POLITECNICO
MILANO 1863

School of Industrial and Information Engineering

Adaptive guidance via Meta-Reinforcement Learning: ARPOD for an under-actuated CubeSat

Master's Degree Thesis in Space Engineering

Author: **Gaetano Calabrò**

ID: 939843
Advisor: Prof. Pierluigi Di Lizia
Co-advisor: Michele Maestrini PhD.

May 11, 2022

Abstract

The need for a solution of very complex space missions, aimed at providing global services on a large scale, has played a key role in the development of autonomous spacecrafts. Classical control techniques have been surpassed in terms of autonomy by artificial intelligence methods, such as machine learning. In particular, a promising technique known as Meta-Reinforcement learning is recently emerging as the strongest method to solve a multitude of problems. For space applications, it can be seen as a way through for solving complex control problems such as the guidance of a cluster of small satellites.

This master thesis focuses on demonstrating the ability of Meta-Reinforcement learning algorithm to accomplish a safe planar Autonomous Rendezvous, Proximity Operation and Docking (ARPOD) manoeuvre with an under-actuated CubeSat from different starting points in a small region. Safety considerations and uncertainties on the dynamics rise the complexity of the problem under analysis. The promising future perspective of Meta-Reinforcement learning could enable even more complex missions, providing to mankind the possibility to explore space in an unprecedented way.

Acknowledgments

This is the work that will launch me throughout a new journey. For this reason, I would like to show gratitude to all the people that stood beside me and helped me through this path.

First of all, I would like to thank my Advisor Professor Pierluigi Di Lizia and my Co-Advisor Michele Maestrini PhD. They guided me through all the problems I encountered in this work, helped me with their knowledge and experience. I would like also to acknowledge the creators of the codes that inspired me for this work, Professor Roberto Furfaro, Professor Richard Linares and Professor Brian Gaudet. Alongside them, also the people who shared their knowledge through internet free-courses: Professor Sergey Levine, with his CS285 online course on Reinforcement Learning, and Professor David Silver, from Google DeepMind.

I would like also to thank my family, starting from my sister, Magi. She is the most precious person of my life. She was, still is and will be beside me every day, that is a joyful day or a bad one. She supported me, scolded me when needed and, even if she is younger, I can barely reach the wisdom she possesses. I would like to thank my mother and my father. They also believed in me, in my aspirations and my goals and put everything they could for the sake of my future. I wish to be able to repay them one day even more , for all the sacrifices that they made for me.

I would like to thanks also my friends, starting from the one I know better than anyone, Fortunato. After many years, he is still present in my life. We shared many things, many secrets, many adventures and many laughs. I hope that, soon enough, he will reach his goals and become the person he wants to be. A thanks to Salvatore, the first person I met in Milan, at the Politecnico. This journey was unique thanks to his presence and his guidance. I sincerely hope him the best. I would like also to thank Pietro and Simone, two important friends in my life that helped me during these years. I shared a lot also with them and I would like to share also this.

And you, Alessia. For all you did in these days, I must thank you. Without you, probably, I wouldn't have made it so far. You showed me what is love, care and affection. You accepted the true me. And so I showed you my dreams and my goals and you made it yours. I'm grateful for having you in my life and I hope to be able to give you all that I can in our future. Love you, with all of me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"Scientists study the world as it is; engineers create the world that has never been."
Theodore von Karman

The evolution of space industries is, currently, more and more headed towards the application of small satellites. The growth in the small satellite ecosystem is driven by different motivations, such as the falling costs of smallsats, enabled by the miniaturization of technology for the small satellite platform and the deployment of relatively less expensive commercial off-the-shelf parts[1], the ability to build large constellations for space-based services and the possibility to accomplish complex missions such as on-orbit servicing and debris removal. As a result, small satellites are making inroads in almost every area of space, including communication, remote sensing, technology demonstration, science and exploration.

It is paramount to highlight that human decision-making and engineering effort are still required to solve the complicate tasks previously mentioned, eventually leading to infeasible solutions. The alternative is to develop techniques capable of solving complex missions autonomously.

This work dives into the reality and complexity of an autonomous solution for a rendezvous, proximity operations and docking mission, strongly linked to on-orbit servicing and debris removal applications. In particular, the accomplishment of these challenges requires an high degree of safety and robustness. To cope with these objectives, this study focuses on an Autonomous Rendezvous, Proximity Operations and Docking (ARPOD) mission for a CubeSat, solved through the application of Meta-Reinforcement Learning. The problem objectives and safety constraints introduced reflect the complexity for a robust, safe and efficient rendezvous and docking.

1.1 About space rendezvous and proximity operations

The idea of orbital rendezvous has its roots on the 1960s, when the U.S. National Aeronautics and Space Administration (NASA) announced its plans for a manned spaceflight projects aimed at proving the techniques of orbital rendezvous [2, 3]. However, orbital rendezvous became reality

during the era of the space race between United States and Soviet Union.
The first attempt of orbital rendezvous occurred on 12 August 1962, when the Russian Vostok 4 spacecraft was launched into orbit and reached 6.5 km of distance from the Vostok 3 [4]. This attempt ended with the drift of the spacecraft, piloted by the cosmonaut Andrian Nikolayev [5]. Similarly, the American counterpart begun to put in place its plans. The Gemini program aimed at demonstrating several key objectives, among which the capability to rendezvous and dock in orbit according to proper equipment and techniques [6], preferring manual control over autonomous one. By the summer of 1965, Gemini V was piloted in a phantom rendezvous operation which became the first astronaut-controlled maneuver in space. On December of the same year, the first-ever orbital rendezvous between two spacecrafts occurred. The two spacecrafts, which were Gemini VII, manned by Frank Borman and James Lovell, and Gemini VI, piloted by Walter Schirra and Thomas Stafford, managed to get closer up to 30 cm of relative distance. Only several months later, on 16 March 1966, the first docking between two spacecrafts, Gemini VIII and an Agena target vehicle, occurred [7]. The choice of manual maneuvering awarded NASA, considering the ability of astronauts to adapt to critical problems, resolving them in real time, something that, through autonomous control, would not have been possible in those years.

By the spring of 1967, the Soviet Union set in motion the Soyuz program, able to surpass in complexity and achievements the American Gemini program. In fact, the Russian program accomplished the first rendezvous and docking between two robotic spaceships, the first docking of two manned vehicles and the transfer of crew members from one spacecraft to another [4]. Furthermore, the Soyuz vehicle was designed especially for autonomous orbital rendezvous with the possibility of human maneuvering in case of contingencies. Even considering the great achievements of the Soyuz program, manned operations were interrupt due to the crash of the Soyuz-1 spacecraft during reentry, paving the way for automated missions, such as Kosmos spacecrafts. Indeed, Kosmos 186 and Kosmos 188, two unmanned and thus automated vehicles, managed to rendezvous and dock in October 1967 [8].

The limitations due to the technical difficulties, such as the need of significant cooperation between two manned vehicles to perform close proximity operations, and the new demands of more complex space missions lead to the ascent of smaller spacecrafts, able to perform rendezvous and docking manoeuvres in autonomy. To this purpose, during the 90s, the National Space Development Agency of Japan (NASDA) designed the ETS-VII flight experiment, thus developing a new technology able to autonomously execute close proximity operations. On 7 July 1998, ETS-VII successfully performed the first autonomous rendezvous and docking procedure between uninhabited and robotic spacecrafts [9]. Alongside the Japan Agency, also the American Lockheed Martin Space Systems Company, under the commission of the U.S. Air Force Research Laboratory, joined the new trend of small and automated space vehicles. Indeed, the XSS-11 demonstration mission aimed at developing and verifying on-orbit guidance, navigation and control capabilities to safely and autonomously rendezvous a microsatellite with multiple space objects [10]. It is paramount to add that the

spacecraft was not operating in full autonomy, but could eventually interact with the ground segment, able to select among a variety of operational modes that granted the vehicle the capacity to respond autonomously to different scenarios.

It can be concluded that both the approaches of the U.S. and the old Soviet Union were able to accomplish great achievements, still making evident the downsides and weaknesses of their half-automated programs. For future rendezvous missions where ground or crew intervention is unfeasible, traditional methods of the past must be surpassed.

1.2 Review of state-of-the-art autonomous control techniques

The rising complexity of space missions, as already mentioned in 1.1, brought to the development of sophisticated and autonomous control techniques. To have an overview of state-of-the-art algorithms applied for space rendezvous and proximity operations, different methods are considered and discussed.

One of the most well-established technique is the Linear Quadratic Regulator (LQR). In [11], a Linear Quadratic optimal control is proposed to solve a space rendezvous problem. Particularly, in addition to minimum fuel cost, smooth rendezvous trajectory is considered as another control goal for the rendezvous. Furthermore, in [12] a control algorithm based on a LQR combined with Artificial Potential Fields (APF) method is developed for multiple small spacecrafts during simultaneous close proximity operations. The algorithm is able to guarantee robust close proximity performance and the capability to avoid collisions, while reducing the control effort. In this work, the LQR algorithm is used as a convergence force that guides the controlled spacecrafts towards the desired goal states. Instead, the APF-based functions act as a repulsive field, providing collision free manoeuvres for both fixed and moving obstacles.

Linked to the classical Linear Quadratic Regulator, another common technique is the Model Predictive Control (MPC). The key difference between predictive control and the LQR is that the predictive control solves the optimization problem considering a receding time horizon window whilst LQR solves the same problem within a fixed horizon. In other words, the predictive control is able to perform real-time optimization [13]. In [14], a MPC approach is applied to spacecraft rendezvous and proximity manoeuvring problems. It is demonstrated that various constraints arising in these manoeuvres can be effectively handled with this approach. Particularly, these include constraints on the thrust magnitude, the matching of the approach velocity and the velocity of the docking port, constraints on the position of the spacecraft with respect to the docking port. The results obtained in this work show that the MPC can be an effective feedback control approach able to satisfy various requirements, reduce fuel consumption and provide robustness to disturbances.

In literature, various papers on different approaches with respect to the classical control techniques discussed above can be found. For instance, the so-called path planning algorithms, that consist in prescribing a set of way points for the satellite to follow, are used in [15, 16]. In the work of Ian Garcia and Jonathan P. How, a path planning algorithm is designed for formation fliyng spacecrafts reconfiguration manoeuvres. The Rapidly-exploring Random Trees (RRT) algorithm is adopted and generates a trajectory consisting of a sequence of states, connected by feasible direct trajectories. This trajectory is then passed to a smoother, used to improve its cost. In the other paper cited, the Artificial Potential Field (APF) approach is adopted as a path planning method. It is worth to say that, in this work, the path planner is combined with a reinforcement learning algorithm. Before going into the details of machine learning algorithms applied in space rendezvous missions, it is paramount to illustrate some drawbacks of path planning methods. In particular, due to the fact that the trajectory and the high-level controls are computed once at the beginning of the manoeuvre, the path obtained is followed until manoeuvre termination. Hence, for circumstances in which it is possible to encounter moving objects, path planning algorithms may fail.

