z,\ \dot{x},\ \dot{y},\ \dot{z},\ t)$ is a solution, then also $(x,\ -y,\ z,\ -\dot{x},\ \dot{y},\ -\dot{z},\ -t)$. In other words, if a trajectory is mirrored across the xz-plane, there is also a trajectory in the opposite direction that is reflected;
2. If $(x,\ y,\ z,\ \dot{x},\ \dot{y},\ \dot{z},\ t)$ is a solution, then also $(x,\ y,\ -z,\ \dot{x},\ \dot{y},\ -\dot{z},\ t)$. Consequently, two distinct types of orbit can be found in the system, one in the North and one in the South.

This is called the *Mirror Theorem* by Parker and Anderson [69].

Single-Shooting Differential Correction

Differential Correction [65] is a targeting approach that utilizes the State Transition Matrix (STM) to adjust the initial conditions of a trajectory in order to meet a certain set of criteria. The objective of the single-shooting differential correction scheme is to find an initial state $\mathbf{x}_0 = (\mathbf{r}_0,\ \mathbf{v}_0)$ that will lead to a desired state $\hat{\mathbf{x}}_f = \left(\hat{\mathbf{r}}_f,\ \hat{\mathbf{v}}_f\right)$ at a later time t_f with a single propagation leg. The variables $\mathbf{x}_0$, $\hat{\mathbf{x}}_f$, and t_f can be constrained, free, or unknown, depending on the problem. The routine is allowed to iteratively adjust the unknown variables to meet the constraints within a certain tolerance. The STM is used to calculate the adjustment needed in the initial condition $\delta\mathbf{x}_0$ to remove any discrepancy $\delta\mathbf{x}_f$ at the end of the propagated trajectory. Therefore, the essential equation for adjustment is as follows:

$$\delta\mathbf{x}_f = \mathbf{\Phi}\left(t_f,\ t_0\right)\delta\mathbf{x}_0 \tag{1.14}$$

Beginning with an initial Southern L_2 Halo Orbit guess [66] for the Earth-Moon system, the procedure mentioned above can be used to locate the periodic orbit using the Mirror Theorem (Paragraph 1.2.1). In particular, starting from:

$$\mathbf{x}_0 = [1.120745458770128,\ 0,\ 0.011371823619780,\ 0,\ 0.175237781529976,\ 0] \tag{1.15}$$

The algorithm[2] constraints z_0 (Halo orbits are defined through the amplitude A_z on the z-axis) and, in an iterative manner, adjusts x_0 and $\dot{y}_0$ until $\dot{x}_f$ and $\dot{z}_f$ become zero when the trajectory, during its propagation, reaches the xz-plane for a second time at t_f. Equation 1.16 serves as the foundation for this differential correction technique: it specifies the desired change in the elements of the final state and then calculates the approximate adjustment to the initial state that is necessary to achieve such a modification.

$$\begin{bmatrix} \delta x_0 \\ \delta \dot{y}_0 \end{bmatrix} = \begin{bmatrix} \Phi_{41}\left(t_f, t_0\right) & \Phi_{45}\left(t_f, t_0\right) \\ \Phi_{61}\left(t_f, t_0\right) & \Phi_{65}\left(t_f, t_0\right) \end{bmatrix}^{-1} \begin{bmatrix} -\dot{x}_f \\ -\dot{z}_f \end{bmatrix} \tag{1.16}$$

Using a non-stiff multi-step Adams integrator with absolute and relative tolerances set to 10^{-13}, and solving the differential correction problem with a tolerance of 10^{-13}, the results are presented in Figure 1.5.

(a) Halo orbit before differential correction.

(b) Halo orbit after differential correction.

Figure 1.5: Single-shooting differential correction scheme applied to the initial conditions in Equation 1.15.

Continuation Method

Periodic orbits in the Circular Restricted Three-Body Problem (CRTBP) can be classified into families. Each family is composed of an infinite number of orbits that differ from each other in terms of their properties. All the orbits in the same family can be characterized by a single parameter, such as their position or velocity on a specific plane. Using a *continuation method* [70], it is possible to trace the entire family of orbits once a single periodic orbit in the CRTBP is known.

[2]The code for this algorithm can be found at `https://github.com/giovannifereoli/CRTBP_DifferentialCorr_Continuation`.

This method starts by changing some parameters of the known periodic orbit and then adjusting the new conditions using the differential correction technique described in Subsection 1.2.1. To guarantee the accuracy of the continuation method, only small modifications are made. In this study, the entire Southern L_2 Halo Orbit family of the Earth-Moon system is examined. To do this, the component z_0 is perturbed first and then the perturbation was shifted to the component x_0 when $\delta z_0 < \delta x_0$, with the differential corrector adjusted accordingly. Both perturbations were of magnitude 0.005 DU. Following these steps, Figure 1.6 is generated.

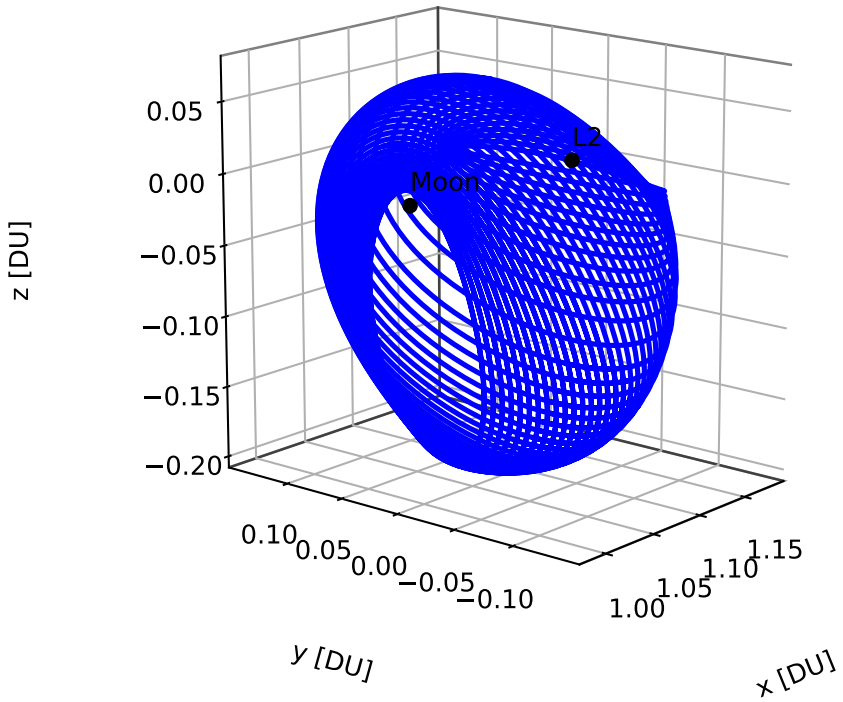


Figure 1.6: Southern L_2 Halo orbits family of the Earth-Moon system obtained combining differential correction and continuation schemes.

Among the orbits recently discovered, the one chosen for the spacecrafts' flight is the Southern 9:2 Resonant NRHO. This orbit is discussed in Section 0.2, but it is worth mentioning some other properties. A synodic resonant orbit is defined by a resonance ratio P/Q, where P is the number of orbital periods and Q is the number of lunar synodic periods. The Southern 9:2 Resonant NRHO has a period of 6.5 *days*, a perilune radius of approximately 3250 km, and an apolune radius of around 71000 km. This orbit is illustrated in Figure 1.7.

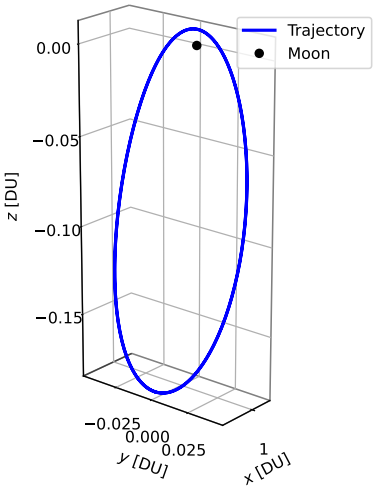


Figure 1.7: Southern L_2 9:2 Resonant Near-Rectilinear Halo Orbit (NRHO) of the Earth-Moon system.

1.2.2. Relative Dynamics

Relative Synodic Reference Frame

The relative motion between a target and a chaser spacecraft, both of generic masses m_T and m_C, can be expressed as well in the synodic frame outlined in Subsection 1.2.1. The origin of the three axes, considering $\delta\mathbf{x}$, is located at the target, which gives $\mathcal{C}_T : \{\hat{\mathbf{x}},\ \hat{\mathbf{y}},\ \hat{\mathbf{z}}\}$.

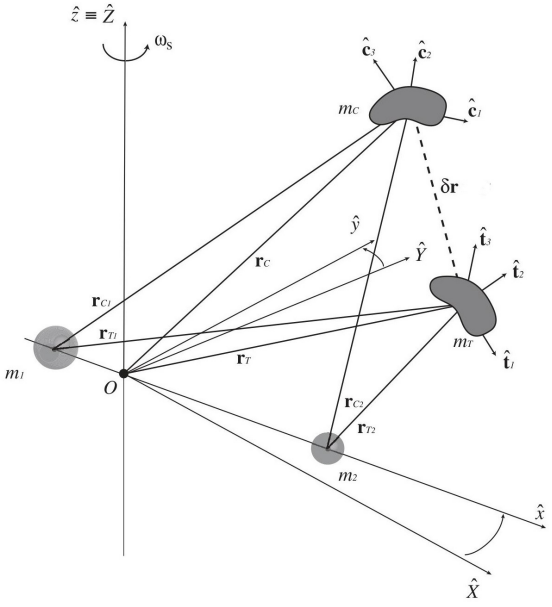


Figure 1.8: Relative synodic reference frame centered in the target body. Source: Reference [63].

Relative Circular Restricted Three-Body Problem (RCRTBP)

In the literature, many types of relative motion can be found for NRHOs. Lizy-Destrez et al. [25] studied the Clohessy–Wiltshire (CW) and Linearized Relative Equations (LRE). Scorsoglio et al. [43] implemented the CW equations near the perilune. Franzini and Innocenti [71] proposed a set of Non-Linear Relative (NLR) equations in the Local-Vertical/Local-Horizon (LVLH) frame, which require the computation of the target's angular velocity and acceleration vectors during its motion around the Moon. However, certain equations are not suitable for generalization or simplicity, so they are not used in this study.

The equations of motion for the Relative Circular Restricted Three-Body Problem (RCRTBP) can be obtained by subtracting the absolute equations of motion from Subsection 1.2.1 of the target and the chaser, resulting in $\delta\mathbf{x} = \mathbf{x}^C - \mathbf{x}^T$. These equations are expressed in the Relative Synodic Earth-Moon frame (Figure 1.8) and are given in Equation 1.17.

$$\begin{aligned}
\delta\ddot{x} &= 2\delta\dot{y} + \delta x + (1-\mu)\left[\frac{x^T+\mu}{\|\mathbf{r_{1T}}\|^3} - \frac{x^T+\delta x+\mu}{\|\mathbf{r_{1T}}+\boldsymbol{\rho}\|^3}\right] + \mu\left[\frac{x^T-\mu-1}{\|\mathbf{r_{2T}}\|^3} - \frac{x^T+\delta x+\mu-1}{\|\mathbf{r_{2T}}+\boldsymbol{\rho}\|^3}\right] \\
\delta\ddot{y} &= -2\delta\dot{x} + \delta y + (1-\mu)\left[\frac{y^T}{\|\mathbf{r_{1T}}\|^3} - \frac{y^T+\delta y}{\|\mathbf{r_{1T}}+\boldsymbol{\rho}\|^3}\right] + \mu\left[\frac{y^T}{\|\mathbf{r_{2T}}\|^3} - \frac{y^T+\delta y}{\|\mathbf{r_{2T}}+\boldsymbol{\rho}\|^3}\right] \\
\delta\ddot{z} &= (1-\mu)\left[\frac{z^T}{\|\mathbf{r_{1T}}\|^3} - \frac{z^T+\delta z}{\|\mathbf{r_{1T}}+\boldsymbol{\rho}\|^3}\right] + \mu\left[\frac{z^T}{\|\mathbf{r_{2T}}\|^3} - \frac{z^T+\delta z}{\|\mathbf{r_{2T}}+\boldsymbol{\rho}\|^3}\right]
\end{aligned} \tag{1.17}$$

$$\boldsymbol{\rho} = [\delta x,\ \delta y,\ \delta z] \tag{1.18}$$

$$\mathbf{r}_{1T} = \left[x^T+\mu,\ y^T,\ z^T\right] \tag{1.19}$$

$$\mathbf{r}_{2T} = \left[x^T+\mu-1,\ y^T,\ z^T\right] \tag{1.20}$$

They are highly advantageous, requiring no extra assumptions in comparison to the Circular Restricted Three-Body Problem, and thus can be employed in any area of the NRHOs.

To regulate chaser dynamics, a control action, such as $\mathbf{u}/m$, should be incorporated into Equation 1.17. To account for mass variation along the trajectory, the following equation should be included as well:

$$\dot{m} = \frac{\|\mathbf{u}\|}{I_{sp}g_0} \tag{1.21}$$

The specific impulse of the chaser thrusters is denoted by I_{sp}, while the constant g_0 is equal to 9.81 m/s^2.

1.3. Reinforcement Learning

The concept of learning through interaction with our surroundings is probably the first that comes to mind when we consider the essence of learning. Babies playing, waving their arms, or observing their environment do not have a formal instructor, but they do have a direct physical connection to their environment. Exploiting this connection provides a great deal of knowledge about cause and effect, the results of actions, and how to reach objectives. Learning from interaction is a fundamental concept that is at the core of almost all theories of learning and intelligence. *Reinforcement Learning* [33] is learning what to do, how to map situations to actions, to maximize a numerical reward signal. The learner is not told which actions to take; instead, the learner must discover which actions yield the most reward by trying them. It is formalized using ideas from dynamical systems theory, specifically a discrete-time stochastic control process known *Markov Decision Processes (MDP)*. A learning agent must be able to sense the state of its environment to some extent and must be able to take actions that affect the state. The agent also must have a goal or goals related to the state of the environment.

Reinforcement learning is different from *Supervised Learning*, the kind of learning from a training set of labeled examples provided by a knowledgeable external supervisor. This is an important kind of learning, but alone it is not adequate for learning from interaction. Reinforcement learning is also different from *Unsupervised Learning*, which consists of generally finding the hidden structure in collections of unlabeled data. Therefore, RL is considered to be a third *Machine Learning (ML)* paradigm.

One of the challenges that arises in reinforcement learning is the trade-off between exploration and exploitation. The agent has to *exploit* what it has already experienced in order to obtain reward, but it also has to *explore* in order to make better action selections in the future. The dilemma is that neither exploration nor exploitation can be pursued exclusively without failing at the task.

In addition to the agent, there are four main components of a reinforcement learning system: a policy, a reward signal, a value function, and a model of the environment. These are discussed in Section 1.3.2.

1.3.1. Reinforcement Learning and Optimal Control

The study of optimal control has been a major thread in the history of early reinforcement learning. This field of research involves the problem of designing a controller to optimize a measure of the behavior of a dynamical system over time. Dynamic programming,

introduced by Bellman in 1954 [72], is a class of methods for solving optimal control problems. Furthermore, Bellman also developed the discrete stochastic version of the optimal control problem, known as *Markov Decision Processes (MDPs)*. It is now evident that reinforcement learning problems are closely related to optimal control problems, particularly those formulated as MDPs.

In the realm of spacecraft guidance and control, in order to make it autonomous, the chosen algorithm must be both robust and computationally efficient. In the context of *Optimal Control Problems (OCP)*, the typical solution choices are open-loop solutions, such as indirect methods (e.g., Euler-Lagrange equations, Pontryagin Maximum Principle) and direct ones (e.g., single or multiple shooting, transcription, and collocation). However, these solutions cannot generate additional corrections with respect to the nominal reference trajectory, making them unreliable, not flexible, and not robust. An alternative that is more secure is to refer to the family of closed-loop solutions such as the *Hamilton-Jacobi-Bellman (HJB)* equations and *Dynamic Programming (DP)* theory. These are partial differential equations, which are not computationally efficient. Artificial Neural Networks (ANNs) can be used as an approximation of these complex solutions [33] and implemented as a closed-loop controller. Neural networks can be trained with Reinforcement Learning to map observations $\mathbf{y}$ into optimal control actions $\mathbf{u}$, as illustrated in Figure 1.9 with a stochastic policy $\boldsymbol{\pi_\theta}$ such that $\mathbf{u} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$.

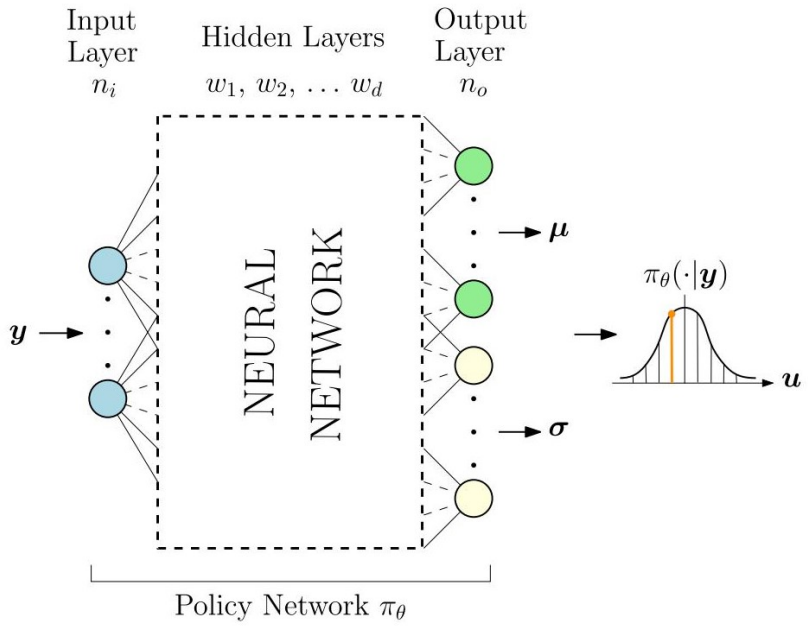


Figure 1.9: Artificial Neural Networks (ANNs) trained with Reinforcement Learning (RL) algorithms can map observations $\mathbf{y}$ into optimal control actions $\mathbf{u}$. Source: Reference [40].

Reinforcement learning does not require an expert control engineer for training, making it a great choice for complicated dynamic situations. Instead of designing a closed-loop controller, it is learned; the engineering effort is focused on reward engineering and adjusting hyperparameters after the environment has been established. Once the policy is trained, it can be deployed in a computationally efficient manner. To emphasize the close connection between RL and closed-loop OCP, Figure 1.10 and Table 1.4 are presented.

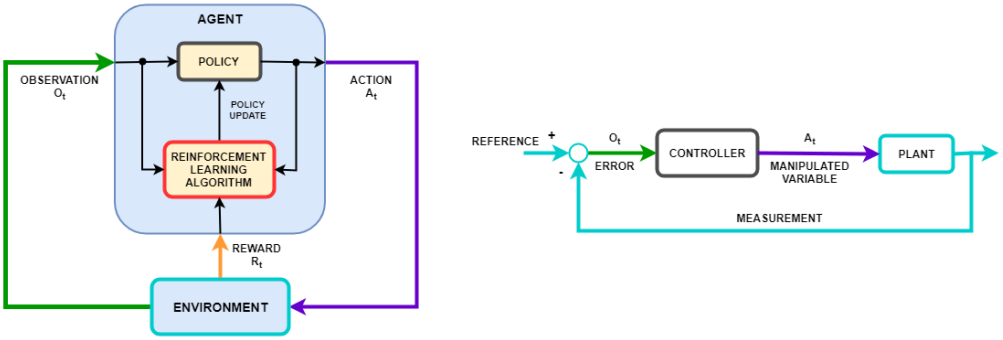


Figure 1.10: Reinforcement Learning (RL) architecture compared to a traditional closed-loop dynamical system. Source: `https://tinyurl.com/2p9b9wux`.

Reinforcement Learning	Closed-Loop Optimal Controller
Policy	Controller
Environment	Plant
Observation	Measurement
Action	Control Signal
Reward	Cost Function
Learning Algorithm	Adaptive Mechanism

Table 1.4: Reinforcement Learning (RL) translated into a representation of a closed-loop optimal controlled dynamical system. Source: `https://tinyurl.com/2p9b9wux`.

1.3.2. Markov Decision Process

Markov Decision Processes (MDPs) are a well-known way to represent sequential decision making, where actions taken affect not only immediate rewards, but also subsequent states and future rewards. This means that MDPs must balance immediate and delayed rewards, making them a mathematical representation of the Reinforcement Learning (RL) problem, which is the challenge of learning from interaction to achieve a goal. The learner

and decision maker is called *agent*, while the thing with which it interacts is called *environment*. At each successive discrete time step ($t = 0, 1, 2, 3, ...$) the agent responds to the environment by receiving a representation of the state, $S_t \in \mathcal{S}$, and then selecting an action, $A_t \in \mathcal{A}(s)$. As a result of its action, the agent is given a numerical reward $R_{t+1} \in \mathcal{R}$. Figure 1.11 shows the interaction between them, leading to a *trajectory* beginning with: S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, etc.

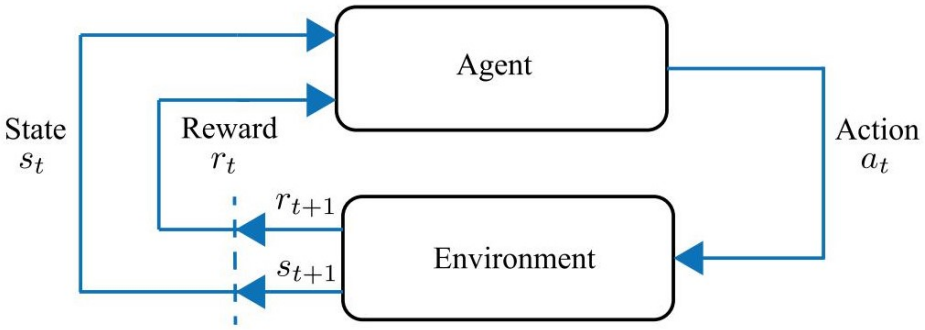


Figure 1.11: The agent-environment interaction at each discrete time step in a Markov Decision Process (MDP). Source: Reference [33].

To utilize this modeling framework, the underlying hypotheses are:

1. State fully observable and known[3];
2. Sets of states $\mathcal{S}$, actions $\mathcal{A}$ and rewards $\mathcal{R}$ all have a finite number of elements;
3. The likelihood of each potential value for S_t and R_t is only contingent on the immediately preceding state and action S_{t-1} and R_{t-1}, and not in any way on prior states and actions. This is called the *Markov property*.

The MDP environment is described by a discrete probability distribution $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0,\ 1]$, which determines the dynamics of the system as:

$$p\left(s',\ r \mid s,\ a\right) \doteq Pr\left(S_t = s',\ R_t = r \mid S_{t-1} = s,\ A_{t-1} = a\right) \tag{1.22}$$

For all $s' \in \mathcal{S}$, $s \in \mathcal{S}$, $r \in \mathcal{R}$ and $a \in \mathcal{A}(s)$. Generally, actions can be any decisions we wish to learn how to make, and the state can be anything we can be aware of that could be beneficial in making those decisions. The expected rewards for state-action pairs can be calculated using a two-argument function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ such as:

[3]The spacecraft is assumed to be equipped with all the sensors and algorithms required to accurately determine its position and speed.

$$r\left(s,\ a\right) \doteq \mathbb{E}\left[R_t \mid S_{t-1} = s,\ A_{t-1} = a\right] = \sum_r r \sum_{s'} p\left(s',\ r \mid s,\ a\right) \tag{1.23}$$

Episodes and Tasks

The aim of the agent is to discover how to maximize the total reward it obtains, thus its attention is on the cumulative reward in the long term and not the immediate reward $R_t \in \mathbb{R}$. Typically, the interaction between an agent and its environment is divided into sequences known as *episodes*. At the end of each episode, a particular condition is reached known as the *terminal* state, which indicates whether the task has been completed or not. After this, the system is reset to a standard initial state or a sample from a standard distribution of initial states. Depending on whether the termination time T is finite or infinite, episodes can be referred to as *episodic* or *continuous* tasks. To prevent high cumulative rewards and facilitate convergence, the agent applies the concept of discounting to select actions that maximize the sum of discounted benefits it will receive in the future; in particular, it chooses action A_t to maximize the so-called *discounted return* denoted as G_t:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = R_{t+1} + \gamma G_{t+1} = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \tag{1.24}$$

Where $\gamma \in [0,\ 1]$ is the *discount rate*. The discount rate determines the current value of a future reward: a reward received k time steps from now is worth only a fraction of what it would be worth if it were received immediately. This specific discounted reward formulation combines episodic and continuous tasks taking into account that the end of an episode, if it exists, enters a special state known as an *absorbing* state that only transitions to itself and does not yield any rewards. It allows for $T = \inf$ or $\gamma = 1$.

Policy and Value Functions

A *policy* π can be formally expressed as a mapping of states to the likelihood of selecting each potential action; specifically, $\pi\left(a|s\right)$ is the probability that $A_t = a$ when $S_t = s$. The expected discounted return when starting in a state $s \in \mathcal{S}$ and following policy π is known as the *state-value function* v_π. This is defined as follows:

$$v_\pi\left(s\right) \doteq \mathbb{E}_\pi\left[G_t \mid S_t = s\right] \tag{1.25}$$

The expected value of a random variable is denoted by $\mathbb{E}_\pi[\cdot]$ when the agent follows the policy π. The expected discounted return when beginning in a state $s \in \mathcal{S}$ and taking an action $a \in \mathcal{A}(s)$ according to the policy π is denoted by the *action-value function* q_π:

$$q_\pi(s,\ a) \doteq \mathbb{E}_\pi\left[G_t \mid S_t = s,\ A_t = a\right] \tag{1.26}$$

The agent is unaware of these two relations beforehand, but they can be estimated while learning.

The state-value function has a key feature in which the value of a state $s \in \mathcal{S}$ under a policy π is consistent with the value of its possible successor state $s' \in \mathcal{S}$, as expressed by the *Bellman Equation*:

$$v_\pi(s) = \sum_a \pi\left(a \mid s\right) \sum_{s',\ r} p\left(s',\ r \mid s,\ a\right)\left[r + \gamma v_\pi\left(s'\right)\right] \tag{1.27}$$

Finding a policy that maximizes the expected discounted return is the goal of solving a reinforcement learning problem. A policy π is considered to be better than or equal to another policy π' if $v_\pi(s) \geq v_{\pi'}(s)$ for all states $s \in \mathcal{S}$. The most advantageous policy is known as an *optimal policy* and is represented by π_*. Its optimal state-value function v_* is defined as:

$$v_*(s) = \max_\pi\ v_\pi(s) \tag{1.28}$$

The optimal state-value function v_* must satisfy the self-consistency condition expressed in Equation 1.27, resulting in the *Bellman Optimality Equation* 1.29. It can be said that the state-value function when following an optimal policy must be equal to the expected discounted return of the most advantageous action from that state:

$$v_*(s) = \max_a\ q_{\pi_*}(s,\ a) = \max_a \sum_{s',\ r} p\left(s',\ r \mid s,\ a\right)\left[r + \gamma v_*\left(s'\right)\right] \tag{1.29}$$

The Bellman optimality equation for v_π has a single solution that is not dependent on the policy. However, there can be multiple optimal policies. In theory, Equation 1.29 can be solved by knowing the dynamics of the environment, as a system of equations in v_*. Once one of the optimal actions is identified for a given state, the optimal policy is the one that assigns a non-zero probability to that action alone. The benefit of v_* is that it allows us

to quickly and easily identify the best course of action for any given state by looking at the immediate rewards it offers. This means that the expected long-term optimal reward can be determined by evaluating the immediate effects of each action.