In order to surpass all the drawbacks and shortcomings of the already described techniques and approaches, machine learning framework has been recently more and more adopted for solving many kind of problems, including space related ones. For instance, among all the possible machine learning algorithms, Reinforcement Learning is one of the most promising one. Reinforcement Learning algorithms can be applied in a model-free fashion, granting an alternative road with respect to model-based methods, that require a great comprehension of the model and its dynamics. Particularly, in a Reinforcement learning algorithm, it is mainly needed to identify a reward logic so that the Agent, which is the learner, can learn very complex behaviors. Hence the engineering effort in the design of such algorithms is heavily reduced. Moreover, once the correct behavior is learned, the implementation of the policy to follow requires low computational effort and memory, making it possible with current small spacecrafts computing capacity.

Typically, in modern algorithms, Reinforcement Learning is combined with the capability of artificial neural networks of approximating nonlinear functions. The so-called Deep Reinforcement Learning is used in many fields, from Artificial Intelligence (AI) in games to more complex domains such as the space one. Consider, for example [17], where a policy for six-degree-of-freedom docking manoeuvres with rotating targets is developed via reinforcement learning, maximizing performances and reducing control costs. In [18], reinforcement learning is adopted for a six-degree-of-freedom planetary power descent and landing. The policy learned through the learning process maps the lander's estimated state directly to a commanded thrust for each engine, with the policy resulting in accurate and fuel-efficient trajectories. In both works, Proximal Policy Optimization (PPO) is used as the learning algorithm for the policy.

Reinforcement Learning techniques suffer when different tasks or different initial conditions are considered. In other words, the learning process is heavily dependent on the initial inputs that are

fed into the neural networks. For this reason, the next step is taken considering Meta-Reinforcement Learning (Meta-RL). Few works can be found about the application of Meta-RL on space related fields. For example, consider [19], where a Meta Reinforcement Learning algorithm is developed in order to carry out multi-target missions. The same authors of [18], developed Meta-RL algorithms for different missions, ranging from adaptive guidance and navigation [20] to rendezvous and close proximity operations with asteroids [21]. In these works, the application of Meta-Reinforcement Learning resulted in robust and safe solutions, with the Agent able to accomplish its mission even considering uncertainties on the dynamics, the environment and the sensors.
In conclusion, the peculiarity of Meta-RL algorithms is their ability to learn different tasks and remember the entire learning process, becoming an adaptive and robust technique considering the uncertainties of sensors and of the dynamics model that is assumed.

1.3 Impact and outline

Due to the rising complexity of actual space missions, addressed to commercial applications such as global internet service and telecommunications, or to more challenging purposes such as active space debris removal and in-orbit satellite servicing, the need of autonomous spacecrafts, able to operate without or at least with the minimum human control, is continually increasing.
The possibilities granted by the application of algorithms such as reinforcement learning and meta-reinforcement learning can definitely help in the accomplishment of such missions. In particular, thanks to the adaptivity and robustness of meta-reinforcement learning algorithms, even more complex space missions can be successfully achieved. This work aims at showing the capacity of meta-reinforcement learning to solve a three-degree-of-freedom rendezvous, proximity operation and docking manoeuvre, in full autonomy.

This document is organized so that the reader can understand the mathematical framework and the theory behind meta-reinforcement learning. In chapter 2, the Autonomous Rendezvous, Proximity Operations and Docking (ARPOD) problem is described, considering the environmental setup, the dynamics behind it and the constraints. In chapter 3, meta-reinforcement learning is explained, introducing the theoretical notions needed to understand the algorithm. Afterwards, in chapter 4, an adequate overview of Artificial Neural Networks is provided focusing on Recurrent Neural Networks necessary for the implementation of a Meta-RL algorithm . In chapter 5, the information of the three previous chapters are used to describe how the meta-reinforcement learning algorithm is implemented to solve the ARPOD problem. Finally, the last chapter, chapter 6, provides the results obtained for different test cases, concluding the entire work and showing the ability of the Agent to learn.

Chapter 2

Autonomous Rendezvous, Proximity Operations and Docking

The peculiarity of reinforcement learning and, equally, meta reinforcement learning is that they are model-free algorithms. In other words, the Agent, that will be detailed in the next chapter, is able to learn and take actions independently from the knowledge of the problem. The only dependence can be found on the environment, with which the Agent interacts in order to take actions. This means that it is paramount to describe the problem in which the Agent is thrown in order to solve it. This chapter aims at illustrating the spacecraft model and the dynamics of a planar (2D) Autonomous Rendezvous, Proximity Operations and Docking (ARPOD) manoeuvre, adding the constraints for a feasible, safe and reliable solution, assumed considering the problem proposed by [22] and already dealt with via reinforcement learning by [23].

It is worth clarifying what is intended for rendezvous, proximity operations and docking before going into the details of the problem. According to [24], the Concept of Operations of ARPOD missions are generally divided into three main phases:

- An initial phase, named as rendezvous phase, that includes the approach of one spacecraft, called chaser, to another one, referred to as target. Typically, this phase is considered between 10 km to 1 km of relative distance between the two spacecrafts.
- The close rendezvous phase, also known as proximity operations phase, is in the range between 1 km and 100 m. In this phase, the attitude is controlled so that the chaser docking port is aligned with respect to the line of sight region of the Target docking port.
- The final phase is the docking phase, in which final manoeuvres are executed to engage the docking ports. The range at which this phase begins is from 100 m to 0 m of separation.

With this in mind, the full ARPOD mission can be initialized considering an initial distance that enters inside the rendezvous phase range.

2.1 Problem Statement

This work considers an ARPOD problem, with two spacecrafts orbiting around the Earth, and aims at the achievement of a successful docking between the two. One of the two spacecrafts is the Target, that, as the name suggests, is the target of the docking. It is assumed to be non-rotating during the entire approach. The second one is the Chaser, that is controlled for the entire manoeuvre and shall dock with the Target ensuring the safety of both spacecrafts.

In this thesis, a successful mission is achieved when the Chaser reaches less than 10 m of separation from the target, with a maximum absolute relative velocity of 0.2 m/s and an absolute relative angle lower than 5°. The docking is achieved when the controlled spacecraft reaches less than 1 m of relative distance from the Target. With respect to the work of [23], thanks to the robustness granted by Meta-Reinforcement Learning, the chaser is still considered omniscient, which means that it knows its relative position, velocity and attitude, but uncertainties are added on its state.

Briefly, the problem faced in this work can be summarized in three high-level objectives:

- *The Chaser must remain inside the Target docking port Line of Sight (LoS) during the entire final phase, the docking one.*
- *The safety of both the two spacecrafts must be ensured throughout the duration of the manoeuvre.*
- *The Chaser must asymptotically dock with the Target with*:

$$||\mathbf{r}|| \leq \mathbf{1m} \quad \text{and} \quad |\theta| \leq \mathbf{5}^\circ \tag{2.1.1}$$

2.2 Chaser Model

As proposed by [22], the model adopted in this problem is a 6U CubeSat measuring 10 cm x 20 cm x 30 cm, as it is pictured in Fig. 2.2.1.
According to the model described above, thrust is only possible from two thrusters assumed to be perfectly aligned with the spacecraft body x-axis $\mathbf{x_b}$ on both sides of the spacecraft. Furthermore, considering the conventions depicted in Fig. 2.2.1, a positive total thrust F induces a displacement in the positive direction of the x-body axis. The attitude is controlled only about the chaser's z-axis $\mathbf{z_b} \equiv \mathbf{n_{rw}}$ through a reaction wheel. Lastly, the docking port of the chaser is assumed to be on the surface with normal aligned to the x-body axis $\mathbf{x_b}$.

In the Table 2.2.1, inertial properties of the spacecraft model are reported. They are taken from the proposed challenge already cited but whatever value can be used, in case of a different problem to tackle. Furthermore, the Meta-RL is able to solve problems even for uncertain inertia properties, not considered for the final solution proposed in this work.

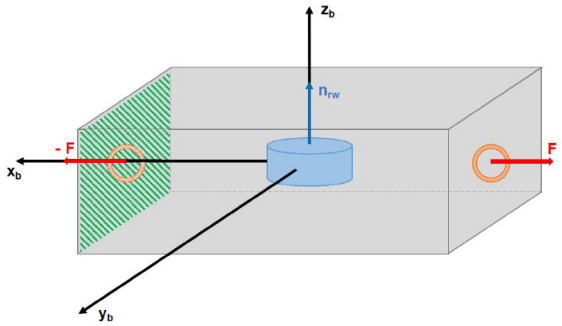


Figure 2.2.1: *6U CubeSat with thrusters (red) aligned with positive and negative x-axes, a reaction wheel (blue) for single axis attitude control aligned with the body z-axis, and the docking port (green) normal to the positive body x-axis. Image taken from [23].*

Variable	Description	Value
D	Reaction wheel spin axis mass moment of inertia	$4.1 \times 10^{-5}\ kg\ m^2$
I_{zz}	Spacecraft mass moment of inertia in z-axis	$5.6 \times 10^{-2}\ kg\ m^2$
m	Spacecraft mass	$12kg$

Table 2.2.1: Inertia properties of the Chaser

2.3 Dynamics model

The equations of motion are obtained following the assumptions, considered in [22], that can be found below:

Assumption 1 *Both spacecrafts are rigid bodies.*
Assumption 2 *The mass of Earth is significantly greater than the mass of the spacecraft.*
Assumption 3 *The mass loss of the Chaser spacecraft is significantly smaller than the total mass of the spacecraft.*

The *Assumption 1* can be applied to most modern spacecrafts as fuel slosh and moving mass are typically not significant contributes to the dynamics of the spacecraft. According to *Assumption 2*, the only gravitational contribute comes from the Earth's mass, condensed inside the universal gravity parameter μ. Thanks to *Assumption 3*, the mass of the spacecraft can be assumed as constant as propellant usage over short time intervals is negligible.
Under these assumptions, the motion of the two spacecrafts around the Earth is governed by the following equations:

$$\ddot{\mathbf{R}}_j = -\frac{\mu}{R_j^3}\mathbf{R}_j \tag{2.3.1}$$

where $j \in t, c$ are the subscripts denoting, respectively, the target and the chaser. $\mathbf{R}_j \in \mathbb{R}^3$ is

the position vector of the spacecraft in the inertial frame, R_j is the module of the position vector, $R_j = ||\mathbf{R}_j||$, and $\ddot{\mathbf{R}}_j \in \mathbb{R}^3$ is the acceleration of position always referred to the inertial frame. The eq. (2.3.1) describes a stable and elliptical orbital motion of the spacecrafts around the center of gravity of the frame, that can be identified approximately with the center of the Earth, according to *Assumption 2*. In order to simplify the problem, dynamics can be linearized. Linearization requires additional assumptions, such as the ones that follow:

Assumption 4 *The target spacecraft is in a circular orbit with radius R_t*
Assumption 5 *The relative distance between the target and the chaser spacecrafts is significantly smaller than the distance of the target with respect to the center of the Earth*

Typically, a good number of spacecrafts revolves around the Earth in near-circular orbits. Hence, *Assumption 4* is reasonable, assuming that the target spacecraft is expected to be launched into a circular orbit. *Assumption 5* is justified by the fact that rendezvous typically happens within distances on the order of tens of kilometers, while the the target spacecraft is in a circular orbit with altitude fixed at 500 km.

Generally, to simplify the equations that describe a rendezvous, a non-inertial frame can be attached to the Target, due to the fact that it is in an equilibrium orbit, which is coherent with *Assumption 4*. The non-inertial frame here considered is the *Hill's Frame*, useful for the description of the orbit of one body about each other [25].