1.3.3. Approximated Solution Methods

In order to tackle reinforcement learning problems with a wide state space, tabular methods are not suitable. Instead, function approximation techniques must be used. This set of algorithms is designed to approximate functions by training a weighted parameterized functional form, such as an *Artificial Neural Network (ANN)*. In this class of techniques, *Policy Gradient Methods* are preferred due to their greater stability and assurance of convergence, as evidenced by Tipaldi et al. [73]. Moreover, they calculate directly the policy without requiring a model, although they are less data efficient compared to traditional value-based methods such as Q-learning. This issue can be addressed by using an actor-critic architecture, which reduces the estimated policy-gradient variance. Policy gradient methods are attractive due to their ability to handle high-dimensional, continuous-state, and continuous-action spaces, especially when combined with nonlinear function approximators such as ANNs. The selected algorithm, *Proximal Policy Optimization (PPO)*, is currently considered the leading continuous control reinforcement learning algorithm [74].

Policy Gradient Methods

Methods that learn a parameterized policy and select actions without computing a value function are discussed in this subsection. The parameter vector is expressed as $\boldsymbol{\theta} \in \mathbb{R}^d$ and the policy as $\pi_{\boldsymbol{\theta}}\left(a|s\right)$. The algorithms to solve the policy parameters are based on the gradient of a scalar performance measure $J\left(\boldsymbol{\theta}\right)$. They are designed to maximize performance, so the update of the parameters at the training step k follows the gradient ascent of J:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha \widehat{\boldsymbol{\nabla} J\left(\boldsymbol{\theta}_k\right)} \tag{1.30}$$

Where $\widehat{\boldsymbol{\nabla} J\left(\boldsymbol{\theta}_k\right)}$ is an estimate of the gradient of the performance metric with respect to the current policy parameters $\boldsymbol{\theta}_k$, and the step size is determined by the *learning rate* α. In RL, the performance metric is usually expressed as in Equation 1.31, which is the state-value function with respect to the policy determined by the current parameters.

$$J\left(\boldsymbol{\theta}\right) = v_{\pi_{\boldsymbol{\theta}}}\left(S_0 = s\right) \tag{1.31}$$

What is the technique for calculating the performance gradient in relation to the policy parameters when the effect of policy modifications on the state distribution is unknown? The *Policy Gradient Theorem* is a great theoretical response to this issue; it gives a mathematical expression of the performance gradient in terms of the policy parameters. It establishes that:

$$\nabla J\left(\boldsymbol{\theta}\right) \propto \sum_{s} \mu\left(s\right) \sum_{a} q_{\pi}\left(s,\ a\right) \boldsymbol{\nabla} \pi_{\boldsymbol{\theta}}\left(a \mid s\right) \tag{1.32}$$

Gradients are column vectors of partial derivatives with respect to the components of $\boldsymbol{\theta}$, and $\mu\left(s\right)$ is the distribution of states following policy π.

The policy $\pi_{\boldsymbol{\theta}}\left(a \mid s\right)$ is usually chosen to be stochastic in order to allow exploration. For continuous action spaces, the policy can be expressed as a normal probability density function, with the mean $\mu : \mathcal{S} \times \mathbb{R}^{d} \rightarrow \mathbb{R}$ and standard deviation $\sigma : \mathcal{S} \times \mathbb{R}^{d} \rightarrow \mathbb{R}^{+}$ determined by a function approximator that depends on the state. In other words:

$$\pi_{\boldsymbol{\theta}}\left(a \mid s\right) \doteq \frac{1}{\sigma\left(s,\ \boldsymbol{\theta}\right)\sqrt{2\pi}} \exp\left(-\frac{\left(a - \mu\left(s,\ \boldsymbol{\theta}\right)\right)^{2}}{2\sigma\left(s,\ \boldsymbol{\theta}\right)^{2}}\right) \tag{1.33}$$

As the learning process progresses, exploration of the environment becomes less important in favor of exploitation (i.e., a decrease in *sigma*). When the final policy is deployed, the exploration is terminated, and the optimal deterministic action for a given observation is returned.

REINFORCE with Baseline

The first policy-gradient learning algorithm is now ready to be derived. The policy gradient theorem (Equation 1.32) provides an estimate of the gradient that can be acquired by taking samples from the environment. This expression, due to $\mu\left(s\right)$, is proportional to the sum of states experienced under the target policy π, weighted by the frequency of their occurrence. Therefore, it can be interpreted as follows:

$$\nabla J\left(\boldsymbol{\theta}\right) \propto \mathbb{E}_{\pi}\left[\sum_{a} q_{\pi}\left(S_{t},\ a\right) \nabla \pi_{\boldsymbol{\theta}}\left(a \mid S_{t}\right)\right] \tag{1.34}$$

Multiplying and dividing by the policy and substituting a with A_{t} sampled from $\pi_{\boldsymbol{\theta}}$, it can be expressed as:

$$\nabla J(\boldsymbol{\theta}) \propto \mathbb{E}_{\pi}\left[G_t \frac{\nabla \pi_{\boldsymbol{\theta}}\left(A_t \mid S_t\right)}{\pi_{\boldsymbol{\theta}}\left(A_t \mid S_t\right)}\right] \tag{1.35}$$

The step size α can absorb any constant of proportionality, making it easy to use the gradient estimate in Equation 1.36 to obtain the update of the stochastic gradient ascent in Equation 1.30, when only one sample trajectory is taken into account.

$$\widehat{\nabla J(\boldsymbol{\theta})} = G_t \frac{\nabla \pi_{\boldsymbol{\theta}}\left(A_t \mid S_t\right)}{\pi_{\boldsymbol{\theta}}\left(A_t \mid S_t\right)} \tag{1.36}$$

At each time step, a sample is taken of the latter quantity and an update is made. Unfortunately, this approach has a high variance and a slow rate of learning. To enhance it, it can be broadened to include the comparison of the action-value function with an arbitrary *baseline* function $b(s)$:

$$\widehat{\nabla J(\boldsymbol{\theta})} = \left(G_t - b(s)\right) \frac{\nabla \pi_{\boldsymbol{\theta}}\left(A_t \mid S_t\right)}{\pi_{\boldsymbol{\theta}}\left(A_t \mid S_t\right)} \tag{1.37}$$

The baseline can be any function, even a random variable, as long as it does not depend on the action taken. It has been shown that the baseline does not introduce bias. Consequently, it is not a *bootstrapping technique*, which is a method by which an agent updates its value estimates or policies using predictions from its current estimates, combining prior knowledge with newly acquired data to improve decision making.

Actor-Critic Methods

This family of techniques attempts to generate an estimate with a lower variance, which is more suitable for online applications. These methods have the advantages of policy gradient methods, but with a sample efficiency that is more similar to traditional Q-Learning [73]. They introduce a bias through a bootstrapping critic, which is similar to the baseline defined in Equation 1.37, with an estimate of the state-value function $\hat{v}_{\mathbf{w}}$ where $\mathbf{w} \in \mathbb{R}^m$ is a learned weight vector[4]. As a result, these methods estimate both the optimal policy through a function approximator known as *actor*, and the state-value function through another called *critic*, which are usually made up of ANNs.

[4]The state-value function is still contingent on the policy π. In actor-critic methods, a change in notation has been made, as it is typically a function of certain parameters $\mathbf{w} \in \mathbb{R}^m$ that are then implicitly associated with the policy.

The actor-critic methods use a policy gradient estimate within the stochastic gradient descent algorithm as follows:

$$\widehat{\nabla J(\boldsymbol{\theta})} = A_t(S_t,\ A_t)\,\frac{\nabla \pi_{\boldsymbol{\theta}}\left(A_t \mid S_t\right)}{\pi_{\boldsymbol{\theta}}\left(A_t \mid S_t\right)} \tag{1.38}$$

$$A_t(S_t,\ A_t) = q_\pi(S_t,\ A_t) - v_{\mathbf{w}}(S_t) \tag{1.39}$$

The *Advantage Function*, $A_t(s,\ a)$, is an indicator of the additional reward that the agent could receive by taking a certain action from a given state. This method is an online incremental learning process, where states, actions, and rewards are processed as they occur, and not reexamined. In many cases, a *surrogate* objective function is used to take advantage of automatic differentiation software [56], with its gradient being the policy gradient estimator, as follows:

$$L(\boldsymbol{\theta}) = \hat{\mathbb{E}}_\pi\left[\log \pi_{\boldsymbol{\theta}}\left(A_t \mid S_t\right)\hat{A}_t\right] \tag{1.40}$$

Therefore, a practical algorithm requires the estimation of both the advantage function $\hat{A}_t$ (which is contingent on the unknown environment) and the expectation operator $\hat{\mathbb{E}}_\pi$, which are typically performed with data from trajectory roll-outs.

Trust Region Policy Optimization

The *Trust Region Policy Optimization* algorithm is the first actor-critic approach to be considered. It maximizes a surrogate objective function while constraining the size of the policy update. It is guaranteed to monotonically enhance the policy function, as stated in Schulman et al. [74], and is especially effective with large non-linear policies such as ANNs. Specifically:

$$\begin{aligned} \max_{\boldsymbol{\theta}} \quad & \hat{\mathbb{E}}_\pi\left[\frac{\pi_\theta\left(A_t \mid S_t\right)}{\pi_{\theta,old}\left(A_t \mid S_t\right)}\hat{A}_t\right] \\ \text{s.t.} \quad & \hat{\mathbb{E}}_\pi\left[KL\left(\pi_{\theta,old}\left(\cdot \mid S_t\right),\ \pi_\theta\left(\cdot \mid S_t\right)\right)\right] \leq \delta \end{aligned} \tag{1.41}$$

The policy prior to parameter update is denoted by $\pi_{\theta,old}$. The unique feature of this approach, apart from the surrogate objective function, is the restriction of the *Kullback-Leibler (KL)* divergence[5] at each point in the state space. This helps to avoid changes to

[5]The Kullback–Leibler divergence (also called relative entropy and I-divergence), denoted $KL(P,\ Q)$, is a type of statistical distance: a measure of how one probability distribution P is different from a second

the policy that are too drastic, thus guaranteeing robustness and a strong convergence. This optimization issue can be effectively tackled by utilizing the conjugate gradient algorithm, after making a linear estimation of the objective and a quadratic estimation of the constraint. TRPO is a reliable and data-efficient algorithm, especially when it comes to continuous control tasks [74]. Despite its effectiveness, it is quite complex and difficult to implement.

Proximal Policy Optimization (PPO)

The *Proximal Policy Optimization (PPO)* algorithm is an actor-critic method that follows the same principles as TRPO. It is simpler to implement and usually has better empirical sample complexity, even though it is a first-order approximation. PPO employs a *clipped surrogate* objective to generate a pessimistic and conservative estimate of the policy's performance without the need for any explicit constraint. Instead of TRPO hard constraints, PPO uses soft constraints that exploit a clipping parameter ε. The resulting unconstrained optimization problem is defined as follows:

$$\max_{\boldsymbol{\theta}} \quad \hat{\mathbb{E}}_{\pi}\left[\min\left(r_t\left(\boldsymbol{\theta}\right)\hat{A}_t,\ clip\left(r_t\left(\boldsymbol{\theta}\right),\ 1-\varepsilon,\ 1+\varepsilon\right)\hat{A}_t\right)\right] \tag{1.42}$$

The probability ratio r_t is expressed as:

$$r_t\left(\boldsymbol{\theta}\right) = \frac{\pi_{\theta}\left(A_t \mid S_t\right)}{\pi_{\theta,old}\left(A_t \mid S_t\right)} \tag{1.43}$$

The first term within *min* in Equation 1.42 is the objective function of TRPO. The second term modifies the surrogate objective by clipping the probability ratio, thus preventing r_t from being outside the range $[1-\varepsilon,\ 1+\varepsilon]$. Finally, by taking the minimum of the clipped and unclipped objectives, the final goal is a lower pessimistic limit for the unclipped objective. This scheme disregards a significant alteration in the probability ratio if it would lead to an improvement in the objective, but if it would cause the objective to worsen, it is taken into account. Figure 1.12 shows the optimization of the surrogate function in a single time step in relation to the probability ratio of positive and negative advantages (left and right, respectively). The red circle on each graph indicates the starting point of optimization (that is, $r_t = 1$).

An example of a PPO implementation with fixed length trajectory segments, as reported in Schulman et al. [56], is presented in Algorithm 1.1. This approach runs the policy

reference probability distribution Q.

in the environment for T time steps and uses all collected samples to update the policy. In each iteration, each of the N parallel actors collects T time steps of data. Then, the surrogate loss and the advantage estimates are constructed on these NT timesteps of the data and gradient descent is applied (typically with Adam optimizer, shown in Section 1.5) for K epochs.

This approach requires an advantage estimator; since it is an actor-critic technique, the state-value function is learned in conjunction with the policy and then a *Generalized Advantage Estimate (GAE)* [75] is typically employed. A truncated GAE version is usually implemented with *Recurrent Neural Networks (RNNs)* agents, as proposed by Mnih et al. [76]. Both versions are discussed in Section 1.5. As shown in Reference [56], PPO takes only a few lines of code and outperforms TRPO in most continuous control environments.

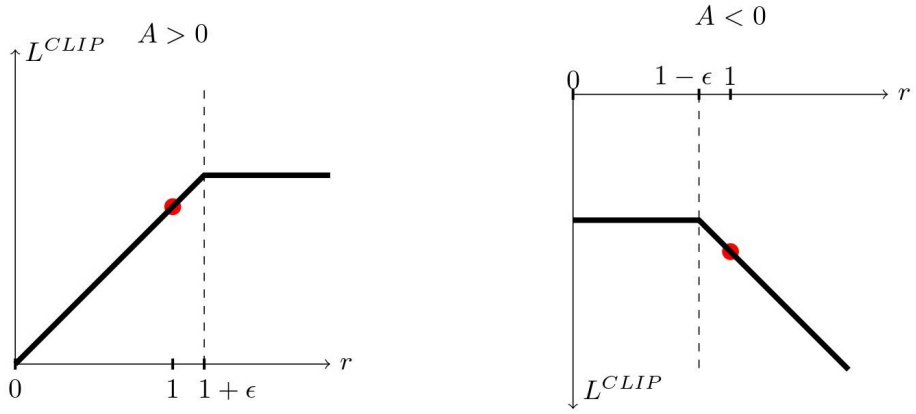


Figure 1.12: Proximal Policy Optimization (PPO) clipped surrogate objective function behavior during optimization with respect to different probability ratios and advantage function signs. Source: Reference [56].

Algorithm 1.1 PPO, Actor-Critic Style. Source: Reference [56].

for iteration=1,2,... **do**
  for actor=1,2,...,N **do**
    Run Policy $\pi_{\theta old}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1$,...,$\hat{A}_T$
  end for
  Optimize surrogate L wrt $\boldsymbol{\theta}$, with K epochs and minibatches size $M \leq NT$
  $\theta_{old} \leftarrow \boldsymbol{\theta}$
end for

In certain scenarios, the objective of Equation 1.42 can be augmented by including an entropy bonus to guarantee adequate exploration, as proposed by Mnih et al. [76]. This

helps to maintain a balance between exploration and exploitation.The entropy of the policy π for a given state s is denoted as $H_\pi(s)$ and is expressed as:

$$H_\pi(s) = -\sum_a \pi(a \mid s) \log \pi(a \mid s) \tag{1.44}$$

The objective function of the PPO version with entropy has this bonus term, $H_\pi(s)$, subtracted from it to stimulate exploration. The magnitude of this incentive is regulated by an entropy coefficient c.

1.4. Artificial Neural Networks

Artificial Neural Networks (ANNs) are commonly used to approximate nonlinear functions. These networks are composed of interconnected units that possess some of the characteristics of human neurons, the primary components of the nervous system. Deep neural networks can be classified into three primary types: Feedforward, Convolutional, and Recurrent networks.

Figure 1.13 illustrates a generic Feedforward ANN, which has an output layer made of two units, an input layer made of four units, and two hidden layers (no input nor output layers). Each link is associated with a weight of real value, which is analogous to the efficacy of a synaptic connection in a real neural network.

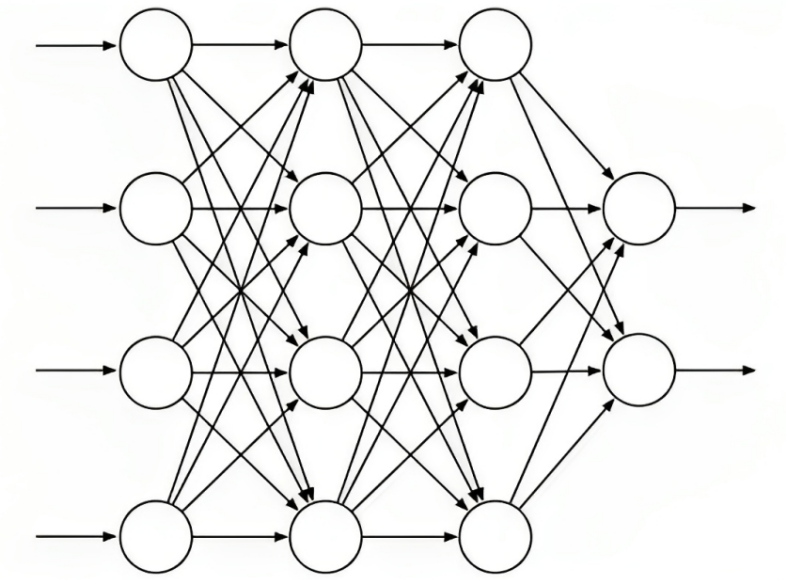


Figure 1.13: A Feedforward Artificial Neural Network (Feedforward ANN) with four input units, two output units, and two hidden layers. Source: Reference [33].

ANNs are designed to approximate a certain function f by assigning an output value to an input value through a mapping $y = N(x,\ \mathbf{w})$. This is done by learning the values of the parameters $\mathbf{w} \in \mathcal{R}^w$, which are referred to as weights, to achieve the most accurate approximation. Each successive unit of hidden layers computes representations of the raw input that become increasingly abstract, providing features that contribute to the hierarchical representation of the entire input-output function of the network, rather than being crafted by hand. ANNs usually learn using a stochastic gradient method. In the most common supervised learning case, the objective function is the mean square error on a set of labeled training examples. In reinforcement learning, ANNs aim to maximize expected reward, as is done in a policy gradient algorithm, by taking advantage of data coming from interactions with the environment. To achieve this, one must evaluate the influence that a change in the weight of each connection has on the general performance of the network; this can be done using a *Backpropagation (BP)* approach to acquire the network gradient.

Before delving into the details of a neuron's structure and the backpropagation process, it is essential to understand why Artificial Neural Networks (ANNs) can accurately approximate any non-linear and complex function. This is achieved by using activation functions, as demonstrated by Theorem 1.1.

Theorem 1.1 (Universal Approximation Theorem)**.** *The universal approximation theorem takes the following classical form. Let $\phi : \mathbb{R} \to \mathbb{R}$ be a non-constant, bounded, continuous function (i.e., the activation function). Let I_m denote the hypercube unit m dimensional $[0,\ 1]$. The space of continuous functions with real value in I_m is denoted by $C(I_m)$. Then, given any $\varepsilon > 0$ and any function $f \in C(I_m)$, there exist an integer N, real constants $v_i,\ b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}$ for $i = 1, ..., N$ such that we may define:*

$$F(\mathbf{x}) = \sum_{i=1}^{N} v_i \phi\left(\mathbf{w}_i^T \mathbf{x} + \mathbf{b}\right) \tag{1.45}$$

As an approximate realization of the function f

$$|F(\mathbf{x}) - f(\mathbf{x})| < \varepsilon \qquad \forall \mathbf{x} \in I_m \tag{1.46}$$

A review of the most popular activation functions is presented in Table 1.5. In the past [77], most neural networks employed the *Sigmoid* activation function, however, its saturation in most of the domain has now discouraged its use (i.e., very difficult to learn

using gradient-based methods). Despite this, RNNs have additional requirements that make sigmoidal units more attractive despite the saturation issue.

Activation Function	ϕ	ϕ'	Codomain
Hyperbolic Tangent	$\tanh x$	$1 - \tanh x$	$(-1,\ 1)$
Sigmoid	$1/\left(1 + e^{-x}\right)$	$\phi\left(1 - \phi\right)$	$(0,\ 1)$
ReLu	$max\left(0,\ x\right)$	$max\left(0,\ x\right)$	$[0,\ \infty)$
Heavyside Step	$\left(sgn\left(x\right) + 1\right)/2$	δ_0	$[0,\ 1]$
Softmax	$e^{x_i}/\sum_j^n e^{x_i}$	$\phi_i\left(\delta_{ij} - \phi_j\right)$	$(0,\ 1)$

Table 1.5: Overview of commonly employed activation functions in Artificial Neural Networks (ANNs).

Multi-Layer Perceptron

The most popular deep model is the *Feedforward ANN*, also known as a *Multi-Layer Perceptron (MLP)*. This type of network has an information flow that goes from the input layer, through the hidden layers, and finally to the output without any feedback. This creates an *acyclic graph*. When designing such a neural network, it is important to consider two important design parameters:

1. Depth: neural networks are usually composed of multiple layers, which are commonly referred to as input, hidden, and output layers, and are evaluated in succession;

2. Width: each layer is typically a vector-valued function, and its size is indicated by the amount of neurons it contains.

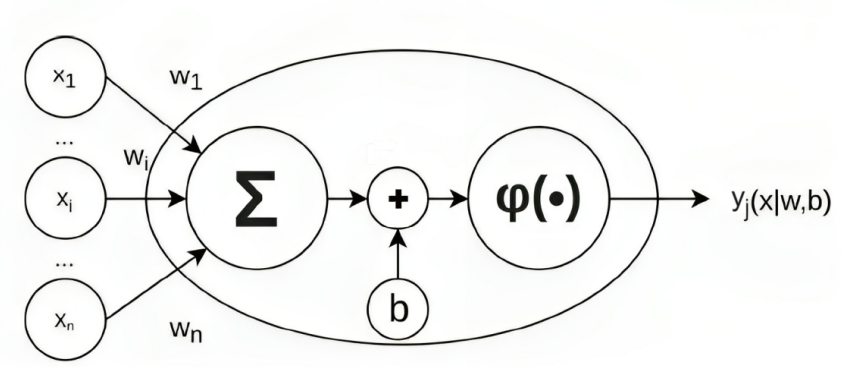


Figure 1.14: Elementary architecture of a Multi-Layer Perceptron (MLP). Source: Reference [78].

The elementary unit of an MLP is the neuron. With reference to Figure 1.14, the so-called induced local field v_j of the neuron j, which is the input of the activation function $\varphi_j(\cdot)$, can be expressed as:

$$v_j = \sum_{i \in C_i} w_{ij} x_j + b_j \tag{1.47}$$

The neurons in the set C_i are linked to layer j, and b_j is the bias term. The output y_j of a neuron is determined by the activation function (which performs the nonlinear transformation) applied to the local field v_j:

$$y_j = \varphi_j\left(v_j\right) \tag{1.48}$$

Backpropagation Algorithm

Most of the training algorithms for artificial neural networks are based on *Backpropagation (BP)*. Generally, finding the weights and biases of an ANN involves identifying the optimal set of variables that minimize a given loss function:

$$(\mathbf{w}_*,\ \mathbf{b}_*) = \operatorname{argmin} J\left(\mathbf{w},\ \mathbf{b}\right) \tag{1.49}$$

It is not possible to obtain a closed form solution, so it is common to use iterative methods that make use of the derivative of the objective function to reach the optimal value. The backpropagation algorithm is a smart way to calculate these derivatives, which can then be used with certain optimization algorithms. Thus, backpropagation only refers to the technique for computing the gradient, while another algorithm, like stochastic gradient descent, is used to do the actual learning with the gradient.

In gradient-based approaches for a traditional MLP, the weights of a neuron, w_{ij}, are adjusted according to the direction indicated by the partial derivatives. This direction can be determined using the chain rule as:

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial v_j} \frac{\partial v_j}{\partial w_{ij}} \tag{1.50}$$

The weights of the neurons are adjusted by Δw_{ij} through a gradient descent step in the weight space, which is based on the gradient made up of the partial derivatives of Equation 1.50. The backpropagation algorithm is called this way because it involves two

passes through the network: a forward pass and a backward pass. During the forward pass, the output of the network and the activation function of each neuron are evaluated without altering the weights. The backward pass begins at the output layer, sending the loss function back to the input layer, and calculating the local gradient for each neuron.

Artificial Neural Networks (ANNs) are susceptible to two main issues during training. The first is *overfitting*, which is the inability to accurately generalize to cases where the network has not been trained. This is a common problem for ANNs, but especially so for deep ANNs due to their large number of weights. To reduce overfitting, various techniques have been developed, such as cross-validation, regularization, and weight sharing.The second issue is that backpropagation may not be effective due to the partial derivatives computed by its backward passes. These derivatives can either decay rapidly towards the input side of the network, making learning by deep layers very slow, or they can grow rapidly towards the input side of the network, making learning unstable. These occurrences are referred to as *vanishing and exploding gradients*, respectively.

1.4.1. Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are ANN architectures that contain at least one feedback loop in their layer interactions. These networks are capable of efficiently processing time series data and sequential data in general. The connections between neurons create a *directed graph*, enabling the network to store memory and display temporal dynamic behaviors through a time-dependent internal state. A *many-to-many RNN* (a type of neural network that takes a sequence of input data and produces a sequence of output data) is typically expressed at time step t with hidden state h_t and output y_t as:

$$\begin{aligned} h_t &= \varphi(w_{hh}h_{t-1} + w_{hx}x_t + b_h) \\ y_t &= w_{yh}h_t + b_y \end{aligned} \tag{1.51}$$

This architecture is illustrated in Figure 1.15 as a computational graph, a way to express the structure of a set of computations (such as those used to map inputs and parameters to outputs). The process of *unfolding* a RNN involves transforming it into a computational graph that illustrates how the parameters are shared across the deep network, particularly the recurrence of hidden states. The concept of unfolding is the basis of the *Backpropagation Through Time (BPTT)* method, which gives RNN the ability to calculate gradients. Unfolding provides a clear understanding of the calculations that must be performed. This concept is necessary because, as shown in Equation 1.51, it is evident that h_t depends on h_{t-1}.

The BPTT algorithm is a straightforward application of backpropagation to the computational graph of an unfolded RNN. This makes it easy to calculate the gradient, which is done in reverse order from $t = T$ to $t = 1$. The gradient obtained can then be used with any gradient-based optimization technique. The fundamental equation of BPTT for the weight w_{hh} can be expressed as:

$$\frac{\partial J}{\partial w_{hh}} = \sum_{t=1}^{T} \frac{\partial J}{\partial h_t} \frac{\partial^+ h_t}{\partial w_{hh}} \tag{1.52}$$

In order to prevent vanishing/exploding gradients and improve memory efficiency, *Truncated Backpropagation Through Time (Truncated BPTT)* is often used when dealing with long time series data:

$$\frac{\partial J}{\partial w_{hh}} \simeq \sum_{t=T-n}^{T} \frac{\partial J}{\partial h_t} \frac{\partial^+ h_t}{\partial w_{hh}} \tag{1.53}$$

Taking into account only n time steps in the backward pass, the gradient is truncated. This allows the hidden state to be initialized at $t = T - n$ instead of the usual 0. The only downside is that the gradient is biased and the trained network has a reduced memory.

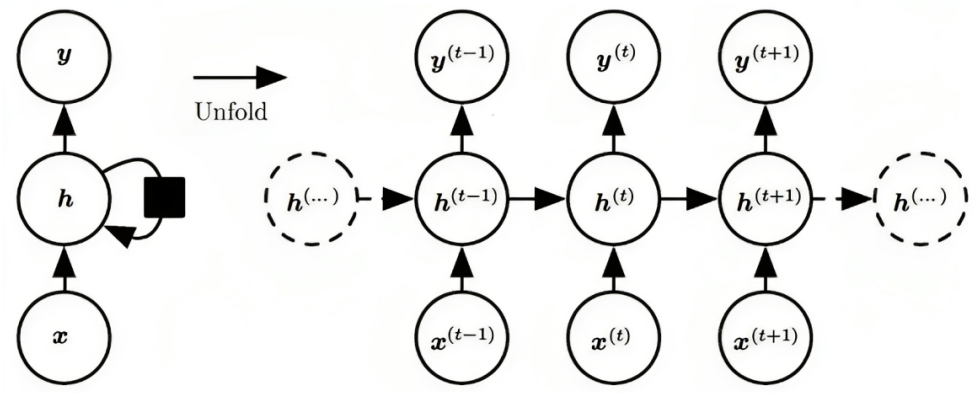


Figure 1.15: Many-to-Many Recurrent Neural Network (RNN) cell and its computational graph after unfolding process. Source: Reference [77].

Long Short-Term Memory

The mathematical challenge of learning long-term dependencies in *Recurrent Neural Networks (RNNs)* has a fundamental issue: the gradients propagated over many stages tend to either vanish (most of the time) or explode (rarely, but with much harm to optimization). To tackle this issue, *Gated Recurrent Neural Networks (Gated RNNs)* have been

created. The idea behind these cells is to create pathways through time that have derivatives that never vanish or become too large, which are accomplished through structures known as gates. Rather than simply accumulating information over a long period, it is advantageous for the neural network to be able to forget the old state and learn when to do so. The gates allow the network to selectively update and control the flow of information through it.

The *Long Short-Term Memory (LSTM)* network is a type of Gated RNN that is widely used to make predictions based on time series data. This architecture was first proposed by Hochreiter and Schmidhuber [79]. The main concept of an LSTM network is that a *cell state* allows data to pass through *three* gates, which are made up of a sigmoid activation layer and point-wise multiplication operations. The sigmoid layer of each gate produces a value between 0 and 1, which decides how much core information is processed. The LSTM cell state is closed-loop, meaning that there is an internal recurrence in addition to the external recurrence of a regular RNN hidden state (which is also present through the output state[6]).

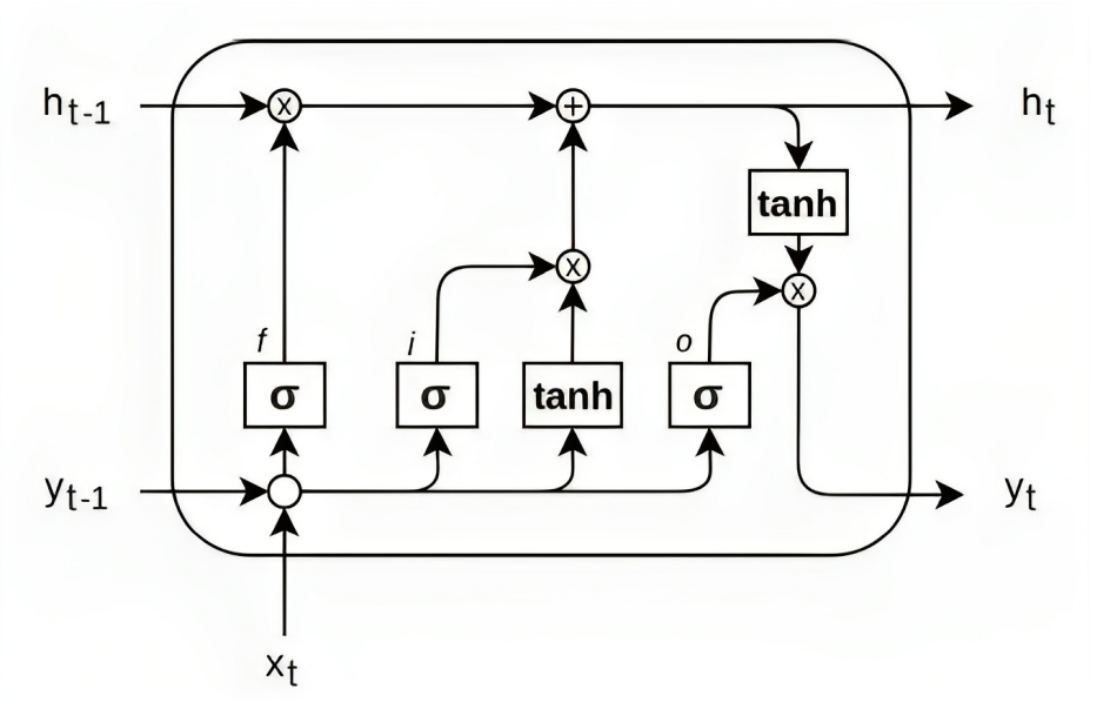


Figure 1.16: Internal architecture of a Long Short-Term Memory (LSTM). Source: Reference [78].

[6]In literature, the notation of Long Short-Term Memory (LSTM) is sometimes not uniform. Generally, the cell state is denoted by C_t and the hidden state, which is also the output in this case unlike RNNs, is symbolized by h_t. To maintain the definition of the vanilla RNN in Equation 1.51 and make concepts more understandable, a more "aerospace" notation [80] is used.

The key components of the LSTM network are summarized as follows:

- Cell state (h): the cell state conveys information through different time steps and is modified by linear interactions with the gates.
- Forget gate (f): the forget gate determines which information should be removed from the cell state.

$$f = \sigma\left(w_{fy}y_{t-1} + w_{fx}x_t + b_f\right) \tag{1.54}$$

- Input gate (i): the input gate is utilized to determine which new information should be included in the cell state. Subsequently, a hyperbolic tangent is used to generate a candidate cell state $\tilde{h}_t$ through point-wise multiplication.

$$i = \sigma\left(w_{iy}y_{t-1} + w_{ix}x_t + b_i\right) \tag{1.55}$$
$$\tilde{h}_t = i \cdot \tanh\left(w_{cy}y_{t-1} + w_{cx}x_t + b_c\right) \tag{1.56}$$

- Memory gate: the memory gate adds the old cell state from the forget gate to the candidate one. It is not presented as a standalone gate because it is a modification of the cell state itself without a sigmoid layer.

$$h_t = fh_{t-1} + \tilde{h}_t \tag{1.57}$$

- Output gate (o): the network output y_t is obtained by the output gate filtering the cell state.

$$o = \sigma\left(w_{oy}y_{t-1} + w_{ox}x_t + b_o\right) \tag{1.58}$$
$$y_t = o \cdot \tanh h_t \tag{1.59}$$

1.5. Meta-Reinforcement Learning

The development of GNC autonomous systems presents a major challenge in terms of robustness to uncertain spacecraft models and environments. Reinforcement learning deep-agents, such as Feedforward ANNs, are usually successful in learning within the training distributions but often have difficulty extrapolating beyond them (low generalization capability). This can lead to an unstable guidance law when the spacecraft encounters states that are not part of the training distribution. Moreover, traditional RL requires a large

amount of experience to even learn basic tasks (sample inefficiency). To address these two main weaknesses, recent advances in *Meta-Reinforcement Learning (Meta-RL)* have been made. This is a *Machine Learning (ML)* technique in which an agent is taught a range of tasks (that is, randomized environment parameters) instead of just one, allowing it to create a meta-policy that can quickly adjust to new and unseen tasks with minimal experience [49]. This enables the agent to generalize its knowledge and adapt its policies efficiently, making it more flexible and capable of fast learning in unfamiliar settings. This transfer learning ability has been referred to as *"learn to learn"* by Wang et al. [48]. There are numerous approaches to put these ideas into practice, such as Meta-Agnostic-Meta-Learning (MAML), Task-Agnostic Meta-Learning (TAML), REPTILE, Meta-SGD, and many more [81–83]. One of the most popular approaches, as proposed by Hochreiter et al. [47], is to employ Long Short-Term Memory (LSTM) networks within a reinforcement learning agent, referred to as *Meta-Recurrent Neural Networks (Meta-RNN)*. Alternative methods become impractical when dealing with large models. However, this implementation, which uses gradient descent on LSTMs, proves capable of managing a substantial number of free parameters. The approach excels in handling non-stationary time series prediction in a single training sequence and has demonstrated applicability to learning and autonomous systems.

The Meta-RNN approach has been studied in the aerospace field and has been shown to be more effective than traditional RL in managing uncertain environments, actuator failures, and Partially Observable MDPs (POMDPs) [35, 40, 50]. The hidden states of Long Short-Term Memory (LSTM) networks provide them with an internal dynamics that is continually updated by new observations along the trajectory. This contributes to better learning in high-uncertainty environments during the training phase and allows for an adaptive policy during the testing phase. The resulting G&C algorithm has a higher overall robustness due to these properties, which is especially important after deployment. Unlike MLPs, LSTMs can modify their hidden state and adjust in real time to capture data that have not been modeled, such as dynamical disturbances or hardware failure, during spacecraft operations. Therefore, Meta-Recurrent Neural Networks (Meta-RNN) is the Meta-Reinforcement Learning solution adopted in this work.

Training Algorithm

In Subsection 1.3.3, PPO was discussed. As it is a model-free algorithm, meaning it does not possess any knowledge of the environment, an estimation of the advantage function must be provided in order to put it into practice. As mentioned previously, $A_t(s,\ a)$ (Equation 1.37) is used in actor-critic methods to reduce the variance of the policy gradient

estimate, although it comes with some bias [75]. According to Sutton et al. [84], the *Residual Temporal-Difference (Residual TD)*[7] of the discounted value function is defined as:

$$\delta_t^v = R_t + \gamma v_{\boldsymbol{\pi}}\left(S_{t+1}\right) - v_{\boldsymbol{\pi}}\left(S_t\right) \tag{1.60}$$

This value can be thought of as an unbiased estimator of the advantage function, provided that the state-value function is estimated as $v_{\boldsymbol{w}}(s)$, as follows:

$$\hat{A}_t = \hat{\mathbb{E}}_{\boldsymbol{\pi}}\left[R_t + \gamma v_{\boldsymbol{w}}\left(S_{t+1}\right) - v_{\boldsymbol{w}}\left(S_t\right)\right] \tag{1.61}$$

This estimator has a high variance, which can cause instability during the learning process. To gain more control and stability, it may be beneficial to find a formulation that balances the trade-off between bias and variance, allowing also for tuning. By using a telescopic sum for $t \to \infty$ of the δ_t^v terms and introducing an exponentially weighted average with the coefficient λ (which controls the trade-off between bias and variance), the *Generalized Advantage Estimate (GAE)* [75] can be calculated as follows:

$$\hat{A}_t^{GAE} = \sum_{k=0}^{\infty} (\gamma\lambda)^k \, \delta_{t+k}^v \tag{1.62}$$

At the end of each episode, the algorithm truncates the formula for $t = T$ to be able to apply this estimator to LSTM networks [76]. To calculate $\hat{A}_t^{GAE}$, the state-value function must still be estimated. To do this, a variety of different techniques can be employed. If a non-linear function approximator is used to represent the state-value function, the most straightforward approach is to solve a nonlinear regression problem by minimizing the *Mean Squared Error (MSE)*:

$$J^{MSE}(\mathbf{w}) = \hat{\mathbb{E}}_{\pi}\left[\left(v_{\mathbf{w}}\left(S_t\right) - \sum_{k=t}^{T} \gamma^{k-t} R_t\right)^2\right] \tag{1.63}$$

In order to calculate the expectancy operators, *batch learning* is used. In this machine learning technique, the model is trained on the entire data set in one go at each training

[7] This definition builds on the classic Temporal-Difference (TD) learning framework and introduces the concept of residuals to improve the estimation of the value function. Residuals are the discrepancies between the target values and the current value estimates. The target value is usually a combination of the immediate reward and the estimated value of the next state based on the current value function.

cycle. The parameters of the model are only updated once, after the entire training dataset has been processed. In reinforcement learning, this implies that a batch of trajectories (or roll-outs) is created through interaction with the environment prior to the training cycle. The amount of trajectories collected is determined by a hyperparameter known as the *batch size*. The architecture of the algorithm is shown more clearly in Figure 1.17.

Gradient ascent is used for both unrolled LSTM-based actor and critic networks, and Backpropagation Through Time (BPTT) is used to compute the gradients of the parameters $\boldsymbol{\theta}$ and $\boldsymbol{w}$ (as demonstrated in Equation 1.52). At each epoch k of the gradient-based optimizer (e.g. Adam), the update equations are implemented as follows:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \beta_\theta \widehat{\nabla J\left(\boldsymbol{\theta}_k\right)} \tag{1.64}$$

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \beta_w \widehat{\nabla J\left(\mathbf{w}_k\right)} \tag{1.65}$$

For each set of roll-out data, the gradient descent step is applied a certain number of times, called *epochs*, which is a hyperparameter. The algorithm will stop when the maximum number of predetermined learning steps is reached.

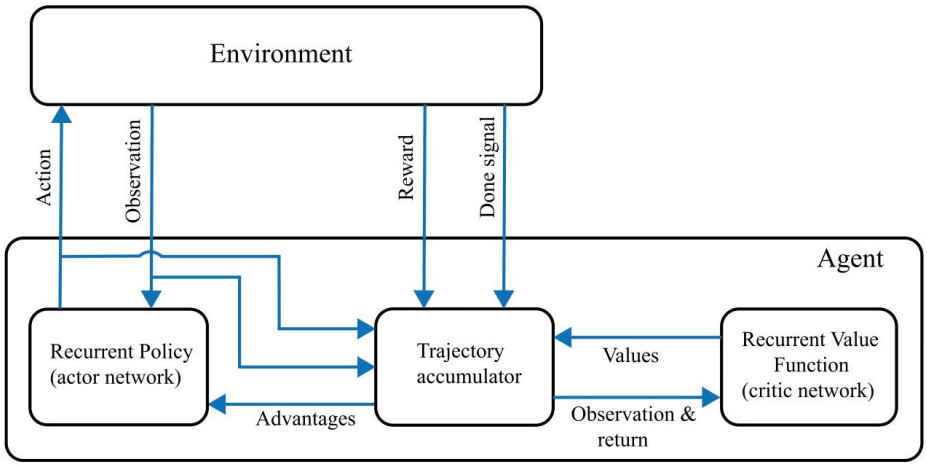


Figure 1.17: Meta-Reinforcement Learning (Meta-RL) training architecture for Recurrent Proximal Policy Optimization (Recurrent PPO) algorithm. Source: Reference [73].

Typical implementations track KL-divergence, which is sometimes used as an early stopping flag [85] or to adjust the PPO clipping parameter [74]. A biased estimator[8] is typically used to calculate it, as follows:

$$\widehat{KL}\left(\pi_\theta, \pi_{\theta,old}\right) = \frac{1}{2}\left(\log \pi_\theta - \log \pi_{\theta,old}\right) \tag{1.66}$$

[8] `http://joschu.net/blog/kl-approx.html`

The weights in the LSTM networks are typically initialized using either a *Xavier* or *Orthogonal* approach. This is a critical step, as incorrect initialization can lead to issues such as all layers learning the same feature, or vanishing/exploding gradients. The Xavier initialization method samples matrix weights from a normal distribution, such as $m \sim \mathcal{N}\left(\mu = 0, \sigma^2 = 1/n_{input}\right)$. The orthogonal initialization method is designed to prevent ill-conditioned gradients by starting from and then multiplying matrices with unitary eigenvalues. This technique creates a matrix by drawing weights from a normal distribution and then making the rows of the matrix orthogonal to one another through either the Gram-Schmidt process or the QR decomposition.

The algorithm described above, due to the specific version of GAE implemented and the use of BPTT, produces a version of PPO that is suitable for use with LSTM networks, called *Recurrent Proximal Policy Optimization (Recurrent PPO)* [52]. The following is an illustration of the algorithm:

Algorithm 1.2 Recurrent PPO

Initialization of neural network parameters $\boldsymbol{\theta}$ and $\mathbf{w}$
for step=1,2,..., learning steps **do**
  Reset environment
  for episode=1,2,..., batch size **do**
    Run Policy π_{old} in environment until *done*
    Compute advantage estimate $\hat{A}_t^{GAE}$
    Store trajectory into batch roll-out
  end for
  for iteration=1,2,..., epochs **do**
    Unroll Agent LSTMs
    Optimize Actor J^{PPO} wrt $\boldsymbol{\theta}$ and Critic J^{MSE} wrt $\mathbf{w}$
    $\boldsymbol{\theta}_{old} \leftarrow \boldsymbol{\theta}$, $\mathbf{w}_{old} \leftarrow \mathbf{w}$
  end for
end for

The implementation of Recurrent PPO, and PPO in general, involves a number of complex optimizations that are not discussed in depth or even mentioned in Schulman et al. [56]. However, these optimizations have been found to have a significant effect on the efficacy of PPO [86]. Examples of these include value-function clipping, reward and advantage normalization, learning rate annealing, global gradient clipping, state-independent logarithmic standard deviation, and many more. To ensure robustness, consistency, comparable results with other researchers, and high flexibility (once the environment is set up, one

can experiment with a variety of RL algorithms), this work utilizes the *Stable-Baselines3 (SB3)*[9] library. This is a collection of reliable implementations of reinforcement learning algorithms in *PyTorch*, an open-source machine learning library created mainly by the Facebook AI Research (FAIR) Laboratory.

Optimization Algorithm

The family of optimization algorithms used is *Gradient Descent (GD)*[87], a method in which the parameters $\boldsymbol{\theta}$ of an objective function J are adjusted in the opposite direction of the gradient of the objective function $\boldsymbol{\nabla} J(\boldsymbol{\theta})$ with a step size α called the *learning rate*[10]. This process, which ends when the gradient is zero, eventually leads to a local optimal solution based on the batch of data used to estimate the gradient. The standard equation of update is the following:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha \widehat{\boldsymbol{\nabla} J(\boldsymbol{\theta}_k)} \tag{1.67}$$

Methods in this family typically compute each update to the parameters based on the estimated cost function gradient using only a portion of the overall training data. Algorithms that use the entire training set are known as *Batch Gradient* methods, while those that use a subset of training examples are usually called *Stochastic Gradient* methods. Reinforcement learning algorithms, such as Algorithm 1.2, often use the last type, which involves the gradient descent of Equation 1.67 with the gradient computed using a roll-out of trajectories. This is referred to as *Stochastic Gradient Descent (SGD)* [77].

Stochastic Gradient Descent (SGD) has inherent randomness or variability in gradient estimates that come from using a randomly chosen subset of training data in each iteration. This noise is a key feature of SGD and sets it apart from deterministic optimization techniques such as batch gradient descent. To address this problem, two strategies can be beneficial [77]:

1. *Momentum* can be used to accelerate the learning process by accumulating an exponentially decaying moving average of past gradients. *SGD with momentum* is an algorithm that implements this concept and computes the first moment estimate $\mathbf{m}_k$ in the learning step k as follows:

[9] `https://stable-baselines3.readthedocs.io/en/master/index.html`

[10] A high learning rate implies that the system is responsive to new information, adapting quickly. However, this also means that the system is more likely to forget and replace old data. On the other hand, a low learning rate makes the system less flexible, leading to a slower learning process and less susceptibility to noise.

$$\mathbf{m}_k = \beta_1 \mathbf{m}_{k-1} + (1 - \beta_1)\, \widehat{\boldsymbol{\nabla} J(\boldsymbol{\theta}_k)} \tag{1.68}$$

The first moment estimate $\mathbf{m}_k$ is the mean of the gradients, with a momentum hyperparameter β_1.

2. *Adaptive learning rate* decreases the size of steps over time to reduce the amount of noise that is present when approaching the minimum. The *RMSProp* algorithm uses an adaptive learning rate α_k which is calculated at the learning step k as follows:

$$\alpha_k = \frac{\alpha}{\sqrt{v_k} + \epsilon} \tag{1.69}$$

$$v_k = \beta_2 v_{k-1} + (1 - \beta_2) \left(\widehat{\boldsymbol{\nabla} J(\boldsymbol{\theta}_k)} \right)^2 \tag{1.70}$$

The second moment estimate v_k is the mean of the squared gradient values, with a decay factor β_2 and a smoothing parameter ϵ.

The *Adaptive Moment (ADAM) Estimation* method [88] includes concepts of momentum and adaptive learning rate. Momentum is taken into account through a biased estimate of the first gradient moment $\mathbf{m}_k$, while adaptive learning rate is taken into account through a biased estimate of the second gradient moment v_k. In particular:

$$\mathbf{m}_k = \beta_1 \mathbf{m}_{k-1} + (1 - \beta_1)\, \widehat{\boldsymbol{\nabla} J(\boldsymbol{\theta}_k)} \tag{1.71}$$

$$v_k = \beta_2 v_{k-1} + (1 - \beta_2) \left(\widehat{\boldsymbol{\nabla} J(\boldsymbol{\theta}_k)} \right)^2 \tag{1.72}$$

These estimators are biased because the initial values of the first and second moments are initially set to zero. Consequently, their bias must be adjusted in the course of the iterations, leading to bias-corrected estimates of the first and second moments as follows:

$$\hat{\mathbf{m}}_k = \frac{\mathbf{m}_k}{1 - \beta_1^k} \tag{1.73}$$

$$\hat{v}_k = \frac{v_k}{1 - \beta_2^k} \tag{1.74}$$

Taking into account the adjusted moments, the Adam algorithm step is computed as follows:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \frac{\alpha}{\sqrt{\hat{v}_k} + \epsilon} \hat{\mathbf{m}}_k \tag{1.75}$$

It should be noted that, typically, the decaying rates are set to the default values of 0.9 and 0.999, as recommended by Goodfellow et al. [77]. The same is true for the smoothing parameter ϵ, which is set to 10^{-8}. The only hyperparameter that must be manually chosen is the learning rate. Adam is usually quite tolerant when it comes to parameter tuning, although the learning rate may need to be changed from the recommended default.

2 | Problem Formulation

> I am, and ever will be, a white socks, pocket protector, nerdy engineer.
>
> Neil Armstrong

This chapter examines the application of reinforcement learning to an autonomous spacecraft guidance and control problem. It starts by describing the spacecraft's final approach and docking procedure that is taken into account in this work. Then it goes into the implementation approach, including the formulation of the Markov Decision Process (MDP) formulation, neural network design, and hyperparameter selection. This two-part narrative provides an overview of the complexities of the problem and the technical solutions that make up the proposed approach.

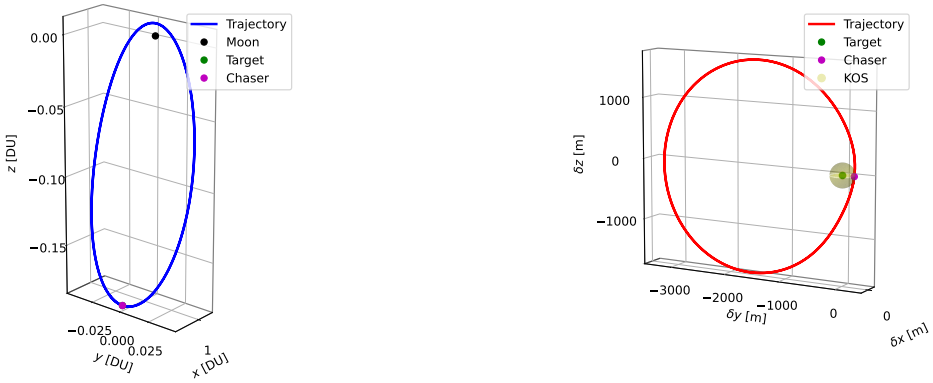
2.1. Study Case

The objective of this work is to design an autonomous G&C algorithm for a spacecraft's final approach and docking phases in cislunar space. This algorithm will move the spacecraft from *GO Final Approach* to *GO Docking* in a secure way (Section 1.1). Before this stage, stable/unstable manifolds offer a reliable and robust natural corridor for a chaser spacecraft, which can be utilized in approaching the target with passive safety assurances. The chaser should aim to utilize these orbits to complete the far-range rendezvous (e.g., 100 km) and then directly inject into the NRHO at a point situated on the edge of the Keep-Out-Sphere (KOS). Within this region, it is essential to have continuous control and active collision avoidance; however, periodic central manifolds can still be used as passive safe holding points, either on the edge of the KOS or within it.

If the chaser spacecraft is situated at the *Southern L_2 9:2 Resonant NRHO* apolune at a certain distance from the target, it can be used as a holding point until it is given the go-ahead to begin its final rendezvous stages. The initial conditions of this motion would guarantee that, when this strategy is used, collisions with the target are passively avoided. Additionally, Reference [24, 25] has identified the apolune region of NRHOs as the most suitable point for complex controlled proximity operations due to its slower dynamics,

which provides greater safety and less fuel consumption. Therefore, the NRHO apolune is chosen as the starting point for the maneuver, and the chaser is situated on a holding point within the KOS that is behind the target in the direction of the orbital motion.

This concept is illustrated in Figure 2.1, where it is possible to observe from an absolute perspective the initial position on the NRHO and from a relative perspective the passive safety of the holding point on the periodic manifold (no collision over the course of a single orbit). The equations of motion, both absolute and relative, and the reference frames associated with them are outlined in Section 1.2.



(a) Absolute synodic reference frame.

(b) Relative synodic reference frame.

Figure 2.1: Chaser and Target on the Southern 9:2 Resonant Near-Rectilinear Halo Orbit (NRHO) of the Earth-Moon system apolune at a relative distance of 200 meters, Keep-Out-Sphere (KOS) edge, during one orbital motion.

The present study takes into account the Lunar Gateway as the target and the Orion spacecraft as the chaser. This work includes the main characteristics of Orion (Table 2.1). The OMS-E and Aux-E thrusters are oriented in the direction of the $\hat{\mathbf{c}}_2$ body-axis.

	Value
Mass [kg]	21000
Thrust [kN]	25.7 (OMS-E), 0.45x8 (Aux-E)
I_{sp} **[s]**	310

Table 2.1: Data and primary characteristics of NASA's Orion spacecraft.