The Hill's Frame (Fig. 2.3.1) is defined by the following axes:

- $\mathbf{e}_R$ identifies the radial direction that points outwards from the Earth's center.
- $\mathbf{e}_N$ for the normal direction, aligned with the angular momentum vector of the orbit, constant and always orthogonal to the orbital plane.
- $\mathbf{e}_T$ for the tangential direction wich completes an orthogonal coordinate system with the two previous unitary vectors. Furthermore, for a circular orbit, $\mathbf{e}_T$ and the inertial orbital velocity are aligned.

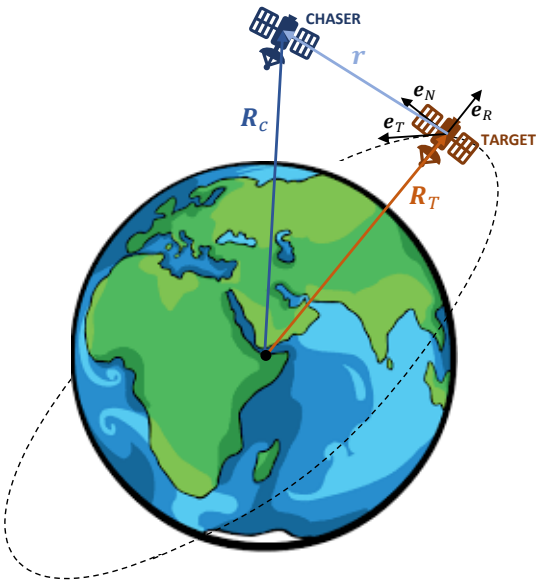


Figure 2.3.1: *Hill's reference frame in black. The positions of the chaser and the target with respect to the inertial frame are pictured, respectively, in blue and red.*

In the Hill's frame, the relative position between the Target and the Chaser is denoted as **r**:

$$\mathbf{r} = x\mathbf{e}_R + y\mathbf{e}_T + z\mathbf{e}_N \tag{2.3.2}$$

It follows that the equation of motions can be written in a simplified form, according to the newly introduced reference frame [26]:

$$\begin{aligned} \ddot{x} - 2\sqrt{\tfrac{\mu}{R_T^3}}\dot{y} - 3\tfrac{\mu}{R_T^3}x &= 0 \\ \ddot{y} + 2\sqrt{\tfrac{\mu}{R_T^3}}\dot{x} &= 0 \\ \ddot{z} + \tfrac{\mu}{R_T^3}z &= 0 \end{aligned} \tag{2.3.3}$$

In addition, it can be introduced the parameter named as mean motion and defined as $n = \sqrt{\frac{\mu}{R_T^3}}$, considering that, for a circular orbit, where R_T is kept constant, the mean motion remains constant too. The linear equations of motion (2.3.3) can be decoupled into the orbital in-plane motion, in the ($\mathbf{e}_R$,$\mathbf{e}_T$) plane, and out-of-plane motion. Hence, in this work, only in-plane motion is considered. It is worth mentioning that, in more complex models, the equations of motion are coupled, thus the approximation here considered is not well-suited.

Therefore, the *planar Clohessy-Wiltshire equations* can be written as:

$$\begin{aligned} \ddot{x} - 2n\dot{y} - 3n^2 x &= 0 \\ \ddot{y} + 2n\dot{x} &= 0 \end{aligned} \tag{2.3.4}$$

It can be noticed that all the coefficients are constant, thus an analytical solution there exists [27]. In matrix notation:

$$\begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} 4 - 3\cos nt & 0 & \frac{\sin nt}{n} & \frac{2}{n}(1 - \cos nt) \\ 6(\sin nt - nt) & 1 & \frac{2}{n}(\cos nt - 1) & \frac{1}{n}(4\sin nt - 3nt) \\ 3n\sin nt & 0 & \cos nt & 2\sin nt \\ 6n(\cos nt - 1) & 0 & -2\sin nt & 4\cos nt - 3 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ \dot{x}_0 \\ \dot{y}_0 \end{bmatrix} \tag{2.3.5}$$

Where $\begin{bmatrix} x_0 & y_0 & \dot{x}_0 & \dot{y}_0 \end{bmatrix}^T = \mathbf{x}_0$ is the translation state vector at time t = 0.

The rotational equations of motion can be obtained by the application of the conservation of angular momentum. The only rotational degree of freedom is about the $\mathbf{e}_N$ axis, according to the approximations made above. The inertia of the spacecraft about the normal axis in the Hill's frame is given by I_{zz} and the one of the reaction wheel results to be D. The numerical values can be found in Table 2.2.1. The rotation of the Chaser body axis $\mathbf{x}_b \equiv \mathbf{n}_C$ about $\mathbf{e}_N \equiv \mathbf{z}_b$ is denoted as θ_N. Since, as already stated in Sec. 2.1, the target spacecraft is not rotating, it is trivial to define the relative rotation of the Chaser with respect to the Target, which is the angle θ_N. By convention, θ_N is measured from the inward normal vector of the Target docking port $-\mathbf{n}_T$. A graphical representation is showed in Fig. 2.3.2.

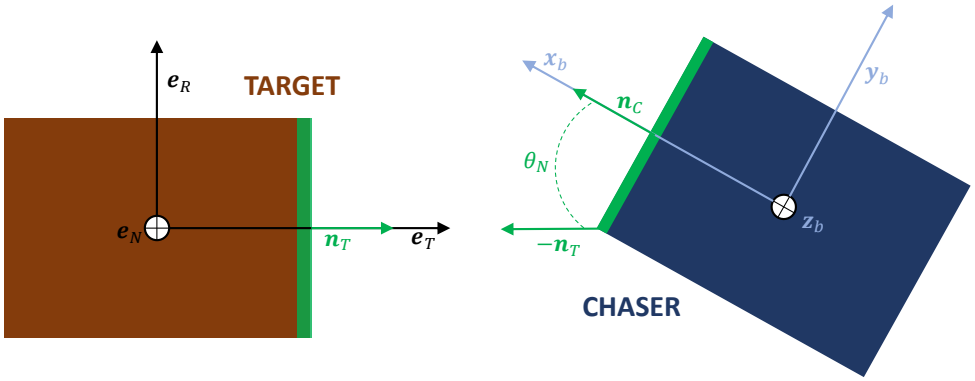


Figure 2.3.2: *Convention of the attitude angle θ_N in the Hill's frame (black). The docking port is displayed in green, along with their respective normal vectors $\mathbf{n}_j$, for both Target (dark red) and Chaser (dark blue).*

The rotational equation of motion of the Chaser is:

$$I_{zz}\ddot{\theta}_N = -D\dot{\psi} \tag{2.3.6}$$

Where $\dot{\psi}$ is the acceleration given to the reaction wheel to produce a torque, required and commanded by the control system. The eq. (2.3.6) can be solved analytically and the solution is:

$$\begin{aligned} \dot{\theta}_N &= -\tfrac{D}{I_{zz}}\dot{\psi}t + \dot{\theta}_{N_0} \\ \theta_N &= \dot{\theta}_N t + \theta_{N_0} \end{aligned} \tag{2.3.7}$$

2.4 Constraints definition

A set of constraints is implemented to assure safety and feasibility of the trajectory, following the guidelines for an assured satellite proximity operations, given by [22].

- **Constraint 1** - *Asymmetric bounded thrust* ($F \in [F_{min} : F_{max}]$): the Chaser is under reasonable thrust limitations. In some cases the thrust capabilites may not be equal in all directions. In this case, due to the size and weights of a typical CubeSat, one thruster is assumed to be twice as powerful as the other.
- **Constraint 2** - *Maximum reaction wheel velocity* ($|\psi| \leq \psi_{max}$): referred to the physical limitations of the attitude control actuator, the reaction wheel. In some cases, in order to reduce overall wear [28], in it is possible to impose a maximum velocity lower than the possible one, increasing durability of the reaction wheel.
- **Constraint 3** - *Maximum reaction wheel acceleration* ($|\dot{\psi}| \leq \dot{\psi}_{max}$): As before, to prevent a premature failure of the reaction wheel due to excessive wear, a repeated cycle of operation of the reaction wheel at the extreme ends of positive and negative acceleration is avoided [28].

The first set of three constraints embeds the physical limitations of the actuators and the prevention of premature failures due to excessive overload. The next set of constraints is added to enforce limitations on the relative velocity and attitude of the Chaser, due to safety considerations.

- **Constraint 4** - *Recoverable relative velocity limit* ($|\dot{x}| \leq v_{x_{max}} and |\dot{y}| \leq v_{y_{max}}$): to maintain recoverable relative motion, the relative velocity is constrained to not exceed a maximum threshold, allowing the actuators to arrest motion within a limited time frame. In this work, the maximum velocity is taken as the one that the spacecraft can travel in the x or y direction and stop within one minute, considering the available thrust. In other words:

$$v_{x_{max}} = v_{y_{max}} = v_{max} = \frac{F_{max}}{m} t_{stop} \tag{2.4.1}$$

- **Constraint 5** - *Bounded relative velocity limit* ($||\mathbf{v}|| \leq v_{dock} + f_s||\mathbf{r}||/T_c$): it links the speed and the relative distance, meaning that the Chaser should not be traveling exceedingly fast when is getting closer to the Target. This constraint can be formalized using a temporal

requirement to avoid collisions, through the time-to-collision T_c, function of the available thrust, spacecraft mass and separation from the Target.

$$T_c = \sqrt{\frac{2m}{F_{max}}||\mathbf{r}||} \tag{2.4.2}$$

The relative velocity limit is thus defined, including a safety factor f_s. The maximum final velocity is constrained to respect a threshold to ensure a safe docking. In this case, from the problem statement, it follows that: $v_{dock} = 0.2\ ms^{-1}$

- **Constraint 6** - *Maximum angular velocity* ($|\dot{\theta}_N| \leq \dot{\theta}_{N_{max}}$): As for the translational velocity, also the spacecraft rotational velocity is limited, allowing to react or recover from commands in a reasonable time frame.

- **Constraint 7** - *Maximum angular acceleration* ($|\ddot{\theta}_N| \leq \ddot{\theta}_{N_{max}}$): excessive rotational acceleration may cause damage to the spacecraft structure, payload or one of all the appendages and deployables, such as solar panels, antennas and so on. To avoid these circumstances, a constraint is introduced to the angular acceleration.

Finally, the last constraint is imposed to force the Chaser to remain inside the Line-of-Sight (LoS) of the Target docking port sensors during the docking phase.

- **Constraint 8** - *Docking cone* ($\alpha_C \leq \alpha_{LoS}$): for a planar problem, it is trivial to define the LoS. Indeed, it is the section of a disk characterized by a semi-angle α_{LoS} with respect to the outward normal vector of the Target docking port $\mathbf{n}_T$.

The safety and physical constraints are summarized in Table 2.4.1. Numerical values, assumed for the problem faced in this work, are indicated.