The RVD maneuver shall respect certain requirements[1] to ensure the safety of the ap-

[1] **Shall**: Mandatory requirements or hard requirements, which must be implemented. They constitute

proach and correct docking. A review of the literature's fundamental requirements for the entire maneuver is conducted in Section 1.1, and the list of constraints to be taken into account for the algorithm, which focuses solely on the final approach and the docking phase, is as follows:

- **Docking Port**: in order to ensure a successful automated docking, the final relative position and velocity *shall* be $\rho_f < \rho_{f,max}$ and $\dot{\rho}_f < \dot{\rho}_{f,max}$ respectively. The selected values are $\rho_{f,max} = 1\ m$ and $\dot{\rho}_{f,max} < 0.1\ m/s$;
- **Approach Corridor**: the chaser *shall* remain within a truncated cone with a half angle of $\beta = 20\ °$ for safety and vision-based navigation purposes. Assuming that the target docking port is located on its positive $\hat{\mathbf{t}}_2$ direction, the truncated cone requirement is expressed as follows:

$$\boldsymbol{\rho}^*(t) \cdot \hat{\mathbf{t}}_2 - \rho^*(t)\, cos(\beta) > 0 \qquad \forall \delta y \geq 0, \forall t \tag{2.1}$$

$$\boldsymbol{\rho}^*(t) = \boldsymbol{\rho}(t) + \left[0,\ \frac{\rho_{f,max}}{\tan\beta},\ 0\right]^T \tag{2.2}$$

A truncated cone [1], known as a frustrum, offers a more realistic docking port model (Figure 2.2) and avoids singularities, enhancing the convergence properties of the algorithm. Additionally, the unrestricted small frustrum base proves advantageous; as the spacecraft progresses, it facilitates the docking mechanism by nearing the end of the docking port stroke (noting that post-docking stability should be addressed).

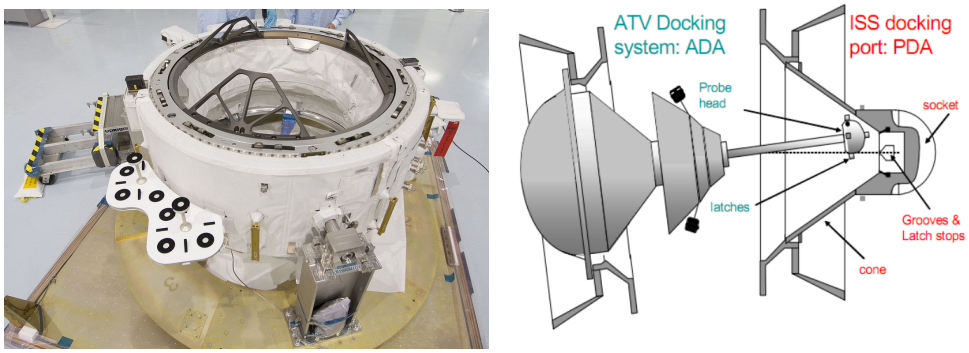


Figure 2.2: Androgynous Peripheral Docking System (APDS) of Orion spacecraft on left, Sistema Stykovki i Vnutrennego Perekhoda (SSVP) of Soyuz spacecraft on right. Source: `https://tinyurl.com/5e4zrkpv`.

This is proposed by Lee [22] and is achieved by using an alternate relative position $\boldsymbol{\rho}^*$ with respect to a point in the $-\hat{\mathbf{t}}_2$ direction to have a small frustrum base that

the core requirements of the project to fulfill the mission objectives. **Should**: These requirements aim to improve the performance or outcome of the mission

meets the docking port requirement. This work assumes that the origin of the target body frame $\mathcal{B}_T : \{\hat{\mathbf{t}}_1,\ \hat{\mathbf{t}}_2,\ \hat{\mathbf{t}}_3\}$ is situated in the center of the docking port opening. The chaser will aim for this point during the RVD maneuver, and then the docking port system will take care of the rest. The requirement should be stated in the body frame of the target, as the orientation of the cone axis depends on its attitude. Therefore, it is assumed also that the target remains aligned with the Earth-Moon synodic frame such that $\hat{\mathbf{t}}_2 \equiv \hat{\mathbf{y}}$. The attitude of the chaser should be adjusted in accordance with its docking port's position in its body frame. This situation is depicted in Figure 2.3.

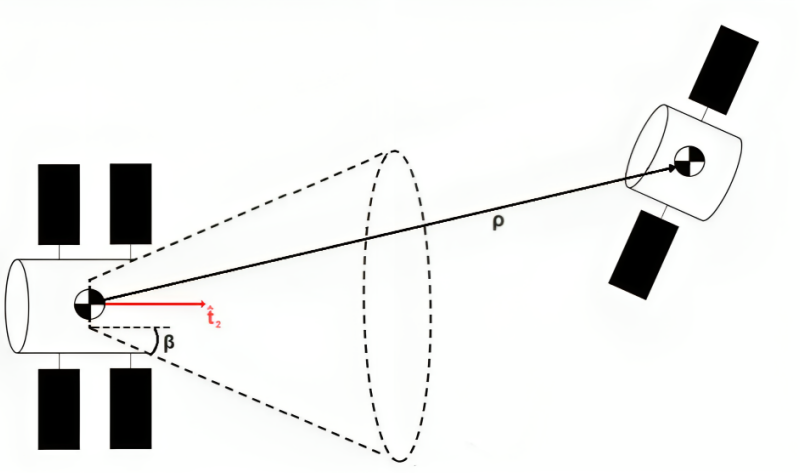


Figure 2.3: The approach corridor is represented by a truncated cone with a maximum semi-angle of β and a small frustum base designed to satisfy the docking port requirements.

- **Time-Of-Flight (ToF)**: the RVD maneuver *shall* be completed in a limited period of time. To guarantee success, it is essential to be able to set limits on the time of flight for such missions, thus $t < ToF_{max}$ must be fulfilled. The value of ToF_{max} will be determined on an individual basis in the subsequent chapters;
- **Thrusters Performance**: the data sheet's maximum thrust value *shall* be adhered to in order to create a viable control action profile. It is essential to ensure that $u\,(t) < u_{max}$, where $u_{max} = 29.3\ kN$ according to Table 2.1.

2.2. Optimal Control Problem Formulation

Taking into account the requirements mentioned above, it is simple to create an *Optimal Control Problem (OCP)*: *"Find the control action profile that minimizes control effort and such that dynamics, initial conditions, maximum control action, maximum time of flight,*

approach corridor and final docking port requirements are respected". It can be expressed mathematically as follows:

$$\begin{aligned}
\min_{\mathbf{u}} \quad & J = \int_{t_0}^{t_f} ||\mathbf{u}(t)||dt \\
\text{s.t.} \quad & \dot{\mathbf{x}}(t) = \mathbf{f}\left(\mathbf{x}(t), \mathbf{u}(t)\right) \quad \forall t \\
& \mathbf{x}(t_0) = \mathbf{x}_0 \\
& u(t) < u_{max} \quad \forall t \\
& t < ToF_{max} \\
& \boldsymbol{\rho}^*(t) \cdot \hat{\mathbf{t}}_2 - \rho^*(t) cos(\beta) > 0 \quad \forall \delta y \geq 0, \forall t \\
& \rho\left(t_f\right) - \rho_{f,max} < 0 \\
& \dot{\rho}\left(t_f\right) - \dot{\rho}_{f,max} < 0
\end{aligned} \tag{2.3}$$

The dynamics of the system, referred to as $\mathbf{f}$, is extensively covered in Section 1.2. The state $\mathbf{x}(t)$ within the ODEs comprises the absolute target state, the relative chaser state, and its associated mass. The details of the other constraints are discussed in Section 2.1. To solve the Optimal Control Problem (OCP) with Meta-Reinforcement Learning (Meta-RL), it is converted to a Markov Decision Process (MDP) in Section 2.3. The Markov property is satisfied once the problem is discretized in time, so it can become an MDP.

2.3. Markov Decision Process Formulation

The agent's goal in a *Markov Decision Process (MDP)* is: *"Find the control policy π^* that maximizes the discounted expected return of rewards received along a trajectory and such that the policy, the environment's dynamics, and the initial conditions are respected"*. The MDP is mathematically expressed as follows:

$$\begin{aligned}
\max_{\pi} \quad & J = \mathbb{E}_{\pi}\left[G_t \mid \mathbf{S}_t\right] \\
\text{s.t.} \quad & \mathbf{A}_t = \boldsymbol{\pi}(\mathbf{S}_t) \\
& \mathbf{S}_{t+1} = \phi(\mathbf{S}_t, \ \mathbf{A}_t) \\
& \mathbf{S}_0 \sim \mathcal{N}\left(\hat{\mathbf{S}}_0, \ \mathbf{C}\right)
\end{aligned} \tag{2.4}$$

The dynamics of the environment, discussed in Subsection 2.3.3, is taken into account by the second constraint, which produces the system state $\mathbf{S}_t$ (not to be confused with the ODEs state $\mathbf{x}_t$) at each time step t. MDPs are capable of handling both deterministic

and stochastic dynamics; this work takes into account the latter in order to strengthen the reliability of the algorithm. Operational constraints (docking port requirements, approach corridor, flight duration, and maximum control action) and fuel minimization are "translated" by specifying the reward function in Subsection 2.3.2. MDPs are not as elegant at dealing with constraints as OCPs; in fact, constraints are expressed as large positive or negative rewards or the end of an episode. The last constraint in Equation 2.4 specifies the initial state as a random variable drawn from a normal distribution.

2.3.1. State and Action Spaces

The neural networks of the agent use the *state space* as input, which is a set of coordinates that provide information to generate the action or prediction of the state-value function. As suggested by Wilson and Riccardi [89] in a spacecraft RVD problem, it is beneficial to include both the absolute state of the target and the relative state of the chaser. The mass of the chaser is also taken into account; optimization of fuel consumption should be part of the reward function. To meet time constraints, the remaining flight time $\Delta T_t = ToF_{max} - t$ is also included. Furthermore, to keep track of all episodes' history and allow LSTMs to internally update their dynamics more effectively, Hochreiter et al. [47] emphasize the importance of including the action and reward of the preceding time step in the observations. Therefore, the state space $\mathbf{S} \in \mathbb{R}^{18}$ consists of:

$$\mathbf{S}_t = \left[\mathbf{x}_t^T,\ \delta\mathbf{x}_t,\ m_t,\ \Delta T_t,\ \mathbf{u}_{t-1},\ R_{t-1}\right] \tag{2.5}$$

In ANNs, it is advisable to normalize inputs to ensure consistent feature scales and enhance stability during training, preventing certain features from exerting excessive influence, which can lead to slower and less reliable convergence. Hence, a straightforward *Min-Max normalization* [90] is chosen and the normalized state $\mathbf{S}_t^* \in [-\mathbf{1},\ +\mathbf{1}]$ is created for each of its elements using the following equation:

$$S_{t,i}^* = 2\left(\frac{S_{t,i} - S_{min,i}}{S_{max,i} - S_{min,i}}\right) - 1 \tag{2.6}$$

Once the problem is defined, the extremal components of $\mathbf{S}_t$ are identified. The specific zero-centered Min-Max approach utilized, especially in combination with the selected activation functions outlined in Section 2.4, effectively addresses symmetry issues in weight initialization and enhances the gradient distribution.

The actor neural network generates a set of all the possible actions that the agent can take in a given environment, which is called the *action space*. In this work, since a

3DOF spacecraft rendezvous and docking problem is being studied, the action space would represent the thrust action. Different models have been used in the literature to represent the control action of thrusters as an action space. The most suitable formulation for this study case [36] consists of an action space $\mathbf{A}_t \in \mathbb{R}^3$ and its unscaled control action $\mathbf{u}_t$ defined as:

$$\mathbf{A}_t = \left[\tilde{u}_{x,t},\ \tilde{u}_{y,t},\ \tilde{u}_{z,t}\right] \qquad \mathbf{u}_t = \sigma\tilde{\mathbf{u}}_t \tag{2.7}$$

The control action $\mathbf{u}_t$ is the input of the spacecraft equations of motion. A definition that directly includes the components of the thrust vector is preferable to one that utilizes the angle in-plane and out-of-plane angle (which has discontinuity issues) because it has better convergence characteristics [36]. To ensure that the maximum thrust value is respected, the scaling coefficient is set to $\sigma = u_{max}/\|\mathbf{1}_3\|$, as the output of the neural network is limited to $[-1, 1]$. As demonstrated by Lorenzo Capra and Lavagna [91], a continuous action space takes longer to train but is more flexible and realistic, resulting in higher performance than a discrete action space.

2.3.2. Reward Function

The reward signal is a way to communicate to the agent what you want it to achieve, not how you want it to be achieved. A spacecraft rendezvous and docking G&C problem can be seen as a sparse-reward RL problem, where bonus or penalty rewards are only given for successful docking or any collisions with the approach corridor. To improve the performance of reinforcement learning in such cases, reward shaping techniques should be used, as suggested by Ng et al. [92]. Non-linear functions, such as *log* and *exp*, possess the property of increasing their first derivative near attractive or repulsive states [36]. These functions can provide a gentle push or pull, gradually guiding the agent away from or towards certain state regions, making the reward landscape appear smoother. The reward function used is as follows:

$$\begin{aligned} R_t = \alpha \log\left(\frac{\|\delta\mathbf{x}_t \oslash \delta\mathbf{x}_{max}\|}{\|\mathbf{1}_6\|}\right)^2 - \lambda \exp\left(\frac{acos\left(\hat{\boldsymbol{\rho}}_t^* \cdot \hat{\mathbf{t}}_2\right)}{\pi}\right)^2 - \gamma \exp\left(\frac{u_t}{u_{max}}\right)^2 \\ + \left(\rho_t < \rho_{f,max} \wedge \dot{\rho}_t < \dot{\rho}_{f,max} \Rightarrow\right)\zeta - \left(\boldsymbol{\rho}_t^* \cdot \hat{\mathbf{t}}_2 - \rho_t^* cos\left(\beta\right) < 0 \Rightarrow\right)\kappa \end{aligned} \tag{2.8}$$

The reward function is made up of two components: dense rewards, which give continuous feedback within each episode, and episodic rewards, which are given at the conclusion of a task or episode. Specifically, they are as follows:

- Dense bonus and penalty rewards:

  1. Ideally, the final relative state should be zero, so a squared natural logarithm is employed to gradually increase the bonus towards the goal. Square increases the convergence performance [93]. To ensure that each component is given the same importance, the relative state is normalized[2] with respect to its assumed maximum values. The logarithmic argument, due to the L_2-norm and the division by $\|\mathbf{1}_6\|$, is always within the range of $[0,\ 1]$, meaning that the greatest distance is not rewarded. The dense bonus term is then:

$$R_{1,t} = \alpha \log \left(\frac{\|\delta \mathbf{x}_t \oslash \delta \mathbf{x}_{max}\|}{\|\mathbf{1}_6\|} \right)^2 \tag{2.9}$$

  2. The angular distance from the approach direction and the control effort are penalized by squaring the exponential of the normalized values, which are always within $[0,\ 1]$. The two tuning coefficients guarantee that there is almost no penalty when the arguments are zero. The unit vector $\hat{\boldsymbol{\rho}}_t^*$ is $\boldsymbol{\rho}_t^*$ normalized. The penalty is expressed as:

$$R_{2,t} = -\lambda \exp \left(\frac{acos\left(\hat{\boldsymbol{\rho}}_t^* \cdot \hat{\mathbf{t}}_2\right)}{\pi} \right)^2 - \gamma \exp \left(\frac{u_t}{u_{max}} \right)^2 \tag{2.10}$$

The dense reward logic, with numerical values only for illustration, can be summarized and shown in Figure 2.4 using these functions.

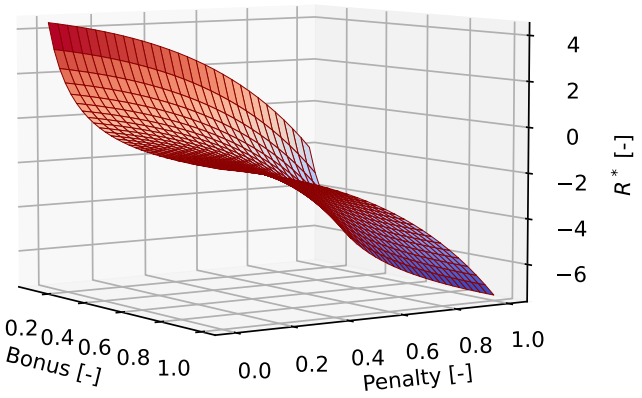


Figure 2.4: Graphic illustration of the reward shaping employing logarithmic and exponential functions. The bonus and penalty axes represent their input arguments.

[2]Hadamard division, denoted as $\oslash$, divides corresponding elements of two matrices element-wise, creating a new matrix of the same dimensions.

- Episodic bonus and penalty rewards:

  1. If the desired relative state is achieved, which meets the docking port requirements, a bonus is awarded as follows:

$$R_{3,t} = \left(\rho_t < \rho_{f,max} \wedge \dot{\rho}_t < \dot{\rho}_{f,max} \Rightarrow\right) \zeta \tag{2.11}$$

     If this situation occurs, the episode is finished with a *done* signal, signifying the success of the mission. Thus, it also bounds the reward of Equation 2.9.

  2. The agent incurs a penalty reward if the trajectory collides with the approach corridor, such as:

$$R_{4,t} = -\left(\boldsymbol{\rho}_t^* \cdot \hat{\mathbf{t}}_2 - \rho_t^* cos\left(\beta\right) < 0 \Rightarrow\right) \kappa \tag{2.12}$$

     If it occurs, the episode ends with a *done* signal. This situation indicates that the mission has not been successful, as the safety requirements have not been met. Consequently, the reward of Equation 2.10 is also bounded.

The coefficients are selected via a trial-and-error approach, as conducting an extensive hyperparameter search using Bayesian optimization is beyond the scope of this work. The chosen values consist of $\alpha = 0.02$, $\lambda = 0.1$, $\gamma = 0.01$, $\zeta = 100$, and $\kappa = 30$.

2.3.3. Environment's Dynamics

Given the continuous nature of an RVD problem, it is challenging to create an explicit transition function with conditional probabilities that accurately reflects state transitions as in Equation 1.22. Therefore, the transition function is represented by a generative model $\mathbf{S}_{t+1} = \phi\left(\mathbf{S}_t,\ \mathbf{A}_t\right)$. This model generates the updated absolute target and relative chaser states based on CRTBP (Equation 1.2) and RCRTBP (Equation 1.17), respectively. Additionally, it updates the chaser mass, the remaining flight time, as well as the action and reward from the previous time step. Equations of motion $\mathbf{f}$ are broken down into discrete steps by integrating and considering a constant control action between the time intervals, resulting in $\mathbf{x}_{t+1} = \boldsymbol{\psi}\left(\mathbf{x}_t,\ \mathbf{u}_t\right)$. At the beginning of each MDP episode, a random initial state $\mathbf{S}_0$ is chosen, as described in Section 2.3. This randomness exclusively affects the initial conditions of the relative chaser dynamics $\delta\mathbf{x}_0 = [\boldsymbol{\rho},\ \dot{\boldsymbol{\rho}}]$ and its initial mass m_0, derived from three very broad normal distributions[3]. The means of the distributions are

[3]In a real-space application, such extensive uncertainties might seem unjustified given the better-known initial conditions. However, they are utilized to test Meta-RL's capacity to handle substantial parameter randomization, showcasing its adaptability. Additionally, successful satisfaction of the constraint under these conditions would further demonstrate the robustness of the algorithm.

taken as the initial conditions, with standard deviations of $\sigma_{\boldsymbol{\rho}_0} = 0.1\rho_0$ (the uppermost value ensuring $\boldsymbol{\rho}_0$ remains within the approach corridor), $\sigma_{\dot{\boldsymbol{\rho}}_0} = 0.5\ m/s$ (to cover the full range of velocity states throughout Figure 2.1b) and $\sigma_{m_0} = 100\ kg$ (reasonable given the m_0 provided in Table 2.1). This approach aimed to integrate the broadest possible range of initial condition distributions. Furthermore, the environment sends out a *done* when the time of flight exceeds its highest value (ToF_{max}), similar to what the reward function does with collision and docking events; this concludes the OCP "translation" in MDP.

In the environment model, additional stochastic and random elements are included to further demonstrate the robustness of the algorithm and to emphasize the features of Meta-RL, which is a great benefit of RL methods compared to the straightforward OCP "translation" [21]. Process noise is included in the chaser spacecraft dynamics to account for unmodeled accelerations (e.g. SRP, the gravity of other celestial bodies, thrust uncertainty, etc.). The equations of motion for the chaser, which include $\mathbf{h}$ as the non-autonomous RCRTBP expressed in Equation 1.17, are as follows:

$$\delta\dot{\mathbf{x}}(t) = \mathbf{h}\left(\delta\mathbf{x}(t), \mathbf{u}(t), \mathbf{x}^T(t)\right) + \mathbf{w}(t) \tag{2.13}$$

$$\mathbf{w}(t) \sim \mathcal{N}\left(\mathbf{0},\ diag\left(\mathbf{0}_3, \mathbf{10_3^{-8}}\right)\right) \tag{2.14}$$

$\mathcal{N}$ represents a *Gaussian White Noise (GWN)*, featuring a covariance matching the non-dimensional magnitude ($\sigma = 10^{-4}$) of typical NRHO disturbances. This simplification assumes independence and stationarity in uncertainty, which is not always the case in reality. Despite this, the normal distribution is widely used due to its mathematical convenience and its alignment with the *Central Limit Theorem* [94]. This theorem suggests that the sum of various dynamic disturbances often tends to follow a normal distribution. The robustness of the algorithm is further enhanced by taking into account and modeling the possibility of a random failure in the control action. At the beginning of each episode, the environment randomly selects a control direction in which to reduce the magnitude of the thrust by 50% or have no failure. This is illustrated by a multinomial distribution in which each case has an equal chance of occurring at 25% as follows:

$$P_{fail} = \begin{cases} 0.5 \cdot u_x & 25\% \\ 0.5 \cdot u_y & 25\% \\ 0.5 \cdot u_z & 25\% \\ No\ Failure & 25\% \end{cases} \tag{2.15}$$

As a result, there is a 75% chance per episode of being unable to produce the required thrust in one direction, effectively simulating a breakdown in 6DOF control.

2.4. Artificial Neural Networks Architecture

Finding the correct architecture and equilibrium between small and large neural networks is essential to achieve a satisfactory result and to possess strong generalization characteristics. The architecture of the model is inspired by *Stable-Baseline3 (SB3) Recurrent PPO*, featuring layers of LSTM followed by layers of MLP extractor, as shown in Table 2.2. The use of a combination of LSTM networks and a MLP extractor layer is a common approach in Reinforcement Learning (RL), as noted by Sak et al. [95]. This is because LSTMs are particularly adept at capturing sequential dependencies, but may not be able to provide a suitable final representation. An MLP-layer at the end is used to condense the variable-length sequence of LSTM outputs into a fixed-size representation, making it easier to make decisions or take actions. This combination of sequence modeling (handled by LSTMs) and representation learning (handled by MLPs) allows the model to effectively use sequential information in RL tasks. The choice of the hyperbolic tangent as an activation function is based on its symmetry and widespread use. The size of the neural network is determined through experimentation to guarantee a high level of generalization and to accurately fit the optimal solution. It consists of two LSTM-layers with 256 neurons and two MLP-layers with 64 neurons for both the agent's neural networks.

Layer	Neurons (A/C)	Activation	Type
Input	16/16	Linear	
Hidden Layer 1	256/256	Tanh	LSTM
Hidden Layer 2	256/256	Tanh	LSTM
Hidden Layer 2	64/64	Tanh	MLP
Hidden Layer 2	64/64	Tanh	MLP
Output	3/1	Linear	

Table 2.2: Architecture of the Artificial Neural Network (ANN) implemented in the Actor (A) and Critic (C) networks of the agent.

During the development of this work, a single layer of LSTM cells did not perform as well as expected in terms of the mission success rate. In theory, two layers could slow down the training phase and reduce generalization performance, potentially leading to an issue of overfitting; however, this is beyond the scope of this work and should be addressed in the future. For now, two layers of LSTM have been the best choice to accurately fit the MDP (Section 2.3) implemented.

2.5. Hyperparameters Selection

In reinforcement learning, hyperparameters are not learned by the algorithm itself but are instead set by the user through trial and error. These hyperparameters are essential in determining the behavior and performance of the RL agent during training, and can have a major effect on the training process and the final results. The hyperparameters chosen for this work are listed in Table 2.3.

The hyperparameters chosen are not far from the default settings [33] and only required minor adjustments. It is recommended to use a batch size of less than 128 to increase the variance of the gradient estimate and to add entropy to promote the algorithm's exploration. Gradient clipping is a particularly important hyperparameter for LSTM networks, as they are prone to catastrophic forgetting. This phenomenon occurs when a network that has been trained for a task loses its ability to perform that task when trained for a new one, due to overwriting of previously acquired knowledge [96]. Gradient clipping in LSTM networks can help reduce this effect by stabilizing the training process and preventing large weight updates that could disrupt previously learned information. Additionally, when setting the GAE parameter to 1, all potential rewards are considered equally in estimating the advantage function, which is beneficial for spacecraft proximity operations. However, there are drawbacks to this approach, such as increased variance, sensitivity to noise, delayed convergence, and the risk of overfitting. Despite the success of the study, it is evident that these potential issues should be addressed in the future through a comprehensive hyperparameter search, which is not included in this work.

	Actor PPO	**Critic MSE**
Batch Size	64	64
Number Epochs	10	10
GAE Lambda (λ)	1	-
Discount Factor (γ)	0.99	-
Clip Parameter (ε)	0.1	-
Learning Rate (α)	0.00005	0.00005
Entropy Coefficient	0.0001	-
Clip Gradient	0.1	0.1

Table 2.3: Selection of hyperparameters for Actor's training using Proximal Policy Optimization (PPO) and Critic's training through Mean Squared Error (MSE) algorithms.

It should be noted that, even though different tasks may require different optimal hyperparameters, the limited computing budget and time available make it unfeasible to tune each study case (and agent neural network) individually. Furthermore, since problems share similar dynamics, action/state space, and reward functions, it is likely that they will have similar optimal hyperparameter configurations, which justifies the use of the same set for all study cases.

3 | Numerical Results

> Everything is relative; and only that is absolute.
>
> Auguste Comte

This chapter provides an in-depth analysis of the numerical results obtained. Meta-Reinforcement Learning (Meta-RL) agents' training and testing phases are examined, and a variety of graphs are displayed to demonstrate the practicality and effectiveness of Meta-RL for Rendezvous and Docking (RVD) maneuvers. The dependability and computational efficiency of the strategy are evaluated through Monte Carlo campaigns in the environment. The findings ultimately show that Meta-RL can be used as an autonomous Guidance and Control (G&C) strategy for RVD in the cislunar space.

3.1. Training and Testing: Nominal Study Case

In this section, the numerical results of the algorithm discussed in Section 1.5 are presented. The RVD problem discussed in Chapter 2 is solved using this algorithm, which is specifically transformed into an MDP in Section 2.3. Different initial conditions and flight times are taken into account, as various Holding Points (HP) within the Keep-Out-Sphere (KOS) are considered. At the end of this process, an autonomous G&C policy is obtained for the situations examined.

Reinforcement learning can be broken down into two distinct stages: training and testing. During the training period, the policy is optimized to create an agent that is capable of performing a desired task. During the testing phase, no further learning occurs; instead, the optimized agent is placed in an environment similar to the one it has been trained in and the policy is used to complete the actual task. The first phase for spacecraft G&C applications would be conducted using high-fidelity simulators on the ground. This is because there is not enough computing power in space, and missions cannot be put at risk by experimenting and making mistakes. Once the policy has been trained, testing is simply a matter of deploying and evaluating it on board, which involves a straightforward multiplication of a few matrices to generate closed-loop control actions.

The code used in this study is accessible on a dedicated GitHub repository [1]. The equations of motion are solved in the MDP environment with the *scipy.integrate* library in *Python*, using the non-stiff Adam predictor-corrector, called *LSODA*, method with relative and absolute tolerances of $2.22 \cdot 10^{-14}$. The integration time step for converting continuous dynamics into an MDP is set to $dt = 0.5\ s$. The device used for this work is a Laptop PC with an Intel(R) Core(TM) i7-8565U CPU 1.99 GHz and 16.0 GB of RAM.

3.1.1. Final Approach and Docking: 50-meter Case

It is now considered a 50-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case. The reference frame used is outlined in Subsection 1.2.2. The absolute target and the relative chaser initial conditions, expressed in a dimensionless form, are as follows:

$$\mathbf{x}_0^T = \begin{bmatrix} 1.02206694 \\ -1.32282592 \cdot 10^{-7} \\ -1.82100000 \cdot 10^{-1} \\ -1.69229909 \cdot 10^{-7} \\ -1.03353155 \cdot 10^{-1} \\ 6.44013821 \cdot 10^{-7} \end{bmatrix} \quad \delta\mathbf{x}_0 = \begin{bmatrix} 1.08357767 \cdot 10^{-13} \\ 1.32282592 \cdot 10^{-7} \\ -4.12142542 \cdot 10^{-13} \\ 1.69229909 \cdot 10^{-7} \\ -3.65860120 \cdot 10^{-13} \\ -6.44013821 \cdot 10^{-7} \end{bmatrix} \tag{3.1}$$

The chaser spacecraft is located at the *Southern L_2 9:2 Resonant NRHO* aposelene, while the target is 50 meters away in the clockwise direction of the orbit. The initial mass of the chaser spacecraft is taken from Table 2.1 and the maximum time of flight (ToF_{max}) is set to 50 s, in order to complete the problem definition described in Section 2.1.

Training

The agent is trained for 5M steps, which is equivalent to 28.9 days in a simulated environment. This process takes 2.3 days and the training curve, the mean discounted cumulative reward of the trajectory roll-outs in relation to the learning step, is shown in Figure 3.1.

The learning curve demonstrates that the reward gradually increases until it reaches a plateau, signifying the success of the learning. The mean reward achieved, thus the optimality of the LSTM-policy, will be evaluated and compared in Section 4.1. For now, it is only essential to determine if the algorithm has converged. The recurrent agent learned to consistently maximize the reward throughout each episode, thereby resolving the associated MDP. According to Sak et al. [95], Long Short-Term Memory (LSTM)

[1] `https://github.com/giovannifereoli/ThesisVer2`

networks, due to their inherent adaptability, may slightly lean toward better or worse solutions when further training is done; however, this is not the case. Once the neural network has been trained, the next step is to assess its performance as a G&C algorithm in the following section.

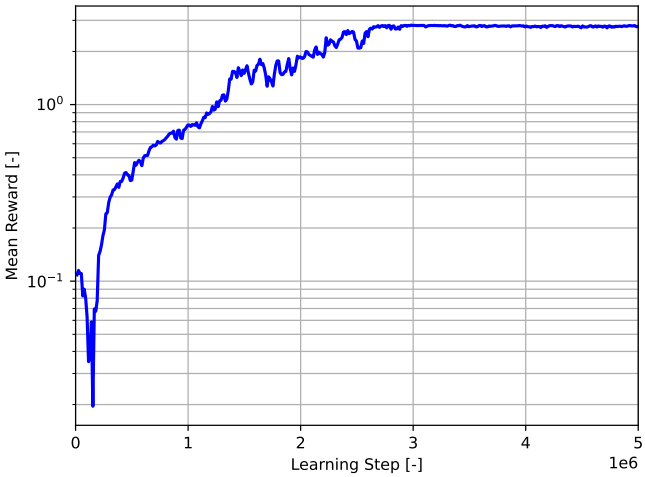


Figure 3.1: Mean discounted cumulative reward of the trajectory roll-outs during the training phase of the LSTM-policy for the 50-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case.

Testing

The LSTM-policy that has been trained is now being used again in the environment for the testing phase, with the same uncertainties as in Section 2.3. The results of deployment are demonstrated through the trajectory in Figure 3.2, as well as the relative position, relative velocity, mass, and thrust profile in Figure 3.3, which are based on a single episode.

Figure 3.2 shows that the chaser spacecraft, starting from a random initial condition (as described in Section 2.3 of the MDP), can reach the target and meet the safety requirements of the approach corridor with a final state of $\delta\mathbf{x}_f = \left[0.578\ m,\ 0.089\ m/s\right]$. Figures 3.4a and 3.4b demonstrate that docking port requirements have been met, since the final relative position and velocity are within the specified limits. The thrust profile[2] shown in Figure 3.3d and the time of flight are both within their respective limits. Therefore, the rendezvous and docking maneuver described in Section 2.1 has been successful.

[2]The control action profile should be further examined with respect to the throttling capabilities of the thrusters and analyzed in a 6DOF environment to determine its suitability for attitude control. The maximum control effort has been evaluated and verified, so at this point these additional investigations are outside the scope of this project.

Figure 3.3c indicates that, according to Tsiolkovsky's equation, the necessary impulse is $\Delta V = 6.49\ m/s$. This value is greater than what is typically observed in the literature; however, its optimality must always be linked to the designated flight time. These maneuvers are usually completed in a few hours, so a fair assessment of the algorithm's fuel efficiency is conducted in Section 4.2. Moreover, it is important to emphasize that the policy was tasked (Section 2.3) with managing unforeseen thrust failures. As depicted in Figure 3.3d, it successfully addresses this problem by learning to utilize half of the maximum thrust.

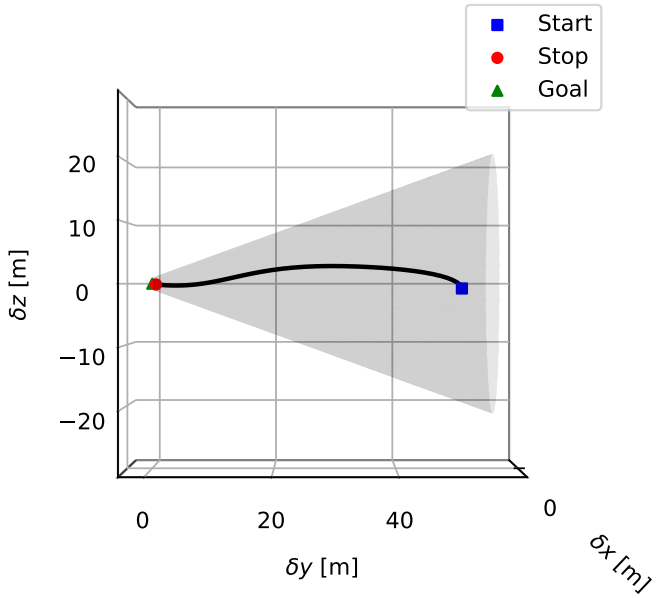
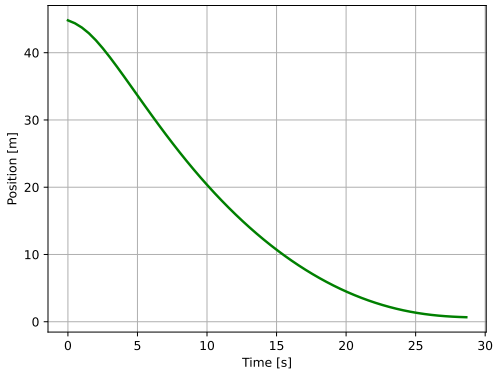
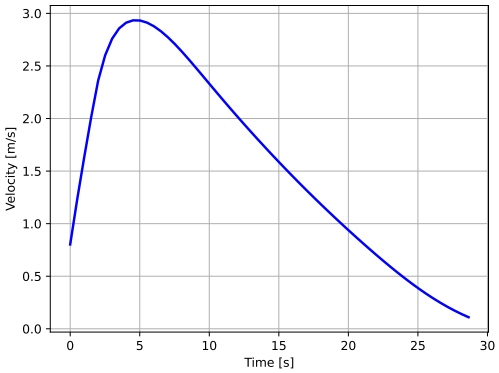


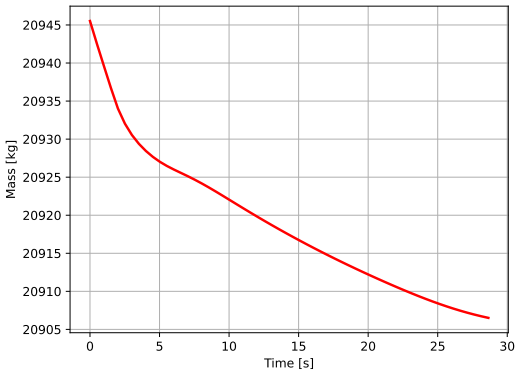
Figure 3.2: Trained LSTM-policy for the 50-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case, deployed in the environment and tested for a single episode. Trajectory inside the approach corridor starting from a randomized initial condition.



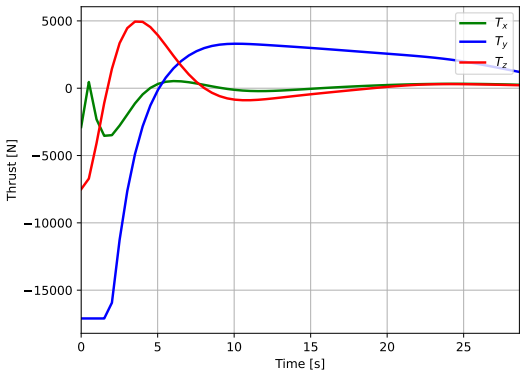
(a) Relative position in time.



(b) Relative velocity in time.



(c) Total mass in time.



(d) Control action in time.

Figure 3.3: Trained LSTM-policy for the 50-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case, deployed in the environment and tested for a single episode. Relative position, relative velocity, mass and control action along the trajectory of Figure 3.2.

The LSTM-policy that has been trained is now evaluated using a set of 500 trajectories, each with a different initial condition based on the uncertainty of the MDPs discussed in Section 2.3. This Monte Carlo campaign has two objectives: to measure the performance of the policy statistically and to evaluate the robustness and computational efficiency as autonomous G&C algorithm for on-board deployment, which is the goal of this work. The trajectories are depicted in Figure 3.4, the CPU-Time for each policy evaluation is displayed in Figure 3.5, and a summary of the results is presented in Table 3.1. The LSTM-policy has been proven to be successful in up to 100% of the cases, meeting the docking port and safety requirements for the entire batch of trajectories. The success rate S_r, which is the fraction of trajectories that comply with all constraints, is determined by the environment *done* signals: a docking signal without a collision signal signifies that the mission has been completed. Despite the wide range of initial conditions, the policy has been able to safely guide the spacecraft to the desired final state with low variance. The mean and standard deviation of fuel consumption and time of flight are also evaluated. Furthermore, the policy evaluation had an average CPU-Time per step of 2.296 ms, which is equivalent to 18.266 ms (55.55 Hz) when taking into account the clock frequency of 250 MHz on a LEON3FT[3] onboard spacecraft processor in an average performance configuration. This value implies that the algorithm can be implemented in Orion's real flight software, which typically runs the GNC blocks[4] at a rate of 40 Hz.

[3] https://www.gaisler.com/index.php/products/boards/gr712rc-board?task=view&id=364
[4] https://ntrs.nasa.gov/api/citations/20190000011/downloads/20190000011.pdf

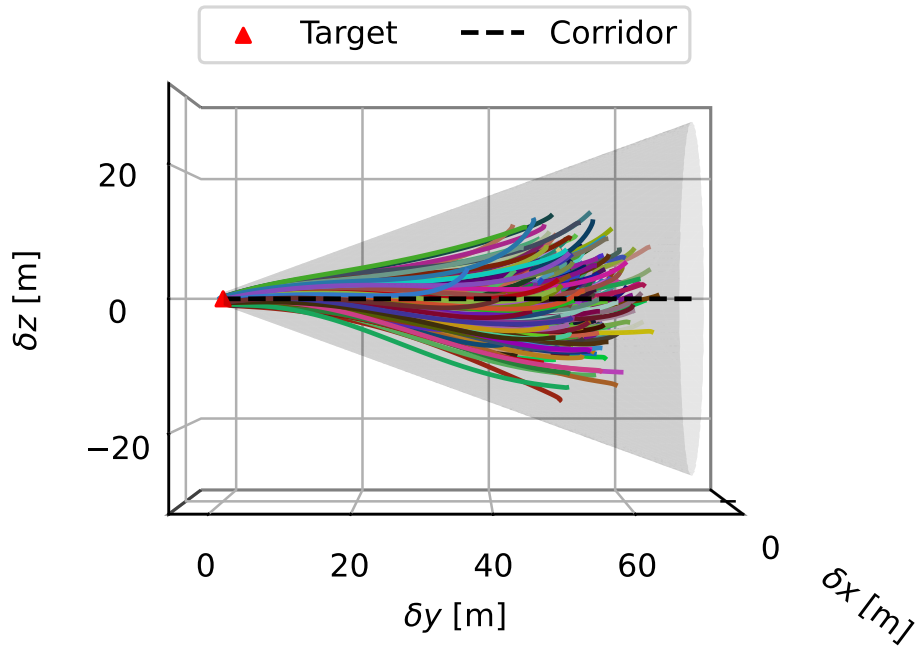


Figure 3.4: Trained LSTM-policy for the 50-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case, deployed in the environment and tested for a batch of episodes. Trajectories inside the approach corridor starting from a randomized initial condition.

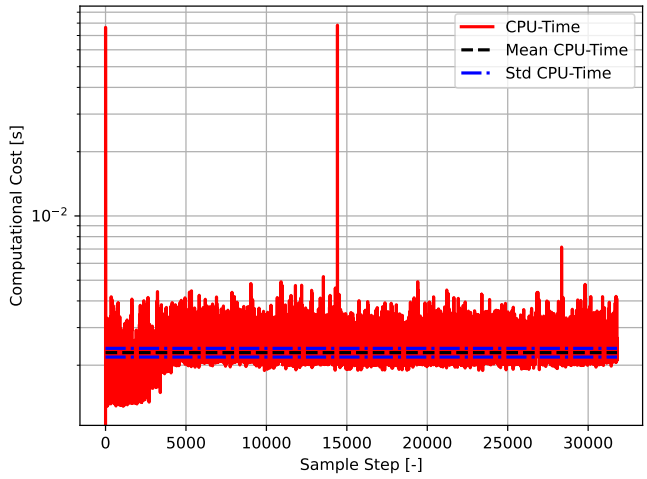


Figure 3.5: Trained LSTM-policy for the 50-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case, deployed in the environment and tested for a batch of episodes. CPU-Time for each policy evaluation step during testing.

Success Rate: $S_r = 100\%$	**Results MC:** $\mu \pm \sigma$
Final Position ρ_f $[m]$	0.552 ± 0.063
Final Velocity $\dot{\rho}_f$ $[m/s]$	0.088 ± 0.003
Fuel Consumption ΔV $[m/s]$	6.350 ± 0.110
Time of Flight ToF $[s]$	32.104 ± 1.267
CPU-Time Step dt_{CPU} $[ms]$	2.296 ± 0.107

Table 3.1: Numerical results summary of the LSTM-policy for the 50-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case testing, as illustrated in Figure 3.4.

As indicated in Subsection 2.3.3, the policy is trained with the inclusion of process noise to make it more reliable. This allows the policy to be trained on a high-fidelity simulator and still perform optimally in the real world even when unmodeled accelerations are present. This enhances the dependability of the solution and decreases mission design time (i.e., the full dynamical model is not required). This statement is demonstrated by testing the trained LSTM-policy on a batch of 500 trajectories in an environment similar to the one described in Subsection 2.3.3, but without process noise and with the addition of the Sun's fourth-body gravitational effect and Solar Radiation Pressure (with $C_r = 0.1$ and $A = 1\ m^2$) accelerations into the chaser dynamics, as defined in Subsection 1.2.1. The results of the Monte Carlo study are shown in Figure 3.5 and Table 3.2, and being the same as those of Figure 3.4 and Table 3.1 demonstrate that the LSTM-policy is also capable of handling unmodeled accelerations.

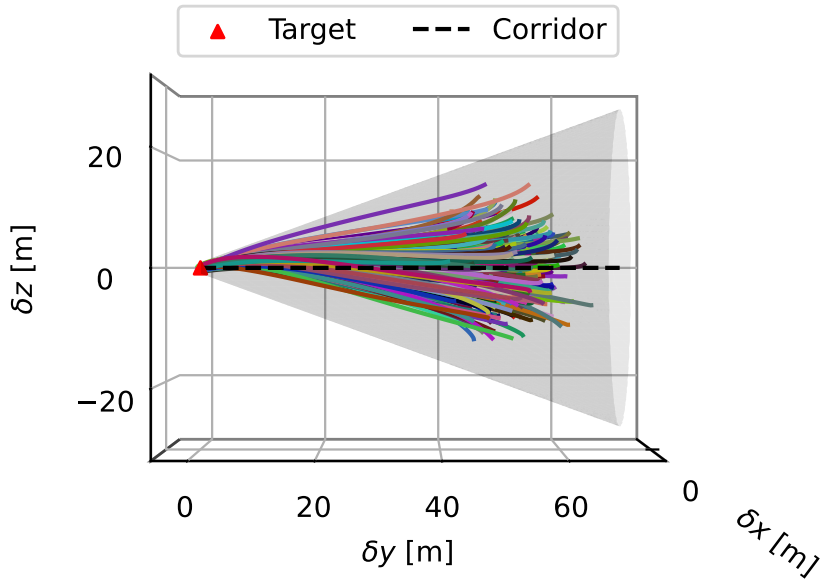


Figure 3.5: Trained LSTM-policy for the 50-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case, deployed in an environment with Solar Radiation Pressure (SRP) and Sun's fourth-body gravity disturbances, and tested for a batch of episodes. Trajectories inside the approach corridor starting from a randomized initial condition.

Success Rate: $S_r = 100\%$	**Results MC:** $\mu \pm \sigma$
Final Position ρ_f $[m]$	0.653 ± 0.082
Final Velocity $\dot{\rho}_f$ $[m/s]$	0.090 ± 0.003
Fuel Consumption ΔV $[m/s]$	6.644 ± 0.254
Time of Flight ToF $[s]$	33.621 ± 0.986

Table 3.2: Numerical results summary of the LSTM-policy for the 50-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case testing with disturbances, as illustrated in Figure 3.5.

This algorithm would be an optimal selection for autonomous proximity operations G&C on a *Southern L_2 9:2 Resonant NRHO* for the Earth-Moon system. It is capable of effectively solving rendezvous and docking maneuvers while guaranteeing safety, computational efficiency, and robustness to uncertainties and unmodeled accelerations. This is evidenced by the two Monte Carlo analyses presented. All in all, this algorithm is highly promising.

3.1.2. Final Approach and Docking: 200-meter Case

The 200-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case is now being studied. The reference frame used is described in Subsection 1.2.2. The absolute target and relative chaser initial conditions, expressed in unitless form, are as follows:

$$\mathbf{x}_0^T = \begin{bmatrix} 1.02206694 \\ -5.25240280 \cdot 10^{-7} \\ -1.82100000 \cdot 10^{-1} \\ -6.71943026 \cdot 10^{-7} \\ -1.03353155 \cdot 10^{-1} \\ 2.55711651 \cdot 10^{-6} \end{bmatrix} \quad \delta\mathbf{x}_0 = \begin{bmatrix} 1.70730097 \cdot 10^{-12} \\ 5.25240280 \cdot 10^{-7} \\ -6.49763576 \cdot 10^{-12} \\ 6.71943026 \cdot 10^{-7} \\ -5.76798331 \cdot 10^{-12} \\ -2.55711651 \cdot 10^{-6} \end{bmatrix} \tag{3.2}$$

The chaser spacecraft is located at the *Southern L_2 9:2 Resonant NRHO* aposelene, while the target is 200 meters away in the clockwise direction of the orbit. The initial mass of

the chaser spacecraft is taken from Table 2.1 and the maximum time of flight (ToF_{max}) is set to 100 s, in order to complete the problem definition described in Section 2.1.

Training

The agent is trained for 7M steps, which is equivalent to 40.5 days in a simulated environment. This process takes 3.2 days and the training curve, the mean discounted cumulative reward of the trajectories roll-outs in relation to the learning step, is shown in Figure 3.6.

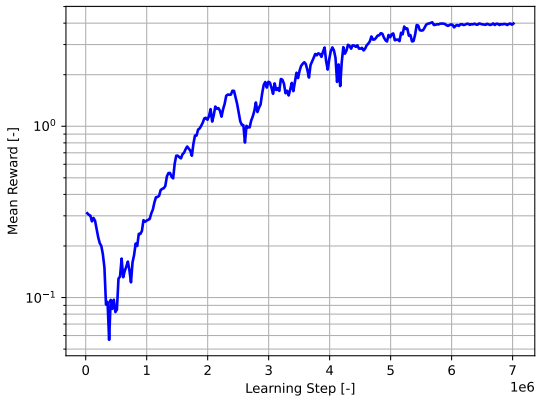


Figure 3.6: Mean discounted cumulative reward of the trajectory roll-outs during the training phase of the LSTM-policy for the 200-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case.

The learning curve shows that the reward gradually increases until it reaches a plateau, which indicates the success of the learning process and the successful resolution of the associated MDP. In Section 4.1, the mean reward achieved, which reflects the effectiveness of the LSTM-policy, will be assessed and compared.
This training was the first to show the presence of catastrophic forgetting [96]. To prevent this from occurring during the learning process, gradient clipping is employed, as discussed in Section 2.5. This enabled the plateau to be reached precisely; however, the phenomenon persisted even after the plateau had been achieved, advancing further with the learning steps. To address this, the training is designed to complete and save the best policy achieved after a certain number of steps without any improvement. This is made possible by the *Stable-Baseline3 (SB3)* callback[5] called "StopTrainingOnNoModelImprovement". To further improve the situation, a systematic hyperparameter search, reward engineering, and the implementation of adaptive learning rate annealing would be necessary. Unfortunately, simpler solutions, such as decreasing the size of neural net-

[5] `https://stable-baselines3.readthedocs.io/en/master/guide/callbacks.html`

works and the learning rate, have not been successful, and further exploration is beyond the scope of this project. However, this training has reached its plateau and the results are satisfactory.

Testing

The LSTM-policy that has been trained is now being tested in the environment, with the same uncertainties as in Section 2.3. The results of the deployment are shown in Figure 3.7 and Figure 3.8, which are based on a single episode. The trajectory, relative position and velocity, mass, and thrust profile over time are shown in these results.

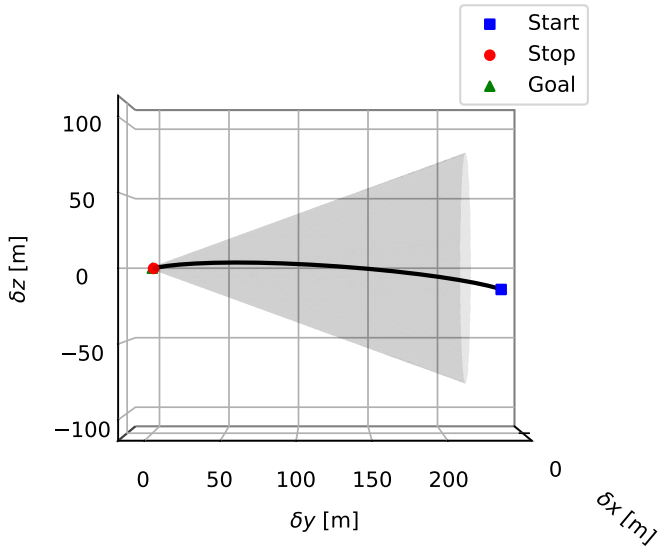
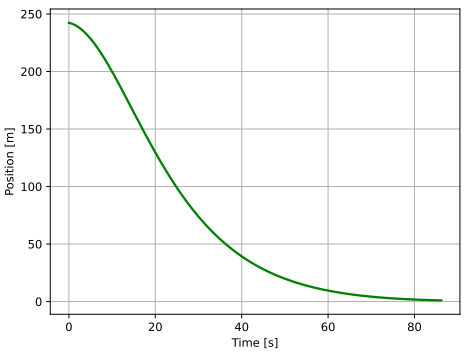
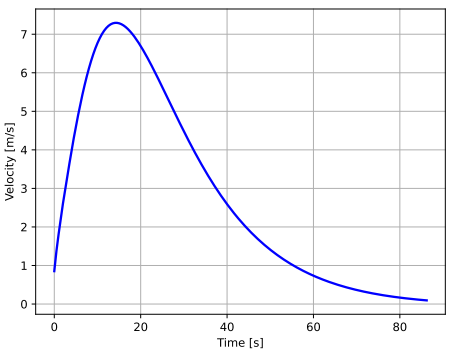


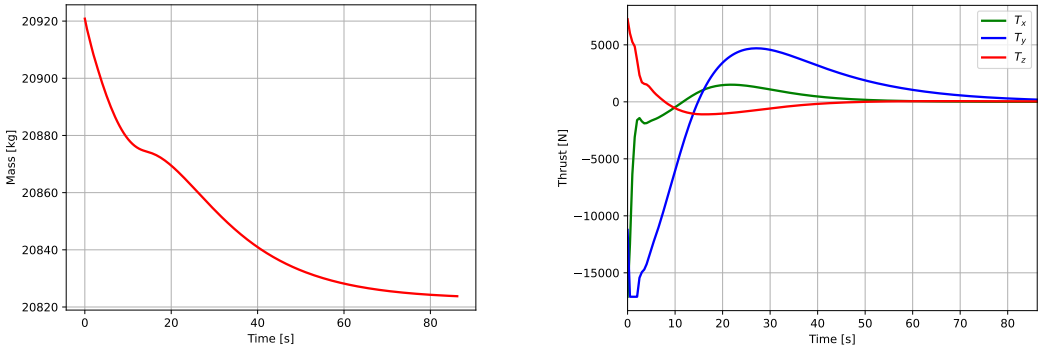
Figure 3.7: Trained LSTM-policy for the 200-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case, deployed in the environment and tested for a single episode. Trajectory inside the approach corridor starting from a randomized initial condition.



(a) Relative position in time.



(b) Relative velocity in time.



(c) Total mass in time. (d) Control action in time.

Figure 3.8: Trained LSTM-policy for the 200-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case, deployed in the environment and tested for a single episode. Relative position, relative velocity, mass and control action along the trajectory of Figure 3.7.

The results of Subsection 3.1.1 are further demonstrated in these graphs, which show that the trained LSTM-policy is again successful in reaching the desired state of the docking port ($\delta\mathbf{x}_f = \left[0.612\ m,\ 0.085\ m/s\right]$) while following safety conditions, maximum thrust, and maximum time of flight. Figure 3.8c in particular displays the mass profile over time, and Tsiolkovsky's equation indicates that a ΔV of 13.113 m/s is needed.
The LSTM-policy that has been trained is now being tested using 500 trajectories, each with a different initial state based on the randomness of the MDPs discussed in Section 2.3. The Monte Carlo campaign trajectories are shown in Figure 3.8, the CPU-Time for each policy evaluation is shown in Figure 3.9, and a summary of the results is given in Table 3.3.