#	Description	Expression	Value
1	Asymmetric bounded thrust	$F \in [F_{min} : F_{max}]$	$F_{min} = -1N$ and $F_{max} = 2N$
2	Maximum RW velocity	$\|\psi\| \leq \psi_{max}$	$\psi_{max} = 576.0\ rad\ s^{-1}$
3	Maximum RW acceleration	$\|\dot{\psi}\| \leq \dot{\psi}_{max}$	$\dot{\psi}_{max} = 181.3\ rad\ s^{-1}$
4	Recoverable relative velocity limit	$\|\dot{x}\| \leq v_{x_{max}} and \|\dot{y}\| \leq v_{y_{max}}$	$v_{max} = 10\ m\ s^{-1}$
5	Bounded relative velocity limit	$\|\|\mathbf{v}\|\| \leq v_{dock} + f_s\sqrt{\frac{F_{max}}{2m}||\mathbf{r}||}$	$v_{dock} = 0.2\ m\ s^{-1}$ and $f_s = 1$
6	Maximum angular velocity	$\|\dot{\theta}_N\| \leq \dot{\theta}_{N_{max}}$	$\dot{\theta}_{N_{max}} = 2\ deg\ s^{-1}$
7	Maximum angular acceleration	$\|\ddot{\theta}_N\| \leq \ddot{\theta}_{N_{max}}$	$\ddot{\theta}_{N_{max}} = 1\ deg\ s^{-2}$
8	Docking cone	$\alpha_C \leq \alpha_{LoS}$	$\alpha_{LoS} = 45°$

Table 2.4.1: Summary of the constraints and their numerical values

Chapter 3

Deep Meta-Reinforcement Learning

Before diving into the details of Deep Meta-Reinforcement learning, it is imperative to explain how standard Reinforcement learning works.
Reinforcement learning is, as previously mentioned in Sec. 1.2, a model-free algorithm. This means that this kind of learning process can be applied independently from the problem that is faced. So, one may wonder how Reinforcement learning process works. In particular, a RL learner, also known as *Agent*, learns what to do through the interaction with the *environment*, that is peculiar for each problem under analysis. This is possible thanks to a logic that will be explained hereafter that *rewards* the Agent if the selected action is a good one. Hence, simply-put the learning process is based on getting the best outcome from all the actions that are taken. However, the Agent is not told which actions it should take, but it must discover which one yield the best reward.

Differently from supervised and unsupervised learning, where, respectively, the learner is trained on a database of labelled examples provided by a knowledgeable external supervisor or tries to find a hidden structure in a database of unlabelled data, Reinforcement learning Agent learns "autonomously", optimizing the reward it gets throughout the entire learning phase. It can be said that Reinforcement learning, combined with the use of Neural Networks (deep RL), is similar to a peculiar learning process that everyone knows: human learning. However, there are at least two aspects of human-like learning that Deep Reinforcement learning lacks. First, deep RL requires a massive volume of training data compared with the ability of humans to learn a wide range of tasks even with less experience. Second, deep RL Agent is typically specialized on one restricted task domain, whereas humans are able to adapt to changing task conditions, using the past knowledge of similar but not equal tasks [29].
In order to meet these challenges, *Deep Meta-Reinforcement learning* algorithms have been developed. The main feature is that standard deep RL techniques are used to train a particular kind of neural networks, known as Recurrent Neural Network (RNN) in such a way that the recurrent network is able to implement its own and autonomous RL procedure, to tackle different problems. As a consequence, this procedure grants an adaptiveness and sample efficiency that the standard RL procedure lacks.

3.1 Reinforcement Learning elements

Being a Reinforcement Learning procedure, Meta-RL shares the same features of standard RL algorithms. The main elements of a reinforcement learning system are typically the following ones:

- The *environment*, peculiar for any problem. It is described, for this specific case, through the dynamics, with the equations of motion. It dictates the evolution of the states of the learner, through the laws of the dynamics.
- The *Agent*, which is the learner. It is thrown into the *environment* to interact with it, through a set of actions, and retrieve the evolution of its states by observations.
- The *policy*, that defines how the Agent behaves at a given time. In other words, it maps the states in which the Agent currently is, by interacting with the environment, into a set of actions to be taken. The *policy* is the main actor of a reinforcement learning algorithm. It is indeed sufficient to determine the correct behavior that the Agent should follow. Typically, *policies* are stochastic and defined as a probability distribution for each action.
- The *reward function* defines the goal a reinforcement learning problem. From the interaction with the environment, once that an action has been selected, the Agent receives a numerical prize, known as *reward*. If the action that has been selected is a "bad" one which means that the state in which the Agent is, it is not a "good state", the reward received is low. When this is the case, the *policy* is typically changed and different actions are taken. The global objective of a RL algorithm is to maximize the *total reward*, given as a sum of the rewards accumulated throughout the entire trajectory.
- The *value function* is, instead, the total amount of reward, referred to a state, that the Agent can expect to accumulate over the future, starting from that state. In other words, the *value function* indicates whether starting from a certain state can grant the maximum reward in the future.

In the framework of Reinforcement learning, a peculiar challenge that arises is the trade-off between exploration and exploitation. In particular, the Agent, in order to obtain the highest amount of reward, must prefer actions that it has collected during its experience and found to be effective in producing reward. On the other hand, to discover such actions or, eventually, better ones, it has to explore, selecting new actions. In other words, the Agent has to *exploit* what it already knows, obtaining rewards, but it has also to *explore* in order to find better actions, though neither of the two can be preferred or individually pursued without failing the task [30]. The exploration-exploitation trade-off is still under analysis among mathematicians, even if some techniques to decouple them without major drawbacks has been already developed and applied in Meta-Reinforcement learning [31], but not considered in the context of this work.

3.2 Markov Decision Processes

In order to fully understand the following description of the learning process of a meta-reinforcement learning or, more generally, of a reinforcement learning-based algorithm, it is mandatory to introduce notions about *Markov Decision Processes* (MDPs) [30].

A reinforcement learning task that satisfies the so-called Markov property is defined as a *Markov Decision Process*. The Markov property is a characteristic of an environment of being able to provide a complete information about the state to the Agent, without keeping track of the past history. In short, the current information is a sufficient statistic for the future. Formally, in a more mathematical sense:

$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, ..., S_t] \tag{3.2.1}$$

The above equation means that the probability to end up in the next state S_{t+1} depends only on the actual state S_t, thus the history of all the visited states is not a necessary information. The eq. (3.2.1) can be generalized including actions and rewards.

In a *Markov decision process*, the timespan is discretized into time steps, $t = 0, 1, 2, 3$ ... .At each time step, the Agent receives information of the environment's state $S_t \in \mathcal{S}$, where $\mathcal{S}$ is the so-called *State Space*, that contains the set of possible states. In state S_t, an action $A_t \in \mathcal{A}(\mathcal{S}_t)$ is selected, where $\mathcal{A}(S_t)$ is the set of actions available in state S_t, also known as *Action Space*. One step later, as a consequence of its action and the previous state, the Agent ends up in a new state S_{t+1} and receives a *reward* $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$. This cycle, illustrated in Fig. 3.2.1 is repeated until some end conditions are met, defining an *episode*. In each episode, a sequence of states, actions and rewards is collected and it is known as *trajectory*, usually in the form:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, S_3...$$

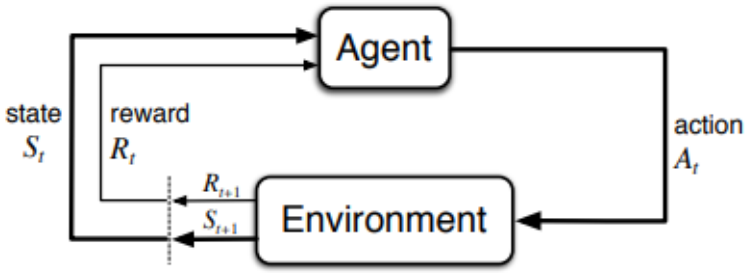


Figure 3.2.1: *Agent-environment cyclic interaction.* Figure taken from [30]

Consider now a finite Markov decision process, which means that the state space $\mathcal{S}$, the action-state space $\mathcal{R}$ and the reward function $\mathcal{R}$ are all discrete and finite spaces. By including actions and rewards on the equation (3.2.1), it is possible to define the *transition probability* as the the

probability to end up in a particular state $s' \in \mathcal{S}$ and collect a reward $r \in \mathcal{R}$ as a function only of the current state $s \in \mathcal{S}$ and the action $a \in \mathcal{A}(s)$, associated with it .

$$p(s', r \mid s, a) = \mathbb{P}(S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a) \tag{3.2.2}$$

Where $p(s', r \mid s, a)$ is intended as a probability distribution for each choice of s and a (eq. (3.2.3)).

$$\forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s), \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) = 1 \tag{3.2.3}$$

Given the finite MDP dynamics specified at eq. (3.2.2), it is possible to compute the marginal probability of ending up in a state s' starting from a state s and selecting action a. This is known as *state-transition probability* and can be expressed mathematically as:

$$p(s' \mid s, a) = \mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a) = \sum_{r \in \mathcal{R}} p(s', r \mid s, a) \tag{3.2.4}$$

And, similarly, the expected reward as a function of a state-action pair s and a can be obtained starting from eq. (3.2.2) and evaluating the expected value of the rewards. This operation, applicable to any finite and countable random variable, is defined as the sum of all the values assumed by the variable weighted by their probabilities.

$$r(s, a) = \mathbb{E}[R \mid S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r \mid s, a) \tag{3.2.5}$$

It is worth mentioning that all the discussion above is valid only when dealing with *finite* Markov decision processes. Naturally, there exist a formulation suited for continuous random variables [32], not treated in this work.

3.3 Deep Meta-Reinforcement learning key features

After the brief overview on reinforcement learning framework, Deep Meta-Reinforcement learning can be discussed and detailed. For the sake of clarity, from now on Deep Meta-RL will be referred simply as Meta-RL. As already mentioned, Meta-RL Agents are able to learn different but similar tasks. In this work, for example, the tasks that make up the training process are interrelated RL problems, where only initial conditions are changed.

Consider now a distribution, the prior, over Markov Decision Processes (MDPs), named as $\mathcal{D}$. Meta-RL is able to learn a prior-dependent RL algorithm, in the sense that it will perform well on average on MDPs drawn from the distribution $\mathcal{D}$. Meta-RL algorithm is characterized by two phases: *training* and *testing*. During training, the Agent, embedding a Recurrent Neural Network (Sec. 4.2), interacts with a sequence of MDP environments, also called as tasks, through episodes. At the beginning of a new episode, a new MDP environment $m \in \mathcal{D}$ and an initial state for this task are sampled, while the internal state of the Agent, in terms of the pattern of activation of the

neurons over its recurrent units, is reset. The agent then executes the action-selection strategy on the newly generated environment for a certain number of discrete time-steps. At each time-step, as it happens in a non-meta Reinforcement learning algorithm, an action $a_t \in \mathcal{A}$ is executed as a function of the whole history $\mathcal{H}_t = \{x_0, a_0, r_1, ..., x_{t-1}, a_{t-1}, r_t, x_t\}$ of the agent interacting in the MDP $m \in \mathcal{D}$ during the current episode. The network weights are then trained to maximize the sum of observed rewards over all steps and episodes [29].

After the training process, the Agent's policy is fixed, which means that the weights of the neural network are frozen, but the activations are changing due to inputs from the environment and the hidden state of the recurrent layers. This phase, known as *testing*, consists then in the evaluation of the policy on a set of MDPs that can be drawn from the same distribution $\mathcal{D}$ used during training or from a slightly modified version of that distribution. The internal state goes under reset at the beginning of the evaluation of any new episode. Now, since the policy is fixed, the Agent is history-dependent, due to the fact that embeds a recurrent network. Finally, when dealing with any new MDP environment, the Agent is able to adapt and select a strategy that optimizes rewards for that task [29].

In order to fully understand how this Agent is able to learn and adapt to new similar environments, Artificial Neural Networks and, especially, Recurrent Neural Networks are explained in chapter 4.

3.4 Policy Gradient Methods

Any reinforcement learning or meta-reinforcement learning algorithm focuses on training an Agent, which embeds a neural network, to learn how to take actions and accomplish predetermined objectives. In this section, the way the Agent selects an action and learn will be discussed.

Firstly, it is necessary to specify that the Agent chosees an action according to a certain *policy* π. This policy maps an observation O_t of the Agent's environment to an action A_t, making the Agent a closed-loop controller.

Different approaches have been adopted in literature. Among them, the ability to learn a parameterized policy without consulting a *value function* is applied in this work. This means that the strategy learned from the Agent is independent from the knowledge of the expected cumulative reward. However, as it will be explained, the *value function* will still be used for the learning process of the policy parameter, but not for the action selection.

3.4.1 Policy Approximation

The policy is a neural network that takes as input signal the observation O_t, coming from the interaction with the environment, and outputs a median and standard deviation as output, in

order to define a probability distribution. This probability is written as $\pi_\theta(\mathbf{a} \mid O_t = \mathbf{o})$, where θ are the parameters of the policy neural network.
It is paramount to underline that there exists a distinction between observation and state. Indeed, in some scenarios, the observation may contain more or less information of the state. This heavily depends on which kind of MDP is under analysis. Typically, when a MDP is not *fully observable*, this distinction between observation and state must be taken into consideration and the MDP is defined as a Partially Observable Markov Decision Process (POMDP). In the context of this work, however, since the chaser is assumed to be omniscient, see chapter 2, the MDP is a fully observable one. Hence, the observation and the state are one and the same.

Going back to the definition of the policy, the probability to take a certain action $\mathbf{a}$ at time tm given the environment in state $\mathbf{s}$ at the same time step, with parameter θ, can be written as follows:

$$\pi_\theta(\mathbf{a} \mid \mathbf{s}) = \pi(\mathbf{a} \mid \mathbf{s}, \theta) = \mathbb{P}(A_t = \mathbf{a} \mid S_t = \mathbf{s}, \Theta_t = \theta) \tag{3.4.1}$$

The learning process of the policy, being itself a neural network, consists in the update of the policy parameters θ based on the information contained in the tuples (S_t, A_t, R_{t+1}). Eventually, at the end of the learning process, the policy becomes the optimal one, which is typically denoted as π^*.

3.4.2 Policy gradient methods

The learning process of the policy, parameterized in θ, consists in the optimization of an objective function $J(\theta)$ over the parameter space.
Policy gradient methods aim at the maximization of this cost function, hence their updates approximate gradient ascent in J. For the sake of clarity, gradient ascent differs from gradient descent only in the sign.

$$\theta_{t+1} = \theta_t + \alpha \widehat{\nabla_\theta J(\theta_t)} \tag{3.4.2}$$

In eq. (3.4.2), α is the learning rate associated with the policy neural network and $\widehat{\nabla_\theta J(\theta_t)}$ is a stochastic estimate of the policy gradient. By applying the expectation of the estimate, the policy gradient $\nabla_\theta J(\theta_t)$ is approximated with respect to its parameters θ.

There exist different expressions for the definition of the policy gradient objective function [33]. A trivial choice could be the expected sum of discounted rewards, that identifies the value function $V^{\pi_\theta,\gamma}(\mathbf{s}_0)$. In this way, the optimization of the cost function results in the maximization of the expectation or, equivalently, in the collection of the highest possible discounted reward.

$$J(\theta) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] \tag{3.4.3}$$

However, the most commonly used gradient estimator [33] has the form:

$$\nabla_\theta J_t(\theta) = \mathbb{E}_t \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) A_w^\pi(\mathbf{s}_t, \mathbf{a}_t) \right] \tag{3.4.4}$$

In eq. (3.4.4), the gradient of the objective function consists in the expected value of the sum of the product between the gradient of the log probability $\log\pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t)$ and the estimation of the Advantage function, detailed in Sec. 3.4.3. The use of the log probability over the probability distribution is a common trick in machine learning [34]. In particular, when the probability distribution is a likelihood function, the derivative of its log probability acts as a weighted gradient with reward, that pushes more in the direction of higher reward and viceversa.
From the definition of the gradient, which is the derivative of the objective function, it can be finally written that:

$$J_t^{PG}(\theta) = J(\theta) = \mathbb{E}_t \left[\sum_{t=0}^{\infty} \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) A_w^\pi(\mathbf{s}_t, \mathbf{a}_t) \right] \tag{3.4.5}$$

The objective function expressed in eq. (3.4.5) depends on the value assumed by the advantage. Especially, if the advantage function is positive, which means that the action selected by the Agent resulted in a better average return than expected, the probability of selecting that action increases. On the contrary, if the advantage is negative, the probability decreases. Even if less intuitive, this expression of the objective function is more prone to the selection of a better policy.

3.4.3 Value Function and Advantage

The goal of the Agent during the learning process it to maximize the total expected reward. However, since rewards are a quantification of a single state transition, the maximization of each reward obtained could lead to a poor policy or, eventually, to the complete failure of the task. Therefore the Agent must take into consideration not only the direct consequences of its actions but also their long term effects. For this purpose, the value of a state is defined as the expected value of the sum of discounted future rewards. Assuming a discounting factor $\gamma < 1$, the expression of the state value function can be written as:

$$V^{\pi,\gamma}(\mathbf{s}_t) = \mathbb{E} \left[\sum_{\tau=0}^{\infty} \gamma^\tau r_{t+\tau} \right] \tag{3.4.6}$$

The value function estimates how good is a state assuming to start from that state and reach a future one in a horizon determined by the discounting factor . In other words, it measures the quality of a possible trajectory. The discounting factor determines how greedy the Agent should be in the collection of high rewards. To put in another way, it quantifies how much the Agent should prioritize the immediate reward over the long term ones.
The eq. (3.4.6) can be expressed in a recursive form, which is more convenient for practical implementation:

$$V^{\pi,\gamma}(\mathbf{s}_t) = \mathbb{E}[r_t] + \gamma V^{\pi,\gamma}(\mathbf{s}_{t+1}) \tag{3.4.7}$$

However, the value function does not give information on how better the outcome of a certain action will be with respect to the prediction made at previous states. It only provides a measurement of the expected outcome, assuming to follow the current policy. Therefore, to compare whether the actions leads to a better or worse outcome, it is needed to introduce another function, called *advantage function*.

$$A^{\pi,\gamma}(\mathbf{s}_t, \mathbf{a}_t) = Q^{\pi,\gamma}(\mathbf{s}_t, \mathbf{a}_t) - V^{\pi,\gamma}(\mathbf{s}_t) \tag{3.4.8}$$

where $Q^{\pi,\gamma}(\mathbf{s}_t, \mathbf{a}_t)$ is the state-action value function, that estimates the value function starting from state $\mathbf{s}_t$ and selecting an action $\mathbf{a}_t$ according to a policy π, thus giving a measurement of the true value of each state. Therefore, from eq. (3.4.8) it is worse considering the value function $V^{\pi,\gamma}(\mathbf{s}_t)$ as the Agent's best guess for each state, according to its current policy.
To put it simply, the advantage function is positive if the selected action leads to a trajectory with better outcome with respect to the current policy followed by the Agent. Or, similarly, it quantifies how much better an action is compared to the expected outcome obtained by following the current policy.

The advantage $A^{\pi,\gamma}$ is not a quantity already determined. Therefore, it must be evaluated or, better to say, estimated.
In literature [33], there exists numerous techniques to estimate the advantage function, such as the Generalized Advantage Estimator. However, in this work, the advantage function is computed differently, as the difference between the empirical return and a state value function baseline. The equation is shown hereafter in eq. (3.4.9).

$$A^{\pi}_{w}(\mathbf{s}_t, \mathbf{a}_t) = \left[\sum_{\tau=0}^{\infty} \gamma^{\tau} r_{t+\tau}\right] - V^{\pi}_{w}(\mathbf{s}_t) \tag{3.4.9}$$

Where the state value function baseline $V^{\pi}_{w}(\mathbf{s}_t)$ is obtained as output of a neural network. For this reason, both the advantage function and the baseline depend on the parameters w of the neural network. For the sake of clarity, the expression of the cost function used to learn the value function V^{π}_{w} is reported in eq. (3.4.10). It consists in the expected value of the squared error between the prediction of the value function, output of the neural network, and the empirical return.

$$J^{VF}_{t}(w) = J(w) = \mathbb{E}_t\left[\left(V^{\pi}_{w}(\mathbf{s}_t) - \left(\sum_{\tau=0}^{\infty} \gamma^{\tau} r_{t+\tau}(\mathbf{s}_t, \mathbf{a}_t)\right)\right)^2\right] \tag{3.4.10}$$

3.4.4 Actor-Critic algorithm

The algorithm presented so far can be referred as an Actor-Critic algorithm. The Actor, which represents the policy gradient algorithm, learns the parameterized policy while the Critic learns and updates the value function, used to update the Actor's policy.
Actor-Critic methods consists in a simultaneous optimization of policy and value function, that are linked together thanks to the advantage function. As explained in [35], Actor-Critic algorithms show good convergence and are commonly used for reinforcement learning and meta-reinforcement learning algorithms.

As the Actor and the Critic have to learn, respectively, the policy and the value function, two neural networks are required. The Actor neural network produces the parameterized policy $\pi_\theta(\mathbf{a} \mid \mathbf{s})$, while the Critic calculates the value function, optimizing the objective function expressed in eq. (3.4.10), which is the squared-error between the true sum of discounted rewards and the Critic's assessment V_w^π. The rewards collected after a certain number of episodes are batched together in roll-outs and the two networks are simultaneously updated to optimize their objective function. For the sake of completeness, the two objective functions of the Actor and the Critic are also reported here below:

$$\begin{aligned} J_t^{PG}(\theta) &= J_t(\theta) = \mathbb{E}_t\left[\sum_{t=0}^{\infty} \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) A_w^\pi(\mathbf{s}_t, \mathbf{a}_t)\right] \\ J_t^{VF}(w) &= J_t(w) = \mathbb{E}_t\left[\left(V_w^\pi(\mathbf{s}_t) - \left(\sum_{\tau=0}^{\infty} \gamma^\tau r_{t+\tau}(\mathbf{s}_t, \mathbf{a}_t)\right)\right)^2\right] \end{aligned} \tag{3.4.11}$$

In addition, gradient ascent is performed on θ and gradient descent on w. The update equations are:

$$\begin{aligned} w_{t+1} &= w_t - \alpha_w \nabla_w J(w_t) \\ \theta_{t+1} &= \theta_t + \alpha_\theta \nabla_\theta J(\theta_t) \end{aligned} \tag{3.4.12}$$

Where α_w and α_θ are respectively the learning rates of the value function and policy neural networks. Note that gradient descent optimization algorithms are explained in detail in Sec. 4.3.