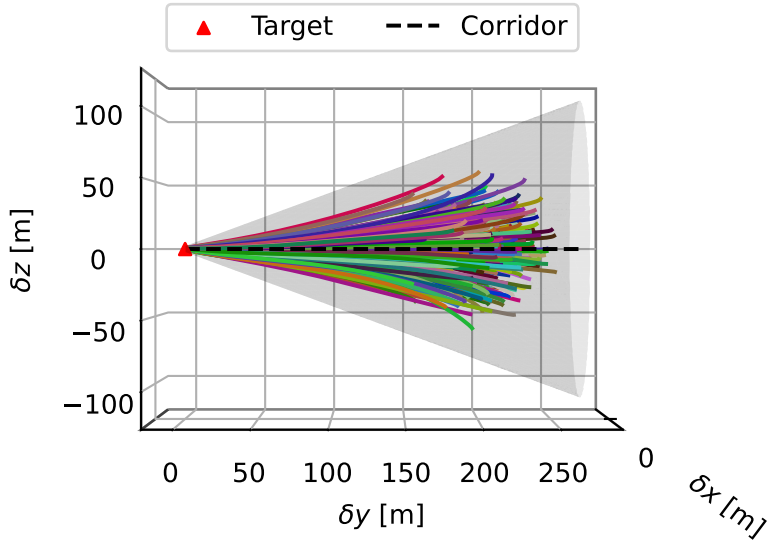


Figure 3.8: Trained LSTM-policy for the 200-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case, deployed in the environment, and tested for a batch of episodes. Trajectories inside the approach corridor starting from a randomized initial condition.

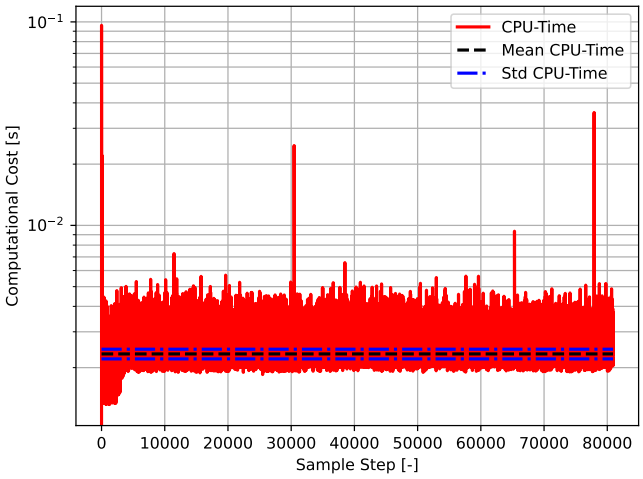


Figure 3.9: Trained LSTM-policy for the 200-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case, deployed in the environment and tested for a batch of episodes. CPU-Time for each policy evaluation step during testing.

Success Rate: $S_r = 100\%$	**Results MC:** $\mu \pm \sigma$
Final Position ρ_f $[m]$	0.561 ± 0.096
Final Velocity $\dot{\rho}_f$ $[m/s]$	0.082 ± 0.004
Fuel Consumption ΔV $[m/s]$	11.981 ± 0.495
Time of Flight ToF $[s]$	82.571 ± 1.571
CPU-Time Step dt_{CPU} $[ms]$	2.334 ± 0.138

Table 3.3: Numerical results summary of the LSTM-policy for the 200-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case testing, as illustrated in Figure 3.8.

The LSTM-policy has been proven to be successful in all scenarios, satisfying all the requirements for the entire set of trajectories. The mean and standard deviation of fuel consumption and time of flight have also been evaluated. Furthermore, the computational efficiency and the capability to execute this algorithm on-board have been verified. The same results were obtained in Subsection 3.1.1.

The LSTM-policy that has been trained is tested on a set of 500 trajectories in a setting similar to the one described in Subsection 2.3.3, but without process noise and with the addition of the Sun's fourth-body gravitational effect and Solar Radiation Pressure (with $C_r = 0.1$ and $A = 1\ m^2$) accelerations into the chaser dynamics, as defined in Subsection 1.2.1. This is to show that the policy is able to withstand unmodeled accelerations during testing by introducing noise during the training phase. The results of the Monte Carlo study are shown in Figure 3.10 and Table 3.4, which are identical to those of Figure 3.8 and Table 3.3, indicating that the LSTM-policy is capable of managing unmodeled accelerations.

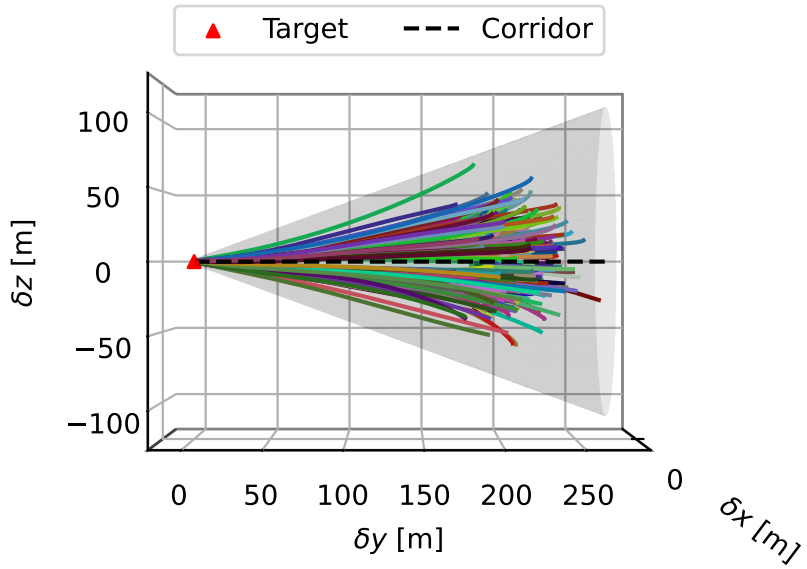


Figure 3.10: Trained LSTM-policy for the 200-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case, deployed in an environment with Solar Radiation Pressure (SRP) and Sun's fourth-body gravity disturbances, and tested for a batch of episodes. Trajectories inside the approach corridor starting from a randomized initial condition.

Success Rate: $S_r = 100\%$	**Results MC:** $\mu \pm \sigma$
Final Position ρ_f $[m]$	0.579 ± 0.084
Final Velocity $\dot{\rho}_f$ $[m/s]$	0.094 ± 0.001
Fuel Consumption ΔV $[m/s]$	12.263 ± 0.524
Time of Flight ToF $[s]$	81.713 ± 0.955

Table 3.4: Numerical results summary of the LSTM-policy for the 200-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case testing with disturbances, as illustrated in Figure 3.10.

The algorithm has also proven its effectiveness in this situation, successfully executing rendezvous and docking maneuvers while providing safety, computational efficiency, and the ability to withstand extensive uncertainties and unmodeled accelerations. This is supported by the two Monte Carlo analyzes presented, which corroborate the findings in Subsection 3.1.1.

4 | Benchmarks

> Nothing in life is to be feared, it is only to be understood. Now is the time to understand more, so that we may fear less.
>
> Marie Curie

This chapter provides a comprehensive overview of the research by performing a series of benchmark analyses. A Multi-Layer Perceptron (MLP) policy is tested and trained to compare the results with those of the Long Short-Term Memory (LSTM) policy. Additionally, an Optimal Control Problem (OCP) pseudospectral direct solution is applied to evaluate optimality, and a Lyapunov's direct method is used to assess system asymptotic stability. This collection of benchmarks offers a detailed assessment of research effectiveness in multiple dimensions.

4.1. Reinforcement Learning: Non-Recurrent Policy

This work seeks to assess whether *Meta-Reinforcement Learning (Meta-RL)* can be used as an autonomous guidance and control algorithm for spacecraft proximity operations. Subsequently, it would be beneficial to determine if Meta-RL has any advantages over more traditional reinforcement learning techniques, as mentioned in Section 1.5.

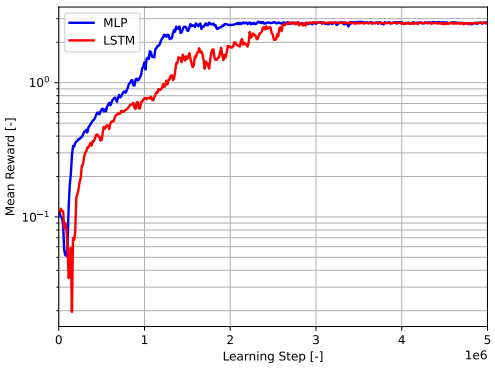
In this section, *Proximal Policy Optimization (PPO)* is used to train a basic *Multi-Layer Perceptron (MLP)* agent. This contrasts with the recurrent network discussed in Chapter 3, which is now replaced by a feedforward network. The width and depth of the agent's neural network[1], the hyperparameters and the environment setup of Sections 2.4, 2.5, and 2.3, respectively, are kept the same to ensure fair comparison. Meta-RL is proposed in the literature as an adaptive guidance and control algorithm due to its agent's internal LSTMs memory. Once trained, the weights and biases of an MLP-policy are fixed and cannot be altered in response to changes in the environment, such as disturbances or failures. An

[1] It is noteworthy that Long Short-Term Memory (LSTM) cells contain six times the number of parameters as a Multi-Layer Perceptron (MLP) cell. Therefore, even if the same neural network architecture is used, the LSTM-based neural networks will take more time to train.

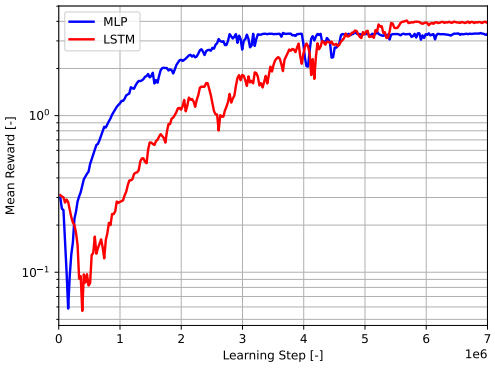
MLP-policy learns a strategy that is generally effective, while an LSTM-policy also learns how to adjust itself in order to optimize its performance for each task in a specific way. Therefore, these assertions will be examined in the following section.

Training

The scenarios being examined are the same as those already discussed in Section 3.1: a 50-meter and a 200-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) cases. Initial conditions and time of flights are kept consistent with the respective scenario. The two MLP-policies are trained with the same number of steps as described in Section 3.1. The mean discounted cumulative rewards of the trajectory roll-outs in relation to the learning step are shown in Figure 4.1.



(a) 50-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case.



(b) 200-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case.

Figure 4.1: Mean discounted cumulative rewards of the trajectory roll-outs during the training phases of the MLP-policies compared to those of the LSTM-policies. The numerically solved study cases are identical to those in Chapter 3.

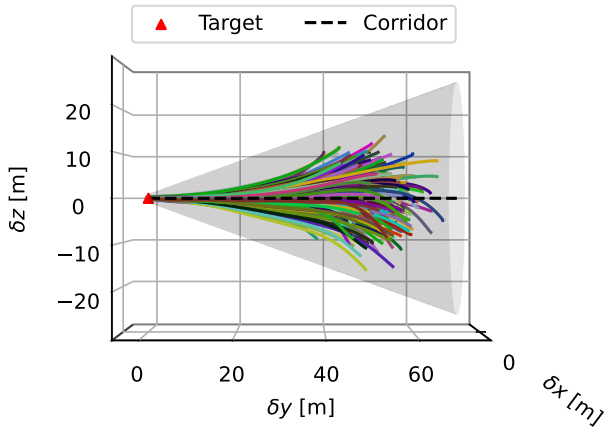
The rewards of MLP-policies increase steadily until they reach a plateau, suggesting that the learning process has been successful. This is analogous to the LSTM-policies discussed in Chapter 3, which always converged.

It is worth noting the distinctions between Long Short-Term Memory (LSTM) networks and Multi-Layer Perceptrons (MLPs) when training, given that LSTMs are a more complex model (with six times more parameters). MLP-policies take less wall-time to train than LSTM-policies with the same number of learning steps, usually about half the time (i.e., 0.9 days for the 50-meter case and 1.8 days for the 200-meter case). MLPs achieve faster convergence in terms of learning steps and exhibit greater stability in both scenarios. The results of this study, as seen in Figure 4.1b, show that LSTMs *can* be more effective in optimizing the policy and consistently obtaining a higher cumulative reward, particularly for more intricate tasks such as the final approach scenario with the farthest range. The reward discussed in Subsection 2.3.2 takes into account both the distance from the axis of the approach corridor and the control effort as dense penalties. MLP solutions in Section 4.1 will be less fuel efficient than the LSTM ones in Section 3.1. However, the cumulative reward of the final approach scenario with the shortest range shown in Figure 4.1a is the same for both MLPs and LSTMs, as LSTMs used less fuel, but the trajectories are, on average, farther away from the axis of the approach corridor. The results of training an RL policy are also dependent on the random seeds used, as it is a stochastic process.

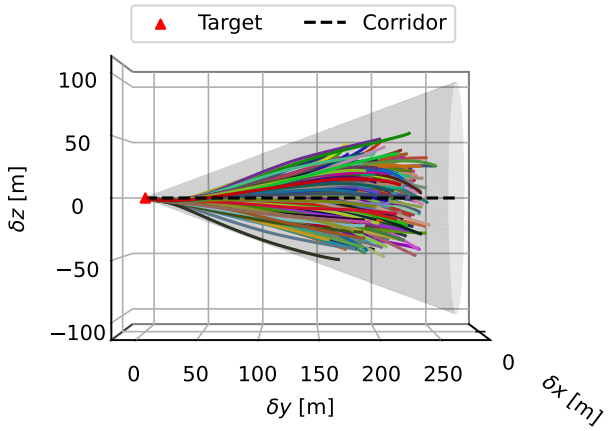
This has been confirmed by Lorenzo Capra and Lavagna [91], where LSTMs achieved a slightly higher or equal cumulative reward at the end of the training process, depending on the scenario, compared to MLPs.

Testing

The MLP-policies that have been trained are now being tested in the environment. To evaluate their performance, a batch of 500 trajectories with different initial conditions is used, based on the uncertainty of the MDPs discussed in Section 2.3. The results of the deployment can be directly demonstrated through Monte Carlo results instead of a single-episode trajectory, as the relative state, mass, and thrust plots in time for the MLP-policies are similar to those in Section 3.1 for the LSTM-policies. Monte Carlo analysis is used to evaluate the accuracy of rendezvous and docking maneuvers of MLP-policies by measuring the success rate S_r, and to acquire statistical data on the final relative state, fuel consumption and time of flight. Trajectories are depicted in Figure 4.2, and a summary of the results is presented in Tables 4.1 and Table 4.2.



(a) 50-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case.



(b) 200-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case.

Figure 4.2: Trained MLP-policies, deployed in the environment, and tested for a batch of episodes. Trajectories within the approach corridor starting from a randomized initial condition. The numerically solved study cases are identical to those in Chapter 3.

Success Rate: $S_r = 100\%$	**Results MC:** $\mu \pm \sigma$
Final Position ρ_f $[m]$	0.349 ± 0.075
Final Velocity $\dot{\rho}_f$ $[m/s]$	0.091 ± 0.002
Fuel Consumption ΔV $[m/s]$	9.930 ± 0.262
Time of Flight ToF $[s]$	23.425 ± 0.438

Table 4.1: Numerical results summary of the MLP-policy for the 50-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case testing, as illustrated in Figure 4.2a.

Success Rate: $S_r = 100\%$	**Results MC:** $\mu \pm \sigma$
Final Position ρ_f $[m]$	0.719 ± 0.051
Final Velocity $\dot{\rho}_f$ $[m/s]$	0.088 ± 0.003
Fuel Consumption ΔV $[m/s]$	18.669 ± 0.537
Time of Flight ToF $[s]$	61.571 ± 1.274

Table 4.2: Numerical results summary of the MLP-policy for the 200-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case testing, as illustrated in Figure 4.2b.

The MLP-policy has been proven to be effective in all scenarios, satisfying the docking port and safety requirements for the entire set of trajectories within the maximum control action and the allocated time of flight. By comparing the results of Section 3.1 with those of the present, it is evident that the success rate of the solutions is the same as the LSTM-policies; however, the MLP-policies have a much higher average fuel consumption. It is evident that the recurrent layers are more successful in obtaining the best results for both scenarios, as demonstrated by the fuel consumption. This is especially true for the 200-meter holding point case, which yielded even higher levels of optimality, as the maximum cumulative reward obtained through learning shows. This is because, as previously mentioned, LSTM-policies can learn how to modify their hidden states during testing to more effectively optimize each individual task in a very "customized" manner. Further comparisons between LSTM-policies and MLP-policies can be seen in Appendix B, which includes an additional restriction on plume impingement compared to MDP described in Section 2.3.

Gaudet and Linares [42] have already determined that both the LSTM and MLP policies could achieve similar terminal relative states, but the LSTMs were able to achieve fuel performance that was superior to that of MLPs. It is evident that recurrent policies are more successful than non-recurrent ones when tackling tasks with a high degree of randomness. Nevertheless, satisfactory results can still be obtained with an MLP-policy, as this work has demonstrated. Federici et al. [53] conducted an investigation into the efficacy of recurrent policies for *Transfer Learning (TL)* and their greater ability to generalize in a collection of six finite-thrust planar rendezvous scenarios. The results showed that the LSTM-policies had a higher mission success rate than the MLP-policies. Further exploration of this topic would be interesting but is beyond the scope of this work. Meta-RL is designed to learn how to handle multiple tasks, but for spacecraft G&C applications, it is more suitable to be used only to improve the robustness of a single scenario, treating tasks as "high parameter randomness" rather than "different scenarios". The application of TL

techniques leads to a decrease in the performance of the single scenario. Therefore, in a real space application, it is recommended to use different policies for different maneuvers to achieve the highest performance in each situation. Furthermore, it would be beneficial to investigate *Hierarchical Reinforcement Learning (HRL)* for this type of applications, as discussed in Section 5.2.

4.2. Optimal Control Problem: Pseudospectral Direct Method

Artificial Neural Networks (ANNs) can provide an approximation of the solution to the Hamilton-Jacobi-Bellman (HJB) equation, thus making them a suboptimal control solution. To assess the effectiveness of the reinforcement learning approach used in this study, a state-of-the-art *Optimal Control Problem (OCP)* solution, such as a pseudospectral direct method [97], is used as reference. Although OCP solutions cannot consider certain features of the RL solutions proposed in this work, such as process noise and actuator malfunctions, they still provide a useful benchmark for optimality.

Computational techniques for OCPs are employed to determine the most appropriate open-loop trajectory of a dynamic system while meeting given restrictions and objectives. Direct methods accomplish this by transforming the continuous-time optimal control problem into a discrete-time form that can be solved using *Non-Linear Programming (NLP)* algorithms. Pseudospectral direct methods approximate the state and control variables over a given period of time by means of a weighted sum of polynomial basis functions. These methods are desirable because of their exponential rate of convergence and their ability to provide high accuracy and efficiently handle constraints and complex dynamics, even with relatively coarse grids. Various software packages can be used to solve OCPs with pseudospectral methods, such as GPOPS-II[2] and PROPT[3]. This work makes use of an open-source Python library, *mpopt*[4], which is suitable for multi-phase problems and different types of pseudospectral collocation methods, including Legendre-Gauss-Radau (LGR), Legendre-Gauss-Lobatto (LGL) and Chebyshev-Gauss-Lobatto (CGL) polynomials. Equations that capture the core components of pseudospectral direct methods for OCPs are presented in the following:

1. **State and Control Approximation**:

   Consider a general state variable, $x(t)$, and a control input, $u(t)$. Approximate them

[2] `https://www.gpops2.com/index.html`
[3] `http://tomdyn.com/`
[4] `https://pypi.org/project/mpopt/`

by using polynomial interpolation in the time period $[t_k, t_{k+1}]$:

$$x(t) \approx x_k(\tau) = \sum_{j=0}^{N} x_{kj}\phi_j(\tau) \qquad u(t) \approx u_k(\tau) = \sum_{j=0}^{N} u_{kj}\phi_j(\tau) \tag{4.1}$$

At a certain moment τ within the period $[t_k, t_{k+1}]$, the state is estimated by $x_k(\tau)$ and the control input by $u_k(\tau)$. This is achieved by using N coefficients, x_{kj} and u_{kj}, and the basis functions, $\phi_j(\tau)$, evaluated at τ.

2. **Collocation Condition**:

   The dynamics of the system can be enforced at specific collocation points within the interval. For example, using a single collocation point (that is, the root of a first-order basis function) at the midpoint of the interval $[t_k, t_{k+1}]$:

$$f\left(t_k + \frac{h}{2}\right) - \dot{x}_{k+\frac{1}{2}} = 0 \tag{4.2}$$

   The dynamics of the system, usually represented by $\dot{x}$, can be expressed as f. The gap between t_{k+1} and t_k is the length of the interval, denoted by h. The time derivative of the state at the collocation point is $\dot{x}_{k+\frac{1}{2}}$.

3. **Path Constraints and Boundary Conditions**:

   Path constraints can be included in the form of algebraic equations that are enforced at the collocation points. For example:

$$g(x_k(\tau), u_k(\tau)) = 0 \tag{4.3}$$

   The path constraint g is contingent on the state $x_k(\tau)$ and control $u_k(\tau)$. Boundary conditions are treated in a similar way.

4. **Optimization Problem**:

   The OCP is an attempt to minimize the cost function while satisfying the dynamics and constraints. The cost function should be discretized in the following manner:

$$\min_{u_{kj}} J = \sum_{k=0}^{N_t-1} \int_{t_k}^{t_{k+1}} L(x_k(\tau), u_k(\tau))d\tau \tag{4.4}$$

The dynamics, path constraints, boundary conditions, and any other restrictions must all be satisfied at the collocation points.

5. **Solver**:

   Nonlinear programming algorithms, for example *Sequential Quadratic Programming (SQP)*, are often used to solve the optimization problem and determine the best control coefficients u_{kj} and state coefficients x_{kj}.

In order to fully specify the algorithm that has been put into practice, it is essential to pick the number of time intervals, the type of polynomial, and its order. In this study, a single time interval of length $h = ToF$ is taken into account and Legendre-Gass-Radau (LGR) polynomials of order 4^{th} are used. The solved Optimal Control Problem (OCP)is expressed in Equation 2.3. Ensuring convergence relies on key numerical methods, such as variable scaling, but delving into their specifics is beyond the scope here.

A Monte Carlo technique is used to compare the outcomes of Chapter 3, which were acquired through Meta-Reinforcement Learning (Meta-RL), with the OCP results. This process involves randomly selecting initial conditions for the relative chaser dynamics and its mass from three broad normal distributions before solving the problem. The means of these distributions are taken as the initial conditions and have standard deviations of $\sigma_{\boldsymbol{\rho}_0} = 0.1\rho_0$, $\sigma_{\dot{\boldsymbol{\rho}}_0} = 0.5\ m/s$, and $\sigma_{m_0} = 100\ kg$. To ensure that the comparison between the LSTM-policies, the MLP-policies, and the OCPs is as precise as possible, the initial guess of time of flight in the OCPs is set to the midpoint between the minimum and maximum values obtained from all the Monte Carlo campaigns (as described in Sections 3.1 and 4.1). The results of a batch of 500 trajectories for both scenarios taken into account in Chapter 3 are shown in Figure 4.3, Figure 4.4 and Table 4.3. The proposed solutions should not be considered as the flawless answer to the OCP, but rather as a benchmark of optimality, so S_r is not displayed. This OCP is complex, with 13 states, making it impossible to accurately tune it within a Monte Carlo analysis. The IPOPT non-linear solver, with a tolerance of 10^{-8}, has been able to successfully solve a large number of trajectories, though not all of them met all of the requirements.

The results in Table 4.3 show that the LSTM-policies from Section 3.1 can generate a solution that is almost optimal, as indicated by the nearly identical average fuel consumption in the results of the Monte Carlo campaign. MDPs provide a closed-loop solution and can take into account stochastic effects such as actuation failures, whereas OCPs are open-loop and cannot. Despite the increased complexity, the LSTM-policies achieved very high optimality, while the MLP-policies did not. It is also noteworthy that the OCP solutions time of flight converged very close to that of the LSTM-policies.

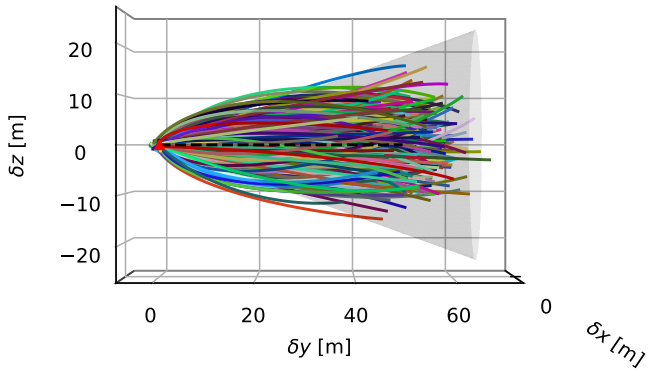


Figure 4.3: Optimal Control Problem (OCP) pseudospectral direct method for the 50-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case, applied to a batch of episodes. Trajectories inside the approach corridor starting from a randomized initial condition.

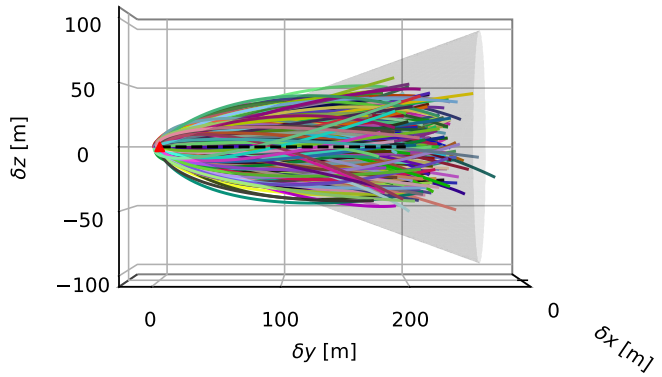


Figure 4.4: Optimal Control Problem (OCP) pseudospectral direct method for the 200-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case, applied to a batch of episodes. Trajectories inside the approach corridor starting from a randomized initial condition.

Results MC: $\mu \pm \sigma$	**50-meter Case**	**200-meter Case**
Fuel Consumption ΔV $[m/s]$	5.763 ± 0.684	11.153 ± 0.772
Time of Flight ToF $[s]$	29.024 ± 1.654	76.619 ± 2.155

Table 4.3: Numerical results summary of the Optimal Control Problem (OCP) pseudospectral direct method for the 50-meter and 200-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) cases, as illustrated in Figures 4.3 and 4.4.

4.3. Stability Analysis: Lyapunov's Direct Method

A G&C algorithm should ensure the asymptotic stability of the controlled system in order to achieve mission success, avoiding any erratic behavior and guaranteeing predictable vehicle performance, while also avoiding potential accidents or deviations from the intended trajectory.

There are different methodologies to assess the asymptotic stability of a closed-loop controller. These include the *Lyapunov's Indirect Method*, which assesses the negativity of the real part of Jacobian eigenvalues, and the *Eigenvalue Analysis Method*, which gauges the bounding of State Transition Matrix (STM) eigenvalues across the trajectory [57]. Additionally, there are a few analytical methods related to the convergence of the Reinforcement Learning (RL) algorithm. If the RL algorithm ensures local optimal convergence, the policy is at least suboptimal, and this implies stability. An RL policy uses actions to guide state variables towards the desired result in order to get the highest possible reward, similar to the Linear Quadratic Regulator (LQR) closed-loop controller, which adjusts the control output to move the state to minimize an objective function. Optimization of a criterion bounds the states along the optimal trajectory, making the policy inherently stabilizing. The proof of the convergence of policy gradient algorithms with function approximators to local optima is outlined in Appendix A from Sutton et al. [84]. This work implements Recurrent PPO, which is an example of such an algorithm.