3.5 Proximal Policy Optimization (PPO)

As stated in Sec. 3.3, Meta-RL aims at learning a prior-dependent RL algorithm, with the prior being a distribution over MDPs. Reinforcement learning algorithms have an intrinsic pathology. In particular, the fact that generated data used for training are themselves dependent on the current policy causes instability in the learning process. In other words, as the Agent learns, the data distribution over observations and rewards constantly changes. If the policy update is too large, the policy network could be brought into a region of the parameters space that corresponds eventually to a very poor policy. As a consequence, the next training dataset, collected following the current bad policy, would trap the policy network in that region, causing it to never recover again and follow a completely wrong policy. This can be translated in the Agent losing its understanding

of the environment.
This can be solved simply by limiting the updates size of the policy.

3.5.1 Trust Region Method

Trust Region Policy Optimization (TRPO) [36] algorithm follows this approach. It implements a "surrogate" constrained objective function.

$$J_t^{TRPO}(\theta) = \mathbb{E}_t \left[\frac{\pi_\theta(\mathbf{a}_t, \mathbf{s}_t)}{\pi_{\theta_{old}}(\mathbf{a}_t, \mathbf{s}_t)} A_w^\pi(\mathbf{s}_t, \mathbf{a}_t) \right] \tag{3.5.1}$$

The constraint is introduced through the so-called *Kullback-Leibler divergence*, which is a measure of the difference between two probability distributions. The constraint, formalized in eq. (3.5.2), obliges the policy update to do not move far away from the current policy.

$$\mathbb{E}_t[\mathcal{KL}[\pi_{\theta_{old}}(. \mid \mathbf{s}_t), \pi_\theta(. \mid \mathbf{s}_t)]] \leq \zeta \tag{3.5.2}$$

In both equations, the term $\pi_{\theta_{old}}$ refers to the old vector of policy parameters. It is then clear that now the update between two consecutive steps of the policy network parameters is reduced in size. The TRPO algorithm aims at maximizing the new surrogate objective, that differs from the objective function in eq. (3.4.5) for the term $\frac{\pi_\theta(\mathbf{a}_t, \mathbf{s}_t)}{\pi_{\theta_{old}}(\mathbf{a}_t, \mathbf{s}_t)}$. This ratio is the probability of an action occurring following the new policy π_θ divided by the probability of the same action to occur under the old policy. The maximization of the new objective function can be translated into the maximization of this ratio of probabilities for a given action and of the associated advantage function. The effects of this optimization can be understood with an example. Imagine to select an action that leads to a terrible result. For this reason, the probability to take this action in the new policy should be lower with respect to the previous policy, thus the ratio becomes lower than 1. The advantage function, at the same time, will be negative for a non convenient action. Therefore, maximizing the objective function $J_t^{TRPO}(\theta)$ will produce the least negative product between the ratio and advantage and, as a consequence, will reduce the probability to retake the same action according to the new policy.

3.5.2 Clipped Surrogate Objective

The TRPO algorithm is very complex to implement. For this reason, in [37] an algorithm inspired to the Trust Region Methods has been developed. The Proximal Policy Optimization (PPO) inherits some of the benefit of the TRPO, but in a much simpler version to implement. A new objective function is introduced in the PPO algorithm: it is the expectation of the minimum between two terms, where the first one is the ratio-advantage product, as in the TRPO surrogate objective function, and the second one is a truncated version of the policy ratio obtained by a clipping operation.

$$J_t^{PPO}(\theta) = \mathbb{E}_t \left[min(r_t(\theta) A_{w,t}^\pi, clip(r_t(\theta), 1-\epsilon, 1+\epsilon) A_{w,t}^\pi) \right] \tag{3.5.3}$$

where $r_t(\theta)$ is the ratio between the new and old policy at time step t and ϵ is an hyperparameter. The selection of the minimum term heavily depends on the sign assumed by the advantage function. If, for example, the advantage function is positive, the optimization tends to increase the probability under the new policy to take the action with positive advantage. However the clipping parameter ϵ limits the change in probability. The same happens if the advantage function is negative but with the opposite result.

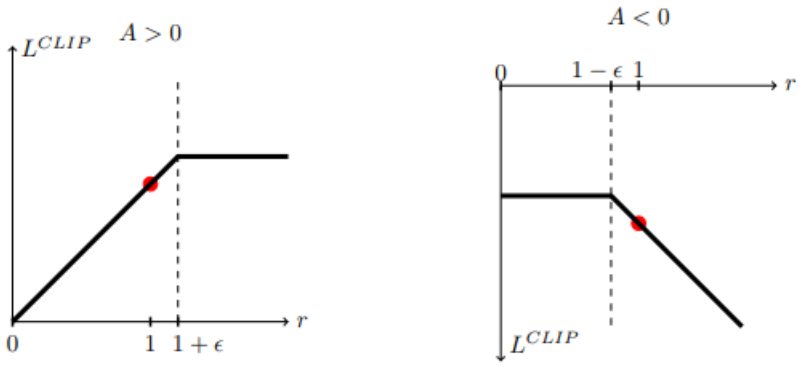


Figure 3.5.1: *Plots showing a single time step of the clipped surrogate function* $L^{CLIP} = J^{PPO}$ *behavior with respect to the probability ratio* r *for positive advantages (left) and negative advantages(right). The red circle on each plot shows the starting point of the optimization, i.e.* $r = 1$. Image taken from [37]

3.5.3 KL Adaptivity

In addition to the clipped surrogate objective, this work consider the Kullback-Leibler (KL) divergence in the policy optimization algorithm. Differently from [37], where the KL divergence is also introduced in the objective function as a penalty term, here it is considered as a control term for the clipping parameter, as done in [20]. This technique, named *servo-kl* controls the value of ϵ comparing the actual KL divergence with a target one, set as a hyperparameter.

The KL-divergence is computed in an approximate and biased form, used by [38] and reported in eq. (3.5.4).

$$\mathcal{KL}[\pi_\theta, \pi_{\theta_{old}}] = \frac{1}{2}(log\pi_\theta - log\pi_{\theta_{old}}) \tag{3.5.4}$$

Then, after selecting a maximum and minimum value for ϵ and a target KL divergence, the control procedure is as follows:

$$\begin{aligned} \epsilon &= min(\epsilon_{max}, 1.5\epsilon) \quad if \quad KL_t < \tfrac{KL_{target}}{2} \\ \epsilon &= max(\epsilon_{min}, \tfrac{\epsilon}{1.5}) \quad if \quad KL_t > 2Kl_{target} \end{aligned} \tag{3.5.5}$$

At the same time also the learning rate of the policy neural nets are modified to keep the KL divergence near the target one.

3.6 Algorithm

This final section tries to put together all the notions and methods discussed throughout this chapter. For the sake of clarity, the whole algorithm is presented hereafter.

Algorithm 1 PPO algorithm in Meta-RL

Initialization of neural network parameters θ and w
Initialization of the scalers
for episode $= 1, 2, ..., E$ **do**
    Reset the environment
    Generate new environment
    while not done **do**
        Sample $\mathbf{a}_t$ from $\pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t)$ ▷ Select action
        $s_{t+1}; r_{t+1}; done \leftarrow Env(\mathbf{s}_t; \mathbf{a}_t)$ ▷ Interaction with the environment
        Store the roll-outs
        Update the scaler
        for epoch = 1,2, ..., K **do**
            Unroll the recurrent layer of each network ▷ Forward pass
            Compute V_w^π and A_w^π
            Compute Actor objective function J_t^{PPO}
            Compute Critic objective function J_t^{VF}
            Perform a gradient ascent on the Actor parameters θ
            Perform a gradient descent on the Critic parameters w
            Adjust clipping parameter and learning rates to target a KL divergence

Chapter 4

Artificial Neural Networks

Artificial Neural Networks (ANNs) are a powerful tool to approximate complex, non-linear functions without any knowledge of the function itself, but only based on input and output data. Because of their wide range of applications, ANNs are commonly used in deep machine learning.

Artificial Neural Networks are a machine learning algorithm that tries to replicate the structure of the human brain. It consists in millions of neurons, connected one to each other through the synapses. Similarly, Artificial Neural Networks are made up of interconnected units, commonly known as neurons, capable of exchanging and processing information and grouped into layers. In deep learning, a variety of neural networks has been developed, with different structures and thus different properties. In this work two types of ANNs are considered, in particular:

- *Feed-Forward Neural Network (FFNN)*, where input data is processed only in the forward direction.
- *Recurrent Neural Network (RNN)*, needed for any Meta-RL algorithm. In particular, a recurrent connection on the hidden units is added with respect to FFNN. This allows the network to propagate data from earlier events to current processing steps, building so a memory of time series events.

4.1 Feed-Forward Neural Networks

In Feed-Forward Neural Networks (FFNNs), sets of neurons are organised in layers, where each neuron computes a weighted sum of its inputs. Input neurons take signals from the environment and the output ones provide signals to the environment. In between them, there exist multiple layers of neurons, called *hidden* neurons.
Feed-forward neural networks are loop-free and fully connected. In other words, each neuron provides an input to each other neuron in the following layer, only in the forward direction. An example of a FFNN is the *single-layer perceptron* network. This kind of neural network consists of a set of input neurons, defined as the *input layer*, and a set of output neurons, defined as *output*

layer. In this case, since no layers are present between the input and the output, the neurons of the two layers are directly connected between each other. This means that the outputs of the input-layer are sent to the neurons of the output layer, passing through the connections, known as arcs. The weights are applied to the arcs that connect input and output layers.
If several layers are added between the input and output ones, the neural network becomes a multilayer forward connected network. Here, the input and output layers are connected via at least one hidden layer, built from sets of hidden neurons [39]. In figure Fig. 4.1.1 a FFNN is sketched, with two hidden layers and one input and output layers.

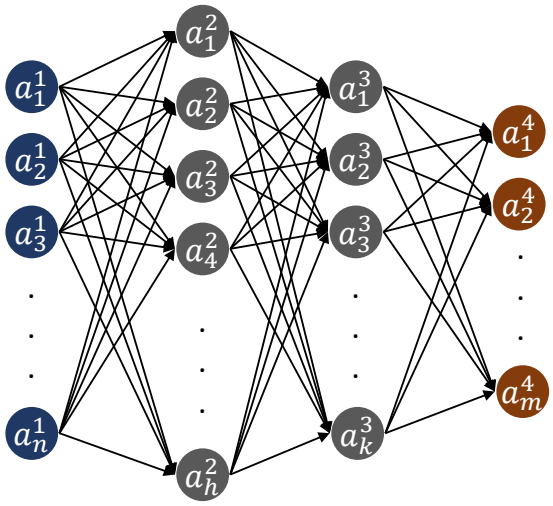


Figure 4.1.1: *Graphical representation of a Feed Forward Neural Network with an input layer of n-neurons (dark blue), two hidden layers of j-neurons and k-neurons respectively (dark grey) and an output layer of m-neurons (dark red).* Taken from [23].

4.1.1 Forward pass and Backpropagation

Neurons are semi-linear units that are represented by scalar values named *activations*, detoned by *a*. The activations are the results of a non-linear function σ, defined as *activation function*, that receives as an input a weighted sum computed from a weight matrix $\mathbf{w}$ and a bias matrix $\mathbf{b}$. Each connection between neurons, the arcs, is characterized by a weight [23].
The forward pass consists in the computation of the neurons' activation of layer l, using as inputs the activations of the previous layer $l-1$. Mathematically:

$$\begin{aligned} \mathbf{p}^l &= \mathbf{w}^l \mathbf{a}^{l-1} + \mathbf{b}^l \\ \mathbf{a}^l &= \sigma(\mathbf{p}^l) \end{aligned} \tag{4.1.1}$$

Where $\mathbf{p}$ is the weighted sum and it is defined as *preactivation*. For what regards the *activation function* σ, it must be a non-linear function, in order to correctly approximate a non-linear function. Typically, the most used activation functions are *sigmoid functions*, such for example *tanh*, or the

Rectified Linear Unit (ReLU).

For the sake of clarity, eq. (4.1.1) is written referred to each layer. In order to better describe the forward pass, it could be useful to write the previous equation referred to each neuron r per layer l. In particular:

$$\begin{aligned} p_r^l &= \textstyle\sum_s w_{rs}^l a_s^{l-1} + b_r^l \\ a_r^l &= \sigma(p_r^l) \end{aligned} \tag{4.1.2}$$

Considering now that the ultimate goal of a generic reinforcement learning algorithm is to optimize a cost function. The training of a neural network consists in the update of the network parameter to better describe the non-linear function. Before doing so, it is necessary to understand how changing these parameters in a network influences the cost function mentioned before. This is what is called *backpropagation*.

Assume a case of single training example of a Feed Forward Neural Network, made of L layers, with an objective function J. The goal of backpropagation is to determine the sensitivity of the objective function J with respect to the network's parameters. In other words, backpropagation consists in computing partial derivatives of the cost function with respect to weights and biases.

In particular, backpropagation, as the name suggests, starts from the outputs and moves back through the network. In layer notation only, consider the last layer, characterised by the activation $\mathbf{a}^L$ and preactivation $\mathbf{p}^L$. Applying the chain rule and equations eq. (4.1.1):

$$\frac{\partial J}{\partial \mathbf{p}^L} = \frac{\partial J}{\partial \mathbf{a}^L}\frac{\partial \mathbf{a}^L}{\partial \mathbf{p}^L} = \frac{\partial J}{\partial \mathbf{a}^L}\sigma'(\mathbf{p}^L) \tag{4.1.3}$$

According to eq. (4.1.3), the the rate of change of the objective function with respect to the preactivation of neurons is influenced by the rate of change of the activation function $\sigma'(\mathbf{p}^L)$ and the sensitivity of the cost function to the neuron output, the activation function $\mathbf{a}^L$.
Now, propagating backwards to the input layer, it is possible to evaluate the sensitivity of the objective function to small changes in the network parameters. Hence, for an arbitrary layer l:

$$\frac{\partial J}{\partial \mathbf{p}^l} = \frac{\partial J}{\partial \mathbf{p}^{l+1}}\frac{\partial \mathbf{p}^{\mathbf{l+1}}}{\partial \mathbf{a}^l}\frac{\partial \mathbf{a}^l}{\partial \mathbf{p}^l} = \frac{\partial J}{\partial \mathbf{p}^{l+1}}\mathbf{w}^{l+1}\sigma'(\mathbf{p}^l) \tag{4.1.4}$$

And finally, considering eq. (4.1.1) and differentiating the cost function for the weights and biases:

$$\frac{\partial J}{\partial \mathbf{w}^l} = \frac{\partial J}{\partial \mathbf{p}^l}\frac{\partial \mathbf{p}^l}{\partial \mathbf{w}^l} = \frac{\partial J}{\partial \mathbf{p}^l}\mathbf{a}^{\mathbf{l-1}} \tag{4.1.5}$$

$$\frac{\partial J}{\partial \mathbf{b}^l} = \frac{\partial J}{\partial \mathbf{p}^l}\frac{\partial \mathbf{p}^l}{\partial \mathbf{b}^l} = \frac{\partial J}{\partial \mathbf{p}^l} \tag{4.1.6}$$

The same equations can be derived at the scale of each neuron. Starting from eq. (4.1.2) and applying the chain rule, a set of equations similar to the ones above is obtained. However, the

derivation will not be explicitly written. Therefore, the rate of change of the cost function with respect to an arbitrary neuron r in the generic layer l is:

$$\frac{\partial J}{\partial p_r^l} = \sum_s \frac{\partial J}{\partial p_s^{l+1}} w_{sr}^{l+1} \sigma'(p_r^l) \tag{4.1.7}$$

The sensitivity of the objective function to changes in the weights and biases, in neuron scale, is:

$$\frac{\partial J}{\partial w_{rs}^l} = \frac{\partial J}{\partial p_r^l} a_s^{l-1} \tag{4.1.8}$$

$$\frac{\partial J}{\partial b^l} = \frac{\partial J}{\partial p_r^l} \tag{4.1.9}$$

4.2 RNN: Long Short-Term Memory (LSTM)

As already mentioned in chapter 4, Recurrent Neural Networks (RNN) are needed for any meta-rl algorithm. In principle, recurrent networks can use their feedback connections to store representations of recent input events in form of activations, where these feedback connections are possible due to an inner loop that allows the information to persist. Looking at Fig. 4.2.1, the yellow block, representing a portion of a neural network **A**, receives as input x_t and outputs a value h_t. The loop allows information to be passed from one step of the network to the next. For a clear view of how data is exchanges, suppose to unfold the loop obtaining a sequence of multiple copies of the same network, each passing information to a successor, such as in Fig. 4.2.2.

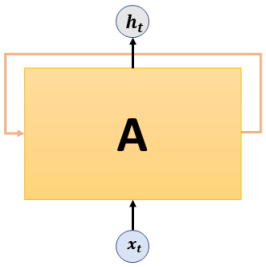


Figure 4.2.1: *Inner loop of a Recurrent Neural Network*

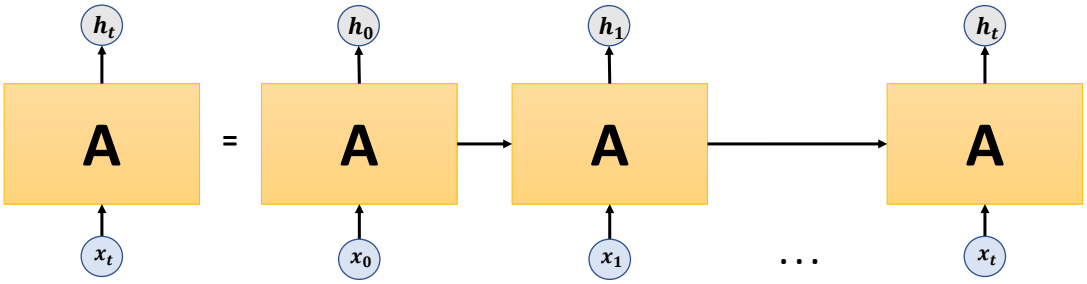


Figure 4.2.2: *Unrolled Recurrent Neural Network*

One of the major drawbacks of a standard Recurrent Neural Network is directly linked with their property of memorizing and exchanging information. In particular, during training of the network, which is different from the case of a FFNN, it is needed to propagate information through the recurrent connections in-between steps. Without going into much details of the learning techniques of standard recurrent networks, the most common learning algorithm is he *backpropagation through time (BPTT)* [39]. In BPTT, the network is unfolded in time to construct a Feed-Forward Neural Network and standard backpropagation is executed. However, a problem arises. The calculated gradients and the signal error that flow backward in time tend to blow up or vanish. These phenomena are called exploding and vanishing gradients problem, respectively, and are analysed and discussed in [40].

To overcome these error back-flow problems, a particular kind of recurrent neural network is adopted in this work. In particular, the so-called Long Short-Term Memory (LSTM) solves these problems. These gradient-based neural networks are able to learn long-term dependencies, meaning that they are able to store relevant information and forget unnecessary data.
Before illustrating how the gradient is backpropagated in a LSTM, it is imperative to explain its architecture.

4.2.1 LSTM Architecture

LSTMs have a chain-like structure of repeating modules, as for any RNN, but the module has a particular structure. It consists of three parts and each part performs an individual function. As it can be seen in Fig. 4.2.3, the first part chooses whether the information coming from previous timestamp is relevant and thus to be remembered or it can be forgotten. This part is known as *Forget gate*. The second part, instead, is used by the cell to learn new information from the inputs, and it is referred to as *Input gate*. Finally, the third part is the *Output gate*, that passes updated information from the current timestamp to the next one. Similarly to a standard RNN, an LSTM has also *hidden state*, represented in the figure by H_{t-1}, for the previous timestamp hidden state, and H_t for the current one. In addition to that, LSTM have a *cell state*, indicated in the figure as

C_{t-1} and C_t. The name of this kind of recurrent networks comes exactly from the role played by these two states: the *hidden state* is known as *Short term memory* and the *cell state* as *Long term memory*.

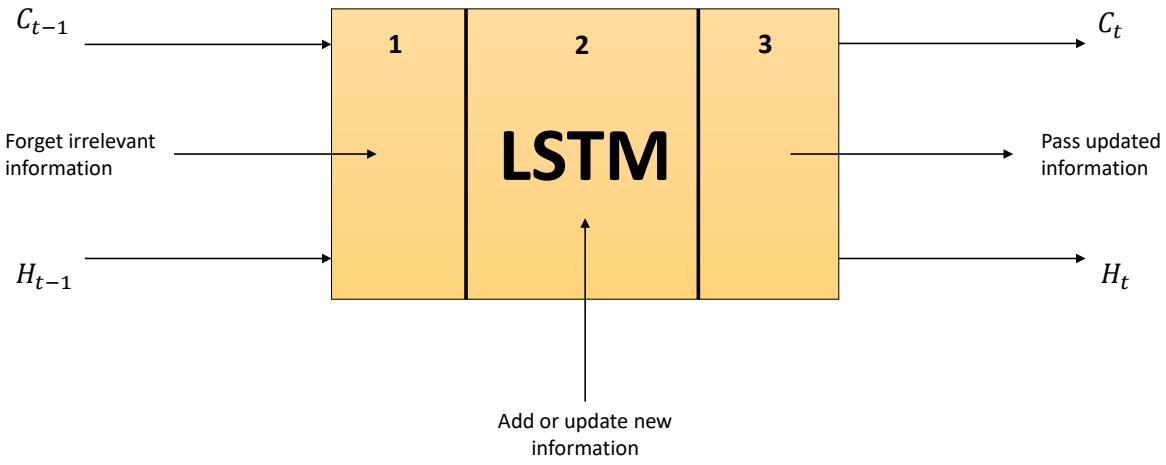


Figure 4.2.3: *Common subdivision of an LSTM module into three main parts, respectively the Forget gate, the Input gate and the Output gate. Each of them plays a particular and unique role.*

It may be useful to break down the architecture of the LSTM, having an detailed description of the information flow through each gate.
Information is carried by the cell state, that passes through each of the three gates. Indeed, the ability of the LSTM to remove or add information to the cell state relies upon the role played by each gate. A common representation of a LSTM module is showed in Fig. 4.2.4, where the cell state is represented by the horizontal line running through the top of the block. In the same figure, gates can be recognized as the composition of a *sigmoid* neural network layer and a pointwise multiplication operation. The output of the sigmoid layer is a number between 0 and 1. This enables the LSTM to decide whether let information pass through the gates or forget it.