In this section, the *Lyapunov's Direct Method* [64] is used to demonstrate asymptotic stability, as defined in Theorem 4.1, from a non-linear equations of motion perspective. This approach is based on Theorem 4.2 and is advantageous as it allows one to evaluate the stability of the controlled system without linearizing around an equilibrium solution. It also provides a global stability analysis and is a theoretical proof. This method is usually employed to design Lyapunov-based controllers, but in this case it is used afterward and numerically to assess the stability of the LSTM-policies.

Theorem 4.1 (Lyapunov's Stability)**.** *Consider an autonomous non-linear dynamics system described by:*

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}), \ \mathbf{f}(\mathbf{x}^*) = 0 \tag{4.5}$$

Where $\mathbf{x}^*$ *is an isolated equilibrium point. Then the equilibrium point is said to be:*

1. ***Stable****: for* $\forall \varepsilon > 0$*, there* $\exists \partial > 0$ *such that if* $\left\|\mathbf{x}(0) - \mathbf{x}^*\right\| < \partial$ *then* $\left\|\mathbf{x}(t) - \mathbf{x}^*\right\| < \varepsilon$ $\forall t > 0$*;*

2. ***Asymptotically Stable**: the equilibrium point $\mathbf{x}^*$ is stable and $\left\|\mathbf{x}(t) - \mathbf{x}^*\right\| \to 0$ as $t \to \infty$.*

The definitions describe a local stability in the vicinity of the equilibrium point; if the region ∂ is unbounded, the definitions describe a global stability.

Theorem 4.2 (Lyapunov's Second Stability Theorem)**.** *Consider an autonomous nonlinear dynamics system described by:*

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}), \ \mathbf{f}(\mathbf{x}^*) = 0 \tag{4.6}$$

Where $\mathbf{x}^*$ is an isolated equilibrium point. If there exists in some finite neighborhood D of the equilibrium point $\mathbf{x}^*$, a scalar function $V(\mathbf{x})$ with a continuous first partial derivative with respect to $\mathbf{x}$ such that the following conditions hold:

$$\begin{aligned} &(i) \quad V(\mathbf{x}) > 0, \ \forall \mathbf{x} \neq \mathbf{x}^* \in D \wedge V(\mathbf{x}^*) = 0 \\ &(ii) \quad \dot{V}(\mathbf{x}) < 0, \ \forall \mathbf{x} \neq \mathbf{x}^* \in D \wedge \dot{V}(\mathbf{x}^*) \leq 0 \end{aligned} \tag{4.7}$$

*Then the system is said to be **asymptotically stable**. If D includes all possible states, the system is said to be **globally asymptotically stable**. If $\dot{V}(\mathbf{x}) \leq 0 \ \ \forall \mathbf{x} \in D$ then the system is said to be **stable**.*

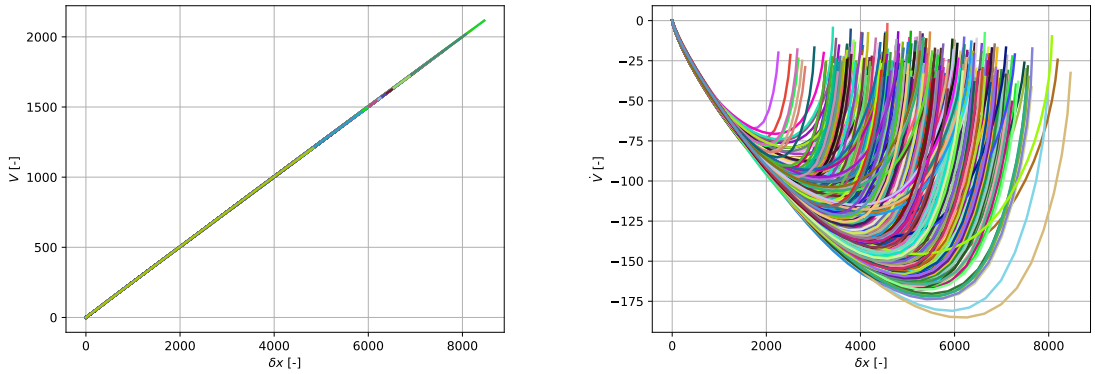
This section considers the *Relative Circular Restricted Three-Body Problem (RCRTBP)* of Equations 1.17 as an autonomous dynamical system. It is assumed that the target absolute state $\mathbf{x}^T$ remains constant throughout the maneuver, since the NRHO has a period of 6.5 days, which is much longer than the duration of any rendezvous maneuver taken into account. Hence, the motion of the target is disregarded and only the equations of motion for the chaser's relative dynamics, expressed as $\delta\dot{\mathbf{x}} = \mathbf{h}(\delta\mathbf{x}, \ \mathbf{u})$ (as opposed to Equation 2.14), are taken into account when defining $V(\mathbf{x})$. With these assumptions, a positive-definite quadratic *Candidate Lyapunov Function* in $\mathbb{R}^6$ can be used as follows:

$$V(\delta\mathbf{x}) = \delta\mathbf{x}^T\mathbf{P}\delta\mathbf{x} \tag{4.8}$$

The diagonal weight matrix taken into account is $\mathbf{P} = diag(\mathbf{0.5}_6)$, and the equilibrium point is $\delta\mathbf{x}^* = \mathbf{0}$, which is the desired outcome of the RVD maneuver and would result in $\mathbf{h}(\delta\mathbf{x}^*, \ \mathbf{u}) = \mathbf{0}$ in the RCRTBP equations. According to Schaub and Junkins [64], the Lyapunov function in Equation 4.8 is of an "energy-error-like" structure.

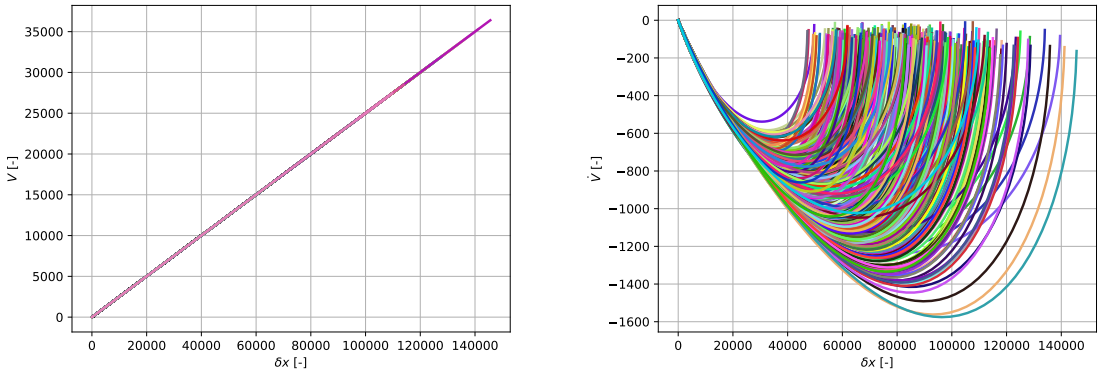
A Monte Carlo technique is used to evaluate stability by randomly selecting the chaser's relative motion and its mass initial conditions from three large normal distributions with

means taken as initial conditions and having standard deviations of $\sigma_{\boldsymbol{\rho}_0} = 0.1\boldsymbol{\rho}_0$, $\sigma_{\dot{\boldsymbol{\rho}}_0} = 0.5\ m/s$, and $\sigma_{m_0} = 100\ kg$. For each of the LSTM-policy study cases discussed in Chapter 3, a set of 500 trajectories is simulated. The trained policies are deployed again in the environment, and the numerical values of $V(\delta\mathbf{x})$ and $\dot{V}(\delta\mathbf{x})$ are determined along the trajectories. The results are shown in Figures 4.5 and 4.6, with the L_2-norm of the relative state vector as the abscissa in each graph.



(a) Candidate Lyapunov Function. (b) Candidate Lyapunov Function time-derivative.

Figure 4.5: Trained LSTM-policy for the 50-meter holding point case, deployed in the environment and tested for a batch of episodes. Candidate Lyapunov Function and its time-derivative of Figure 3.4 with regard to the distance from the equilibrium point.



(a) Candidate Lyapunov Function. (b) Candidate Lyapunov Function time-derivative.

Figure 4.6: Trained LSTM-policy for the 200-meter holding point case, deployed in the environment and tested for a batch of episodes. Candidate Lyapunov Function and its time-derivative of Figure 3.4 with regard to the distance from the equilibrium point.

The LSTM-policies developed in Chapter 3 demonstrate asymptotic stability, as evidenced by the discovery of a Candidate Lyapunov Function that meets the criteria outlined in

Theorem 4.2. Moreover, the Monte Carlo experiments conducted in Chapter 3 could also potentially indicate the asymptotic stability of the controlled systems. This inference arises from the systems' capability to withstand diverse uncertainties without resulting in catastrophic consequences. Asymptotic stability is required to ensure the system's dependability and predictability, and from a spacecraft perspective, it ensures that relative positions and velocities will always converge to the desired configuration within a finite flight time.

5 | Conclusions and Future Work

> I have learned to use the word 'impossible' with the greatest caution.
>
> Wernher von Braun

This work presents a *Meta-Reinforcement Learning (Meta-RL)* algorithm, which is an LSTM-based agent trained by a recurrent version of the actor-critic gradient-based method *Proximal Policy Optimization (PPO)*. This algorithm has been demonstrated to be able to autonomously guide and control a spacecraft during rendezvous and docking missions in the cislunar space. It is a viable option for a real space application, as it has been proven to be robust and computationally efficient for on-board implementation. The findings of this research and potential future works are summarized in this chapter.

5.1. Conclusions

1. **To what extent can a Meta-Reinforcement Learning algorithm be used as an autonomous spacecraft proximity operations guidance and control algorithm in the cislunar space? Does it meet safety requirements while being computationally efficient for on-board execution?**

   This work demonstrates that the *Meta-Reinforcement Learning (Meta-RL)* algorithm is capable of accurately resolving the Markov Decision Process (MDP) posed. The algorithm's efficacy is tested in two spacecraft's final approach and docking scenarios on a *Southern L_2 9:2 Resonant Near-Rectilinear Halo Orbit (NRHO)* of the Earth-Moon system. Despite a great deal of uncertainty in initial conditions, process noise, and actuator malfunctions, all objectives have been achieved. The learning curves show that the training phases were successful as they reached the expected plateau. During the testing phase, the same agents were tested through a Monte Carlo campaign, which revealed that trained policies were able to meet docking port requirements, avoid collisions with the approach corridor, and adhere to the maximum control effort and time of flight in all cases. As a result, the Meta-RL algorithm has proven its capability to effectively learn a guidance and control

policy tailored for rendezvous and docking maneuvers within cislunar space.

To achieve autonomy for spacecraft guidance and control, a closed-loop controller should be developed that is reliable, computationally efficient, and nearly optimal. The proposed Meta-RL policy, a closed-loop controller that approximates the solution of the *Hamilton-Jacobi-Bellman (HJB)* equation, can be considered as a potential solution for autonomous G&C applications. This policy has been evaluated through Monte Carlo simulations not only in the training environment but also in an enhanced one with unmodeled dynamics. It has been determined to be reliable even in the latter situation. In addition, its computational efficiency has been proven to be suitable for on-board processor execution. Finally, its optimality has been confirmed by comparing it with a state-of-the-art Optimal Control Problem (OCP) pseudospectral direct solution, the results of which are reported in Table 5.1.

2. **What are the benefits of using Long Short-Term Memories (LSTMs) instead of basic Multi-Layer Perceptrons (MLPs) when it comes to a reinforcement learning agent for spacecraft guidance and control applications?**

   Long Short-Term Memory (LSTM) and Multi-Layer Perceptron (MLP) policies have both been able to learn the proposed Markov Decision Problems (MDPs) correctly, as demonstrated by the learning curves, which have reached the expected plateaus. During the testing phase, both policies achieved the highest possible success rate in the Monte Carlo campaigns. This achievement highlights their complete proficiency in executing rendezvous and docking operations while meeting all imposed requirements across the entire range of evaluated trajectories. Despite this, the LSTM-policy demonstrated the ability to acquire greater cumulative rewards during training and significantly higher fuel efficiency during testing (see Table 5.1). Its internal memory enabled it to generate an adaptive guidance and control policy that is better suited to the optimal solution of the problem.

   It is evident that recurrent policies are more successful than non-recurrent ones when tackling tasks that have a lot of randomization in their parameters. However, at this stage satisfactory outcomes can still be achieved with an MLP-policy; thus, more intricate MDPs should be addressed to gain a better comprehension of Meta-RL's potential and its benefits in comparison to traditional reinforcement learning. It should be noted that training LSTMs is more complex than other models, as seen in their slightly higher instability during training and the time it takes them to reach convergence. This is a natural result of the complexity of recurrent layers, but can

be addressed with a comprehensive hyperparameter search and reward engineering. From a spacecraft guidance and control point of view, this is not a major problem as long as an optimal policy is achieved.

Fuel Consumption ΔV $[m/s]$	**50-meter Case**	**200-meter Case**
LSTM-Policy	6.350 ± 0.110	11.981 ± 0.495
MLP-Policy	9.930 ± 0.262	18.669 ± 0.537
OCP Direct Method	5.763 ± 0.684	11.153 ± 0.772

Table 5.1: The fuel consumption of Long Short-Term Memory (LSTM) policy, Multi-Layer Perceptron (MLP) policy, and Optimal Control Problem (OCP) pseudospectral direct method solutions are compared for the 50-meter and 200-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) scenarios.

5.2. Future Work

In the future, potential areas of improvement for this work on Meta-Reinforcement Learning, as well as potential new machine learning solutions for spacecraft guidance and control, could be explored.

- **Meta-Reinforcement Learning (Meta-RL) enhancements for this work**:
  - The proposed Meta-RL solution could be improved by putting more effort into reward engineering with various coefficients and functions, with the aim of further improving the final relative state and fuel consumption. Training could be enhanced by performing a thorough search for the best hyperparameters using Bayesian optimization, and it would be advantageous to examine the state/action spaces using different sets of coordinates and scalings. An effective strategy could involve logarithmically scaling the state space before the agent's neural networks. This adjustment aims to enhance the minimization of the final relative state, making the ANNs more responsive to smaller state values, specifically related to better meeting the docking port requirements. It would also be interesting to compare the proposed Meta-Recurrent Neural Network (Meta-RNN) solution with other Meta-Reinforcement Learning implementations (e.g., MAML, TAML, REPTILE, etc.) to have a benchmark;
  - Problem formulation can be improved by taking into account a more realistic and comprehensive framework. For actual deployment in space, it is necessary

to model the full 6-Degree-Of-Freedom (6DOF) dynamics, incorporate additional constraints (such as thruster throttle capabilities, plume impingement, maximum angular velocity, reaction wheel saturation, and so on), and implement a full ephemeris model to achieve the most accurate dynamics possible.

- **Other Machine Learning (ML) solutions for spacecraft Guidance and Control (G&C)**:
  - Space missions often have long-term goals, multiple tasks, complex maneuvers, and safety regulations during each phase. This work focuses directly on the final approach and docking maneuvers, but before reaching this point, a rendezvous and docking mission has many stages (such as orbital insertion, targeting different holding points, and possible fly-arounds to modify the approach corridor). A potential machine learning framework to design the whole mission could be *Hierarchical Reinforcement Learning (HRL)*[98]. Introduces a multilayered approach to decision making, each layer responsible for a distinct element of the problem. At the highest level, a meta-controller is responsible for selecting subgoals and deciding which lower-level controllers to use. The lower levels are responsible for performing subtasks and establishing policies that dictate how the agent should act in different states or situations. The spacecraft would be able to make decisions at various levels of abstraction. At a higher level, it can make general decisions, such as selecting between different mission stages (e.g., orbit insertion, phasing, rendezvous, etc.). At a lower level, it can make more precise decisions related to particular maneuvers or trajectory changes. The mission is divided into subtasks, each with its own reinforcement learning agent, and helps in planning and execution. Additionally, HRL can assess risks at different levels of the mission and take the necessary steps to reduce them. For instance, if a high-level agent identifies a potential collision, it can activate a lower-level agent to carry out evasive maneuvers;
  - It would be interesting to investigate the possibility of using *Neural Ordinary Differential Equations (Neural ODEs)*, *Physics-Informed Neural Networks (PINNs)* to approximate the Hamilton-Jacobi-Bellman (HJB) equations, and *AI-aided traditional methods* (e.g., Optimal Control Problem indirect solutions Warm-Started with Neural Networks) for spacecraft autonomous G&C applications. These techniques are proposed as an alternative to the current Meta-Reinforcement Learning (Meta-RL) solution of this work, and are intended to be used for a single phase of the mission.

Bibliography

[1] Wigbert Fehse. *Automated Rendezvous and Docking of Spacecraft*. Cambridge University Press, 11 2003.

[2] J. R. Burton and W. E. Hayes. Gemini rendezvous. *Journal of Spacecraft and Rockets*, 3:145–147, 1 1966.

[3] Kenneth A. Young and James D. Alexander. Apollo lunar rendezvous. *Journal of Spacecraft and Rockets*, 7:1083–1086, 9 1970.

[4] John L. Goodman. History of space shuttle rendezvous and proximity operations. *Journal of Spacecraft and Rockets*, 43:944–959, 2006.

[5] Timothy E. Rumford. Demonstration of autonomous rendezvous technology project summary. In *Proceedings of SPIE - The International Society for Optical Engineering*, 2003.

[6] Rafail Murtazin and Nikolav Petrov. Short profile for the human spacecraft soyuz-tma rendezvous mission to the iss. *62nd International Astronautical Congress 2011, IAC 2011*, 4:3294–3300, 2011.

[7] Piero Miotto. Designing and validating proximity operations rendezvous and approach trajectories for the cygnus mission. In *AIAA Guidance, Navigation, and Control Conference*. American Institute of Aeronautics and Astronautics, 8 2010.

[8] Emilio De Pasquale. Atv jules verne: a step by step approach for in- orbit demonstration of new rendezvous technologies. In *SpaceOps 2012 Conference*. American Institute of Aeronautics and Astronautics, 6 2012.

[9] Yuichiro Nogawa, Toru Kasai, and Ryosuke Kajiwara. Operational concept evolution from htv3 to htv6 and future improvement up to htv9. In *2018 SpaceOps Conference*. American Institute of Aeronautics and Astronautics, 5 2018.

[10] Yongchun Xie, Changqing Chen, Tao Liu, and Min Wang. *Guidance, Navigation, and Control for Spacecraft Rendezvous and Docking: Theory and Methods*. Springer Singapore, 2021.

[11] Elisa Capello, Fabrizio Dabbene, Giorgio Guglieri, and Elisabetta Punta. "flyable" guidance and control algorithms for orbital rendezvous maneuver. *SICE Journal of Control, Measurement, and System Integration*, 11:14–024, 2018.

[12] Yazhong Luo, Jin Zhang, and Guojin Tang. Survey of orbital dynamics and control of space rendezvous. *Chinese Journal of Aeronautics*, 27:1–11, 2 2014.

[13] Seyyed Mohammad Hosseini Rostami, Arun Kumar Sangaiah, Jin Wang, and Xiaozhu Liu. Obstacle avoidance of mobile robots using modified artificial potential field algorithm. *EURASIP Journal on Wireless Communications and Networking*, 2019:70, 12 2019.

[14] Paul Zarchan. *Tactical and Strategic Missile Guidance, Sixth Edition*. American Institute of Aeronautics and Astronautics, Inc., 3 2012.

[15] Matt Hawkins, Yanning Guo, and Bong Wie. Spacecraft guidance algorithms for asteroid intercept and rendezvous missions. *International Journal of Aeronautical and Space Sciences*, 13:154–169, 6 2012.

[16] Jia Jie, Yao Yu, and Ma Kemao. Continuous optimal terminal proximity guidance algorithm for autonomous rendezvous and docking. *Information Technology Journal*, 12:1011–1017, 2 2013.

[17] D.T. Stansbery and J.R. Cloutier. Position and attitude control of a spacecraft using the state-dependent riccati equation technique. In *Proceedings of the 2000 American Control Conference*, volume 3, pages 1867–1871. IEEE, 2000.

[18] Martina Mammarella, Elisa Capello, and Giorgio Guglieri. A comprehensive analysis of guidance and control algorithms for orbital rendezvous maneuvers. In *AIAA/AAS Astrodynamics Specialist Conference*. American Institute of Aeronautics and Astronautics Inc, AIAA, 2016.

[19] Martina Mammarella, Elisa Capello, Hyeongjun Park, Giorgio Guglieri, and Marcello Romano. Tube-based robust model predictive control for spacecraft proximity operations in the presence of persistent disturbance. *Aerospace Science and Technology*, 77:585–594, 6 2018.

[20] Qi Li, Jianping Yuan, and Huan Wang. Sliding mode control for autonomous spacecraft rendezvous with collision avoidance. *Acta Astronautica*, 151:743–751, 10 2018.

[21] Yunlong Song, Angel Romero, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Reaching the limit in autonomous racing: Optimal control versus reinforcement learning. *Science Robotics*, 2023.

[22] David E. Lee. Gateway destination orbit model: A continuous 15 year nrho reference trajectory. Technical report, NASA Johnson Space Center, 2019.

[23] Ryan Whitley and Roland Martinez. Options for staging orbits in cis-lunar space. Technical report, NASA, 2015.

[24] Francesco Colombi, Andrea Colagrossi, and Michèle Lavagna. Characterisation of 6dof natural and controlled relative dynamics in cislunar space. *Acta Astronautica*, 196:369–379, 7 2022.

[25] Stéphanie Lizy-Destrez, Bastien Le Bihan, Antonino Campolo, and Sara Manglativi. Safety analysis for near rectilinear orbit close approach rendezvous in the circular restricted three-body problem. In *68 th International Astronautical Congress (IAC), Adelaide*, pages 25–29, 2017.

[26] Giordana Bucchioni and Mario Innocenti. Rendezvous in cis-lunar space near rectilinear halo orbit: Dynamics and control issues. *Aerospace*, 8, 2021.

[27] Giordana Bucchioni, Matteo De Benedetti, Fabio D'Onofrio, and Mario Innocenti. Fully safe rendezvous strategy in cis-lunar space: Passive and active collision avoidance. *Journal of the Astronautical Sciences*, 69:1319–1346, 10 2022.

[28] Andrea Colagrossi Lorenzo Bucci and Michèle Lavagna. Rendezvous in lunar near rectilinear halo orbits. *Advances in Astronautics Science and Technology*, 1:39–43, 9 2018.

[29] Harry A. Pierson and Michael S. Gashler. Deep learning in robotics: a review of recent research. *Advanced Robotics*, 31:821–835, 8 2017.

[30] B. Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthil Yogamani, and Patrick Perez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23: 4909–4926, 6 2022.

[31] Dario Izzo, Marcus Märtens, and Binfeng Pan. A survey on artificial intelligence trends in spacecraft guidance dynamics and control. *Astrodynamics*, 3:287–299, 12 2019.

[32] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1 1989.

[33] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction Second edition, in progress*. MIT Press, 2019.

[34] Nicholas B. Lafarge, Daniel Miller, Kathleen C. Howell, and Richard Linares. Guidance for closed-loop transfers using reinforcement learning with application to libration point orbits. In *AIAA Scitech 2020 Forum*, volume 1 PartF. American Institute of Aeronautics and Astronautics Inc, AIAA, 2020.

[35] Lorenzo Federici and Alessandro Zavoli. Robust design of interplanetary trajectories under severe uncertainty via meta-reinforcement learning. In *73rd International Astronautical Congress (IAC), Paris*, 2022.

[36] Stefano Bonasera, Christopher John Sullivan, Natasha Bosanac, Jay W. Mcmahon, Ian Elliott, and Nisar Ahmed. *Designing Impulsive Station-Keeping Maneuvers near a Sun-Earth L2 Halo Orbit via Reinforcement Learning*. PhD thesis, University of Colorado Boulder, 2021.

[37] Gaudet Brian and Furfaro Roberto. Adaptive pinpoint and fuel efficient mars landing using reinforcement learning. *IEEE/CAA Journal of Automatica Sinica*, 1:397–411, 10 2014.

[38] Stefan Willis, Dario Izzo, and Daniel Hennes. Reinforcement learning for spacecraft maneuvering near small bodies. In *AAS/AIAA Space Flight Mechanics Meeting, Napa, CA*, 2016.

[39] Hao Yuan and Dongxu Li. Deep reinforcement learning for rendezvous guidance with enhanced angles-only observability. *Aerospace Science and Technology*, 129, 10 2022.

[40] Lorenzo Federici, Boris Benedikter, and Alessandro Zavoli. Deep learning techniques for autonomous spacecraft guidance during proximity operations. *Journal of Spacecraft and Rockets*, 58:1774–1785, 2021.

[41] Kirk Hovell and Steve Ulrich. Deep reinforcement learning for spacecraft proximity operations guidance. *Journal of Spacecraft and Rockets*, 58:254–264, 2021.

[42] Brian Gaudet and Richard Linares. Adaptive guidance with reinforcement meta-learning. *ArXiv*, 1 2019. URL `https://arxiv.org/abs/1901.04473`.

[43] Andrea Scorsoglio, Roberto Furfaro, Richard Linares, and Mauro Massari. Relative motion guidance for near-rectilinear lunar orbits with path constraints via actor-critic reinforcement learning. *Advances in Space Research*, 71:316–335, 1 2023.

[44] Zichao Liu, Jiang Wang, Shaoming He, Hyo-Sang Shin, and Antonios Tsourdos. Learning prediction-correction guidance for impact time control. *Aerospace Science and Technology*, 119:107187, 12 2021.

[45] Andrea Brandonisio, Michèle Lavagna, and Davide Guzzetti. Reinforcement learning for uncooperative space objects smart imaging path-planning. *The Journal of the Astronautical Sciences*, 68:1145–1169, 12 2021.

[46] Margherita Piccinin, Paolo Lunghi, and Michèle Lavagna. Deep reinforcement learning-based policy for autonomous imaging planning of small celestial bodies mapping. *Aerospace Science and Technology*, 120:107224, 1 2022.

[47] Sepp Hochreiter, A. Steven Younger, and Peter R. Conwell. Learning to learn using gradient descent. In *Lecture Notes in Computer Science*, volume 2130, pages 87–94. Springer Verlag, 2001.

[48] Jane X. Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z. Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *ArXiv*, 11 2016. URL `https://arxiv.org/abs/1611.05763`.

[49] Nicolas Schweighofer and Kenji Doya. Meta-learning in reinforcement learning. *Neural Networks*, 16(1):5–9, 2003.

[50] Brian Gaudet, Richard Linares, and Roberto Furfaro. Deep reinforcement learning for six degree-of-freedom planetary landing. *Advances in Space Research*, 65:1723–1741, 4 2020.

[51] Brian Gaudet, Richard Linares, and Roberto Furfaro. Adaptive guidance and integrated navigation with reinforcement meta-learning. *Acta Astronautica*, 169:180–190, 4 2020.

[52] Gaetano Calabrò. Adaptive guidance via meta-reinforcement learning: Arpod for an under-actuated cubesat. Master's thesis, Politecnico Di Milano, 2022.

[53] Lorenzo Federici, Andrea Scorsoglio, Alessandro Zavoli, and Roberto Furfaro. Meta-reinforcement learning for adaptive spacecraft guidance during finite-thrust rendezvous missions. *Acta Astronautica*, 201:129–141, 12 2022.

[54] Brian Gaudet, Richard Linares, and Roberto Furfaro. Six degree-of-freedom hovering over an asteroid with unknown environmental dynamics via reinforcement learning. In *AIAA Scitech 2020 Forum*. American Institute of Aeronautics and Astronautics, 1 2020.

[55] Brian Gaudet, Richard Linares, and Roberto Furfaro. Terminal adaptive guidance via reinforcement meta-learning: Applications to autonomous asteroid close-proximity operations. *Acta Astronautica*, 171:1–13, 6 2020.

[56] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv*, 7 2017. URL `https://arxiv.org/abs/1707.06347`.

[57] Andrea Scorsoglio, Roberto Furfaro, Richard Linares, and Mauro Massari. Actor-critic reinforcement learning approach to relative motion guidance in near-rectilinear orbit space. Master's thesis, Politecnico Di Milano, 2019.

[58] Charles E. Oestreich, Richard Linares, and Ravi Gondhalekar. Autonomous six-degree-of-freedom spacecraft docking maneuvers via reinforcement learning. *ArXiv*, 2020. URL `https://arxiv.org/abs/2008.03215`.

[59] Jacob Broida and Richard Linares. Spacecraft rendezvous guidance in cluttered environments via reinforcement learning. *29th AAS/AIAA Space Flight Mechanics Meeting*, 2019.

[60] Christopher D'souza, F. Chad Hanak, Pete Spehar, Fred D. Clark, and Mark Charles Jackson. Orion rendezvous, proximity operations, and docking design and analysis. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, 2007.

[61] James M. Free, Kathryn L. Lueders, Sergey Krikalev, David Parker, Gilles Leclerc, and Hara Katsuhiko. International docking system standard (idss) interface definition document (idd). Technical report, NASA, 2022.

[62] William H. Gerstenmaier, David Parker, Gilles Leclerc, and Ryuichiro Shirama. International rendezvous system interoperability standards (irsis). Technical report, NASA, 2019.

[63] Andrea Colagrossi, Michelle Lavagna, James Douglas Biggs, and Pierangelo Masarati. *Absolute and Relative 6DOF Dynamics, Guidance and Control for Large Space Structures in Cislunar Environment*. PhD thesis, Politecnico di Milano, 2019.

[64] Hanspeter Schaub and John L. Junkins. *Analytical Mechanics of Space Systems*. AIAA Education Series, Reston, VA, 4th edition, 2018.

[65] Bruce A. Conway. *Spacecraft Trajectory Optimization*. Cambridge University Press, 8 2010.

[66] Chelsea Thangavelu and Bryn Mawr College. *Transfers between Near Rectilinear Halo Orbits and Low Lunar Orbits*. PhD thesis, University of Colorad Boulder, 2019.

[67] Patrick W. Kenneally. High geometric fidelity solar radiation pressure modeling via graphics processing unit. Master's thesis, University of Colorado Boulder, 2016.

[68] Brian P. Mccarthy and Kathleen C. Howell. Quasi-periodic orbits in the sun-earth-moon bicircular rrestricted four-body problem. In *31st AAS/AIAA Space Flight Mechanics Meeting*, 2021.

[69] Jeffrey S. Parker and Rodney L. Anderson. *Low-Energy Lunar Trajectory Design*. John Wiley and Sons, Inc., 6 2014.

[70] Kenza K. Boudad, Kathleen C. Howell, and Diane C. Davis. Dynamics of synodic resonant near rectilinear halo orbits in the bicircular four-body problem. *Advances in Space Research*, 66:2194–2214, 11 2020.

[71] Giovanni Franzini and Mario Innocenti. Relative motion dynamics in the restricted three-body problem. *Journal of Spacecraft and Rockets*, 56:1322–1337, 2019.

[72] Richard Bellman. The theory of dynamic programming. In *American Mathematical Society, Laramie, Wyoming*, 1954.

[73] Massimo Tipaldi, Raffaele Iervolino, and Paolo Roberto Massenio. Reinforcement learning in spacecraft control applications: Advances, prospects, and challenges. *Annual Reviews in Control*, 54:1–23, 1 2022.

[74] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *ArXiv*, 2 2015.

[75] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *ArXiv*, 6 2015. URL `https://arxiv.org/abs/1506.02438`.

[76] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *ArXiv*, 2 2016. URL `https://arxiv.org/abs/1602.01783`.

[77] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[78] Stefano Silvestrini, Lorenzo Pasqualetto Cassinis, Robert Hinz, David Gonzalez-Arjona, Massimo Tipaldi, Pierluigi Visconti, Filippo Corradino, Vincenzo Pesce, and Andrea Colagrossi. *Modern Spacecraft GNC*, pages 819–981. Elsevier, 2023.

[79] Sepp Hochreiter and J. Urgen Schmidhuber. Long short-term memory. *Neural Computation*, 8, 1997.

[80] Stefano Silvestrini and Michèle Lavagna. Deep learning and artificial neural networks for spacecraft dynamics, navigation and control. *Drones*, 6, 10 2022.

[81] Jacob Beck Risto Vuorio, Evan Zheran Liu, Zheng Xiong, Luisa Zintgraf, Chelsea Finn, and Shimon Whiteson. A survey of meta-reinforcement learning. *ArXiv*, 1 2023. URL `https://arxiv.org/abs/2301.08028`.

[82] Peng Shengguang. Overview of meta-reinforcement learning research. In *Proceedings - 2020 2nd International Conference on Information Technology and Computer Application, ITCA 2020*, pages 54–57. Institute of Electrical and Electronics Engineers Inc., 12 2020.

[83] Safa Alver and Doina Precup. What is going on inside recurrent meta reinforcement learning agents? *ArXiv*, 4 2021. URL `https://arxiv.org/abs/2104.14644`.

[84] Richard S. Sutton, David Mcallester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *ArXiv*, 2015. URL `https://arxiv.org/abs/1706.06643`.

[85] Rousslan Fernand Julien Dossa, Shengyi Huang, Santiago Ontanon, and Takashi Matsubara. An empirical investigation of early stopping optimizations in proximal policy optimization. *IEEE Access*, 9:117981–117992, 2021.

[86] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep policy gradients: A case study on ppo and trpo. *ArXiv*, 5 2020. URL `https://arxiv.org/abs/2005.12729`.

[87] Sebastian Ruder. An overview of gradient descent optimization algorithms. *ArXiv*, 9 2016. URL `https://arxiv.org/abs/1609.04747`.

[88] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference for Learning Representations, San Diego*, 12 2015. URL `https://arxiv.org/abs/1412.6980`.

[89] Callum Wilson and Annalisa Riccardi. Improving the efficiency of reinforcement learning for a spacecraft powered descent with q-learning. *Optimization and Engineering*, 24:223–255, 3 2023.

[90] Nicholas B. LaFarge, Kathleen C. Howell, and David C. Folta. Adaptive closed-loop maneuver planning for low-thrust spacecraft using reinforcement learning. *Acta Astronautica*, 211:142–154, 10 2023.

[91] Andrea Brandonisio Lorenzo Capra and Michèle Lavagna. Network architecture and action space analysis for deep reinforcement learning towards spacecraft autonomous guidance. *Advances in Space Research*, 71:3787–3802, 5 2023.

[92] A. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning*, 1999.

[93] Stefano Bonasera. *Incorporating Machine Learning into Trajectory Design Strategies in Multi-Body Systems*. PhD thesis, University of Colorado Boulder, 2022.

[94] Carl Edward Rasmussen and Christopher KI Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[95] Haşim Sak, Andrew Senior, and Françoise Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *ArXiv*, 2 2014. URL `https://arxiv.org/abs/1402.1128`.

[96] Schak Monika and Gepperth Alexander. A study on catastrophic forgetting in deep lstm networks. *Artificial Neural Networks and Machine Learning - ICANN 2019: Deep Learning*, pages 714–728, 09 2019.

[97] I. Michael Ross and Mark Karpenko. A review of pseudospectral optimal control: From theory to flight. *Annual Reviews in Control*, 36(2):182–197, 2012.

[98] Nicholas B. Lafarge. *Autonomous Guidance for Multi-body Orbit Transfers using Reinforcement Learning*. PhD thesis, Purdue University, 2020.

[99] Alexander A. Soderlund and Sean Phillips. Hybrid systems approach to autonomous rendezvous and docking of an underactuated satellite. *Journal of Guidance, Control, and Dynamics*, pages 1–18, 8 2023.

A | Appendix A: Proof of Proximal Policy Optimization Convergence

Proximal Policy Optimization (PPO) is a reinforcement learning approach that employs policy gradients with the help of function approximators, such as *Artificial Neural Networks (ANNs)*, to optimize the agent. It aims to learn a policy (Actor) characterized by parameters $\boldsymbol{\theta} \in \mathbb{R}^d$ and a state-value function (Critic) described by parameters $\mathbf{w} \in \mathbb{R}^m$. Within the PPO implementation, the computation of an advantage function is essential, which, in the current version, is accomplished using *Generalized Advantage Estimation (GAE)*, as shown in Equation 1.62. This implies that the parameters of the critic network also indirectly estimate the action-value function, $q_{\mathbf{w}}(s,\ a) = \hat{A}^{GAE}_{\mathbf{w}}(s,\ a) + v_{\mathbf{w}}(s)$ (as indicated in Equation 1.39). For further details and definitions, see Section 1.3.

Theorem A.1 (Policy Gradient with Function Approximation)**.** *Let $q_{\mathbf{w}} : \mathbb{S}X\mathbb{A} \to \mathbb{R}$ be the approximation of q_π, with the parameter $\mathbf{w} \in \mathbb{R}^m$. If $q_{\mathbf{w}}$ converges to a local optimum, it satisfies:*

$$\sum_s \mu(s) \sum_a \pi_{\boldsymbol{\theta}}(a \mid s) \left[q_\pi(s,\ a) - q_{\mathbf{w}}(s,\ a)\right] \boldsymbol{\nabla} q_{\mathbf{w}}(s,\ a) = 0 \tag{A.1}$$

If $q_{\mathbf{w}}$ satisfies Equation A.1 and is compatible with the policy parameterization in the sense that:

$$\boldsymbol{\nabla} q_{\mathbf{w}}(s,\ a) = \frac{\boldsymbol{\nabla} \pi_{\boldsymbol{\theta}}(a \mid s)}{\pi_{\boldsymbol{\theta}}(a \mid s)} \tag{A.2}$$

Then:

$$\boldsymbol{\nabla} J(\boldsymbol{\theta}) = \sum_s \mu(s) \sum_a q_w(s,\ a)\, \boldsymbol{\nabla} \pi_{\boldsymbol{\theta}}(a \mid s) \tag{A.3}$$

Theorem A.2 (Policy Iteration with Function Approximation)**.** *Let* $\pi_{\boldsymbol{\theta}}$ *and* $q_{\mathbf{w}}$ *be any differential function approximators for the policy and action-value function, respectively, that satisfy the compatibility condition and for which* $\max_{\boldsymbol{\theta}} |\partial_{\theta_i \theta_j} \pi_{\boldsymbol{\theta}} \left(a|s\right)| < B < \infty$*. Let* $\{\alpha_k\}_{k=0}^{\infty}$ *be any step-size sequence such that* $\lim_{k\to\infty} \alpha_k = 0$ *and* $\sum_k \alpha_k = \infty$*. Then, for any MDP with bounded rewards, the sequence* $\{J\left(\boldsymbol{\theta}_k\right)\}_{k=0}^{\infty}$ *,defined by* $\boldsymbol{\theta}_0$*,* $\boldsymbol{\pi}_{\boldsymbol{\theta}_k}$*, and*

$$\boldsymbol{w}_k = \boldsymbol{w} \quad s.t. \quad \sum_s \mu\left(s\right) \sum_a \pi_{\boldsymbol{\theta}_k}\left(a \mid s\right) \left[q_\pi\left(s,\ a\right) - q_{\mathbf{w}}\left(s,\ a\right)\right] \boldsymbol{\nabla} q_{\mathbf{w}}\left(s,\ a\right) = 0 \tag{A.4}$$

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_k \sum_s \mu\left(s\right) \sum_a \boldsymbol{\nabla} \pi_{\boldsymbol{\theta}_k}\left(a \mid s\right) q_{\mathbf{w}_k}\left(s,\ a\right) \tag{A.5}$$

converges such that $\lim_{k\to\infty} \boldsymbol{\nabla} J\left(\boldsymbol{\theta}_k\right) = 0$*.*

Equation A.2 is verified for compatibility due to the congruence of characteristics between actor and critic neural networks (see Table 2.2). Furthermore, the estimation of the advantage function is carried out using the definition of *Residual Temporal-Difference (Residual TD)* as described in Equation 1.60. The constraint on $\partial_{\theta_i \theta_j} \pi_{\boldsymbol{\theta}} \left(a|s\right)$, although awaiting further theoretical analysis, is supported by numerical validation and the use of gradient clipping. Furthermore, both the MDP reward limits and the step-size criteria are satisfied. The *Recurrent Proximal Policy Optimization (Recurrent PPO)* of this study, as supported by Theorem A.2, ensures convergence to at least a local optimum.

B | Appendix B: Plume Impingement Study Case

In this appendix, a few extra numerical results from the algorithm described in Section 1.5 are presented. These outcomes utilize the Markov Decision Process (MDP) outlined in Section 2.3, now incorporating an additional constraint concerning plume impingement. The chaser spacecraft is assumed to be underactuated, meaning it has thrusters in only one direction of its body frame $\mathcal{C}_T$, the $\hat{\mathbf{c}}_2$-axis, as conceptualized in a 3DOF model. This allows the control action vector to be thought of as the direction of the particulate emissions.

As the chaser performs rendezvous and docking maneuvers, the direction of thrust becomes significant. The thrusters of the chaser should not be pointed directly at the target when in close proximity. The typical representation of thruster emissions' geometry usually involves modeling it as a cone-shaped region surrounding the thrust vector, defined by an angle α. In this analysis, due to the close-range proximity operations considered, this angle and the plume shape are assumed to be constant. Particulate emissions from thrusters impact target spacecraft when:

$$\begin{aligned} &\frac{\mathbf{u}_t \cdot \boldsymbol{\rho}_t}{\|\mathbf{u}_t\|\|\boldsymbol{\rho}_t\|} \leq \cos\left(\frac{\alpha}{2}\right) \\ &\mathbf{u}_t \cdot \boldsymbol{\rho}_t \geq 0 \end{aligned} \tag{B.1}$$

In order for the policy to learn this additional restriction, it should be incorporated into the MDP formulation of Section 2.3. When Equation B.1 is satisfied, the reward function of Subsection 2.3.2 is increased by a penalty of $\phi = -30$ and a signal *done* is sent by the environment to terminate the episode (similar to the approach corridor constraint). For the sake of simplicity, only an episodic reward without reward shaping is considered in this case. The control action $\mathbf{u}_t$ is taken into account in the equations, with $\boldsymbol{\rho}_t$ being the relative position vector. The plume cone is assumed to have an angle of 20°. The docking port requirements are eased since this appendix is only for comparison purposes, so the final relative state shall now meet the criteria of $\rho_f < 2\ m$ and $\dot{\rho}_f < 0.1\ m/s$.

It would be interesting to carry out a thorough examination of a 6DOF underactuated spacecraft, taking into account further restrictions such as the maximum angular speed, reaction wheels saturation, and limitations from sensors and navigation filters (e.g. no bright objects, target within the field of view, maximum radial velocity, etc.). However, this is beyond the scope of this work. The idea for this research direction is inspired by the work of Soderlund and Phillips [99].

Final Approach and Docking: 50-meter Case with Plume Impingement

To investigate the impact of the newly imposed restriction on plume impingement in comparing LSTM-policies and MLP-policies, the scenario involving the 50-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) is now considered. The specified reference frame is detailed in Subsection 1.2.2. The initial conditions of the absolute target and relative chaser match those indicated in Subsection 3.1.1, including the initial mass. The duration of the flight is established at 100 seconds.

Training

To compare Meta-Reinforcement Learning (Meta-RL) and traditional Reinforcement Learning (RL), the LSTM-agent and the MLP-agent (outlined in Section 2.4 and 4.1) are trained for 8M steps. The training curves, which are the mean discounted cumulative rewards of the trajectory roll-outs in relation to the learning step, are shown in Figure B.1.

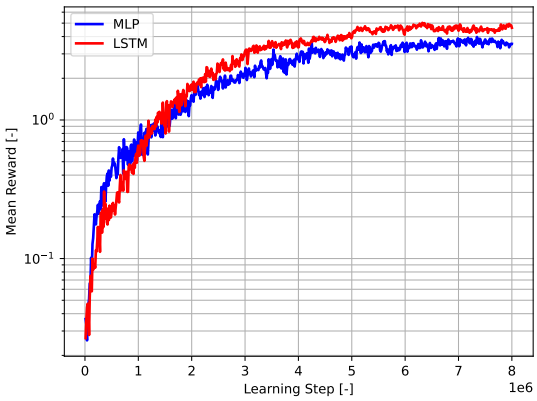


Figure B.1: Mean discounted cumulative rewards of the trajectory roll-outs during the training phases of the LSTM and MLP policies. The numerically solved study case is the 50-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) with plume impingement.

The learning curves show that the rewards gradually increase until they reach a plateau, indicating the success of both learning processes. The recurrent and non-recurrent agents are able to maximize the reward during each episode, thus solving the associated Markov Decision Process (MDP). Trainings are much more stable than in Subsection 3.1.1, due to the plume impingement constraint, which enabled a less "explorative" learning process. As demonstrated in Section 4.1, the LSTM-policy can achieve consistently higher rewards than the MLP-policy. Its internal memory and ability to adapt to changing conditions allowed it to optimize the proposed MDP more effectively, as expected for such a complex problem.

Testing

The LSTM-policy that has been trained is now being used again in the environment for the testing phase, with the same uncertainties as in Section 2.3. The performance of the LSTM-policy is demonstrated in Figure B.2 and Figure B.3, which show the trajectory, relative position and velocity, mass, and thrust profile of a single episode. The Monte Carlo campaign is the only situation in which the MLP-policy is evaluated, as the results of each episode are quite similar to those of LSTMs.

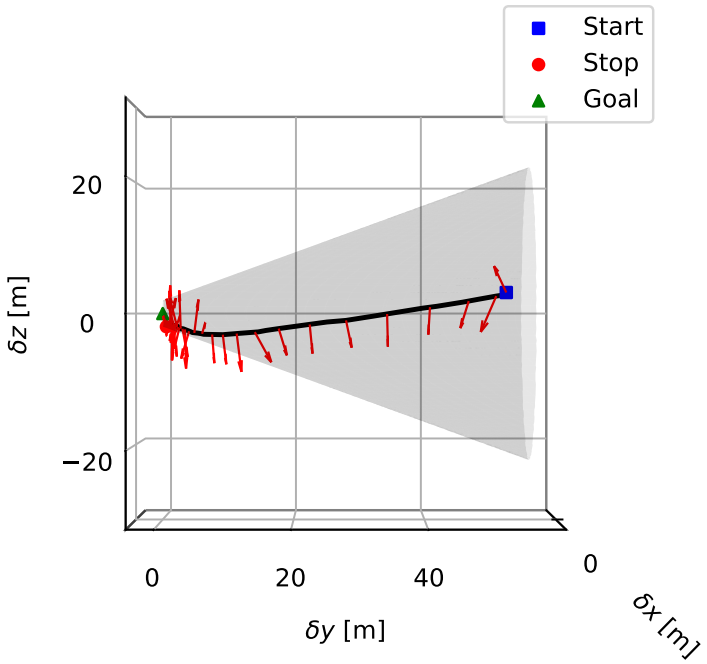
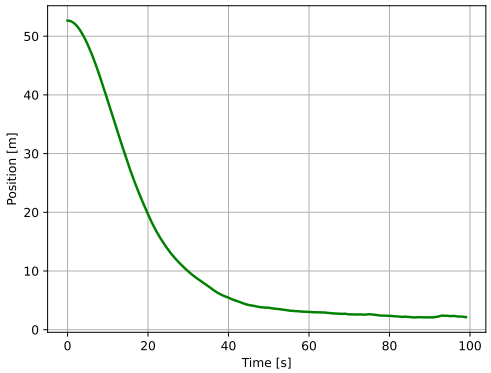
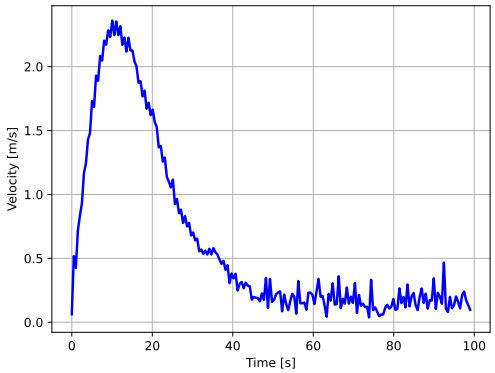


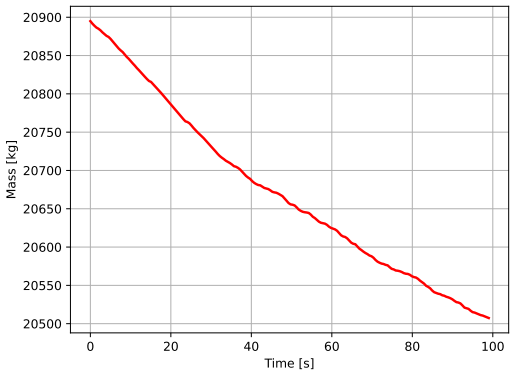
Figure B.2: Trained LSTM-policy for the 50-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case with plume impingement constraint, deployed in the environment and tested for a single episode. Trajectory inside the approach corridor starting from a randomized initial condition.



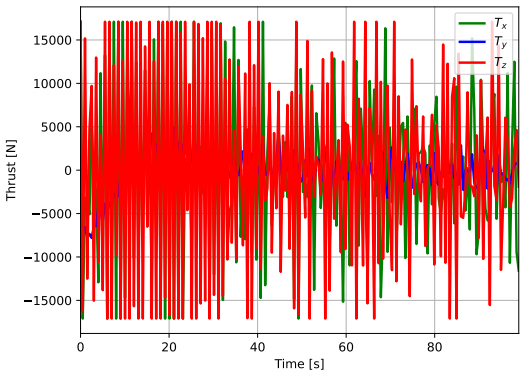
(a) Relative position in time.



(b) Relative velocity in time.



(c) Total mass in time.



(d) Control action in time.

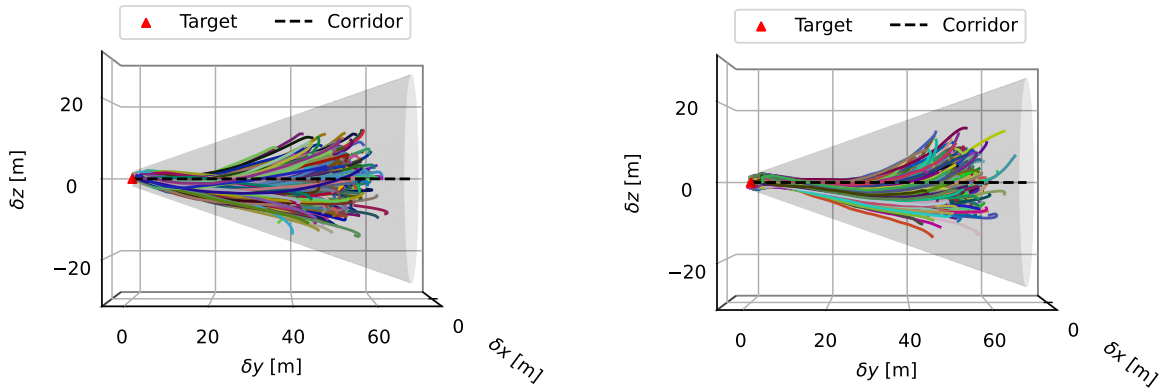
Figure B.3: Trained LSTM-policy for the 50-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case with plume impingement constraint, deployed in the environment and tested for a single episode. Relative position, relative velocity, mass and control action along the trajectory of Figure B.2.

Figure B.2 illustrates that the chaser spacecraft, which starts from a random initial condition (as discussed in Section 2.3 of the MDP), can reach the target without its thrusters' exhaust coming into contact (arrows indicate the opposite direction of plume). The docking port requirement with a final state[1] of $\delta\mathbf{x}_f = \left[1.991\ m,\ 0.097\ m/s\right]$ and the safety approach corridor constraint are satisfied, as can be seen in Figure B.3. The thrust and time of flight are within their maximum values, which make the rendezvous and docking maneuver described in Section 2.1 with plume impingement as an additional constraint a success. The Tsiolkovsky equation (Figure B.3c) shows that the required impulse for

[1]The episodic approach of the reward function, indicated by the *done* signal for docking requirements, prevents the chaser from engaging in an "endless" pursuit of closeness, unlike traditional G&C algorithms. This design is intended to guarantee the uniformity and verifiability of the outcomes; however, it is possible to make the requirements more stringent, though this may prolong the training period.

the task is $\Delta V = 61.320\ m/s$, which is much higher than the optimal value without the plume impingement restriction (as seen in Section 4.2). This outcome is foreseen due to the increased complexity of the problem. The thrust action in Figure B.3d is not smoothly adjusted, but instead has a "bang-bang" pattern. The plume impingement case has been mainly studied as a more thorough comparison between LSTM-policies and MLP-policies.

The LSTM-policy and the MLP-policy that have been trained are now evaluated simulating 500 trajectories, each with a different initial condition based on the uncertainty of the MDP discussed in Section 2.3. Trajectories are illustrated in Figure B.4 and a summary of results is provided in Table B.1.



(a) LSTM-policy. (b) MLP-policy.

Figure B.4: Trained LSTM-policy and MLP-policy for the 50-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case with plume impingement constraint, deployed in the environment and tested for a batch of episodes. Trajectories inside the approach corridor starting from a randomized initial condition.

Results MC: $\mu \pm \sigma$	**LSTM-policy** ($S_r = 82.4\%$)	**MLP-policy** ($S_r = 36.6\%$)
Final Position ρ_f $[m]$	1.998 ± 0.042	1.566 ± 0.343
Final Velocity $\dot{\rho}_f$ $[m/s]$	0.106 ± 0.020	0.164 ± 0.032
Fuel Consumption ΔV $[m/s]$	65.103 ± 5.595	112.177 ± 4.718
Time of Flight ToF $[s]$	91.887 ± 3.956	99.967 ± 2.568

Table B.1: Numerical results sum-up of the LSTM-policy and MLP-policy for the 50-meter Holding Point (HP) within the Keep-Out-Sphere (KOS) case with plume impingement constraint testing, as illustrated in Figure B.4.

In this scenario, the LSTM-policy demonstrates superior effectiveness compared to the MLP-policy, given identical conditions of environment, neural network sizes, and hyperparameters. The MLP-policy has been assessed and found to be inadequate in addressing

the G&C problem, as demonstrated by its lower success rate and higher fuel consumption. The majority of trajectories approach the target closely; however, in their attempt to decelerate and meet the terminal relative velocity requirement, they encounter either plume impingement or fall short within the allotted maximum flight time. The LSTM-policy has been successfully tested ($S_r = 82.4$ %), and most trajectories meet docking port requirements (as indicated by the confidence intervals of the relative terminal states). In addition, all these trajectories abide by the safety regulations of the approach corridor, the maximum control action, the time of flight, and, most importantly, almost all of them also ensure that the chaser thruster plume does not come into contact with the target. This study is only a preliminary examination of the plume impingement scenario. However, the LSTM-policy has demonstrated its ability to effectively handle complex MDPs that include highly randomized parameters, even in this instance.

List of Figures

List of Tables