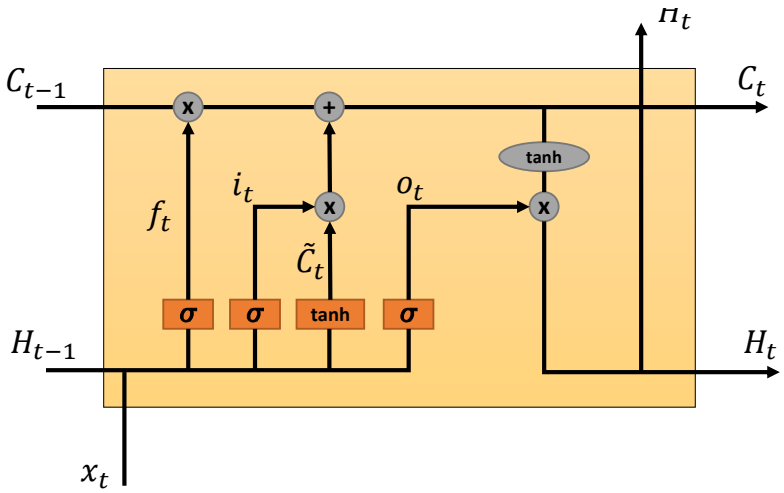


Figure 4.2.4: *Inner operations of an LSTM, such as pointwise operations (grey ellipses) and neural network layers (orange boxes). As before, C_t and H_t are, respectively, the cell state and the hidden state.*

4.2.2 Forward pass

For understanding how the forward pass is executed in a LSTM, it is now convenient to break down also the operations made by each gate to the *cell state*, the *hidden state* and the input x_t.

- *Forget gate.* It is made by the first sigmoid layer and a pointwise multiplication, and decides what information can be ignored Fig. 4.2.5. It receives as inputs the *hidden state* of the previous time-step H_{t-1} and the input from the environment x_t. It then outputs a number between 0 and 1, thanks to the sigmoid layer, for each number in the cell state C_{t-1}. In other words, if the information is worth to be remembered, the output is 1. Viceversa, if this is not the case, the output is 0. Mathematically, the operation made by the *forget gate* is:

$$f_t = \sigma(W_f[H_{t-1}, x_t] + b_f) \tag{4.2.1}$$

  where W_f and b_f are the weight and bias associated with the sigmoid neural network layer.

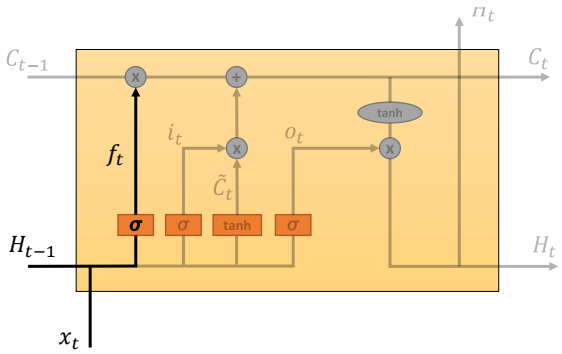


Figure 4.2.5: *Forget gate of a LSTM*

- *Input gate*. It decides what new information will be stored in the cell state. In the input gate two main operations are executed. First, the *hidden state* H_{t-1} and the external input x_t flow through a sigmoid layer which determines the values to update. Next, a tanh layer creates a vector of values, $\tilde{C}_t$, to be eventually added to the state. From the combination of the two outputs from the sigmoid and tanh layers, it is possible to generate an update to the *cell state*. The operations i_t and $\tilde{C}_t$, made in this gate, can be seen in eq. (4.2.2), where W_i and b_i are weight and bias associated with the sigmoid layer, while W_C and b_C are the ones associated with the *tanh* layer. A graphical representation is shown in Fig. 4.2.6.

$$\begin{aligned} i_t &= \sigma(W_i[H_{t-1}, x_t] + b_i) \\ \tilde{C}_t &= \tanh(W_C[H_{t-1}, x_t] + b_C) \end{aligned} \tag{4.2.2}$$

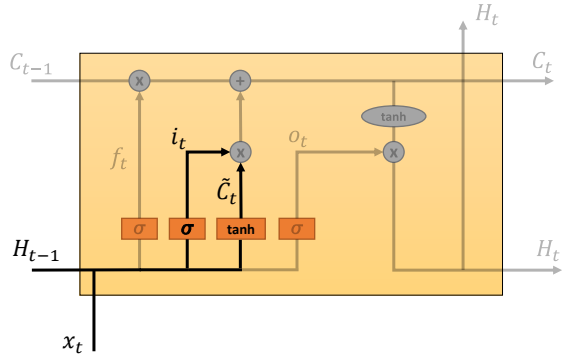


Figure 4.2.6: *Input gate of a LSTM*

Thanks the outputs of the *input gate* and *forget gate*, it is now possible to update the old cell state cell C_{t-1} into the new cell state C_t. The update is performed as indicated in eq. (4.2.3) and shown in Fig. 4.2.7.

$$C_t = f_t C_{t-1} + i_t \tilde{C}_t \tag{4.2.3}$$

In particular, from eq. (4.2.3), it can be seen that the old cell state is multiplied by the output of the *forget gate*. This means that part of the information is discarded. Then, a new update, scaled by the output of the sigmoid layer in the *input gate*, is added to the modified old cell state. As a result, the new cell state C_t is computed.

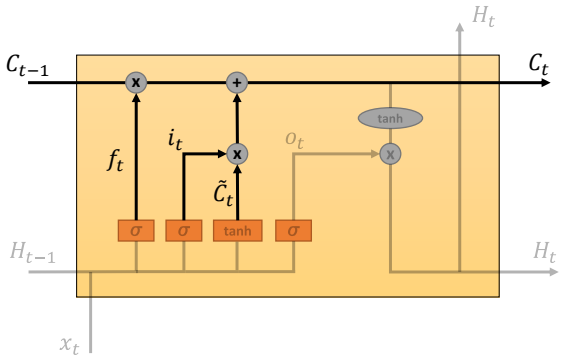


Figure 4.2.7: *Update process of the cell state of the previous time-step, C_{t-1}.*

- *Output gate.* It provides a filtered version of the cell state as output. First, a sigmoid layer decides what parts of the cell state will enter in the outputs of the gate. Before the multiplication with the sigmoid layer output, the updated cell state passes through a *tanh* layer, that clips the cell state values between -1 and 1. The procedure is formalized in eq. (4.2.4), where W_o and b_o are the biases associated with the sigmoid layer in the output gate. In Fig. 4.2.8, the flow of the information is illustrated. A new hidden state exits from the *output gate* and it will be fed into a new memory cell.

$$\begin{aligned} o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \\ h_t &= o_t \tanh(C_t) \end{aligned} \tag{4.2.4}$$

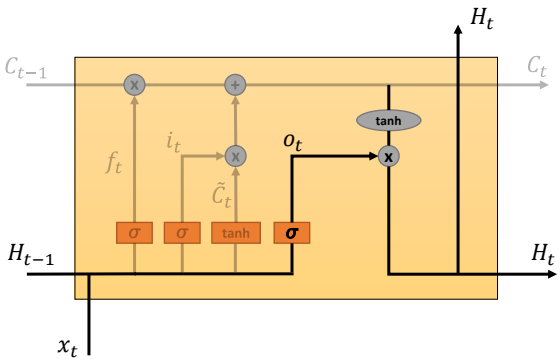


Figure 4.2.8: *Output gate of a LSTM.*

4.2.3 Backpropagation

The backpropagation of a LSTM is not straightforward as in the case of the Feed-Forward Neural Networks, due to the complex interaction of its layers. Nevertheless the approach still consists in the identification of the dependencies and the application of the chain rule, obtaining the sensitivity of a cost function J with respect to the weights and biases.

4.3 Gradient descent optimzation through Adam Optimizer

The gradient of the cost function obtained through backpropagation needs to be optimized in last phase of the learning process of the neural networks, for both FNN and LSTM. In this work, Adam Optimizer is selected as the algorithm of gradient descent optimization. However, as explained from the creators of this algorithm in [41], an adequate and brief overview of other gradient descent algorithms [42] is needed to fully understand how Adam Optimizer works.

4.3.1 Vanilla Gradient Descent and Stochastic Gradient Descent (SGD)

Gradient descent is a technique aimed at the minimization of an objective function $J(\theta)$, parameterized by model parameters θ. These parameters are updated in the opposite direction of the gradient of the objective function $\nabla_\theta J(\theta)$ with a step size η called *learning rate*. This process, that ends when the gradient becomes null, leads eventually to a local optimal solution based on the batch of data used for the evaluation of the gradient.

The most common gradient descent method is the so-called Batch Gradient Descent or Vanilla Gradient Descent. The entire dataset is used as a batch for the descent, as expressed in eq. (4.3.1). Typically, this method is slow and does not allow the update of the dasatet with new information, thus it can be distinguished as an offline method. In the context of a reinforcement learning

algorithm, its use is not acceptable.

$$\theta = \theta - \eta \nabla_\theta J(\theta) \tag{4.3.1}$$

On the other hand, Stochastic Gradient Descent (SGD) is a variation of the Vanilla Gradient Descent, well-suited for neural network approximation. In particular, the update of the parameters is not performed for the entire dataset as before, but for randomly selected data point x^i and label y^i:

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^i; y^i) \tag{4.3.2}$$

Compared to Batch gradient descent, that performs redundant computations in case of large datasets, SGD executes one update at a time. Therefore, it is usually much faster and can be used to learn online. However, the use of single data for the updates increases the variance and leads to a fluctuating objective function, that allows a better exploration of the domain at the cost of worst performances in terms of convergence to the exact minimum.

By mixing the two previous methods, Mini-batch gradient can achieve a more stable convergence to the local minima and a very efficient computation of the gradients. In particular, it performs an update for every mini-batch of n training examples:

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^{i:i+n}; y^{i:i+n}) \tag{4.3.3}$$

Typically, mini-batch gradient descent is one of the most chosen algorithm for the training of a neural network. Furthermore, when dealing with mini-batch gradient descent, it is common to use the term SGD to refer to this method.

4.3.2 Gradient descent optimization algorithms

Stochastic gradient descent suffers when dealing with surfaces that shows a higher steepness in one dimension than in another [43], often called as *ravines*. This areas are usually close to the local optima. The SGD tends to oscillate across the slope of the ravine without making particular process on the descent. To face this issue, adding a momentum term [44] dampens the oscillations and accelerates the descent in the relevant direction, towards the local optimum. In particular, this is achieved by adding a fraction γ of the update vector of the past mini-batch to the current update vector. In other words:

$$\begin{aligned} \mathbf{v_t} &= \gamma \mathbf{v}_{t-1} + \eta \nabla_\theta J(\theta) \\ \theta &= \theta - \mathbf{v}_t \end{aligned} \tag{4.3.4}$$

The eq. (4.3.4) can be explained as follows. The momentum term increases for those dimensions whose gradients point always in the same direction and, instead, reduces the updates of those dimensions whose gradients often change direction. As a consequence, the convergence is faster and the fluctuation are reduced. The momentum-based methods, however, do not take into

consideration the shape of the surface and, while multiple parameters are being updated, the learning rate remains fixed and common for each parameter.

To cope with these limitations, an adaptive algorithm has been developed. Known as *Adagrad* [45], this algorithm adapts the learning rate to the parameters, performing larger updates for rare parameters and smaller updates for more frequent ones. Thus, the learning rate is increased when infrequent parameters occur and it is reduced for those parameters that appear more frequently. As a result, this algorithm is well-suited for dealing with sparse datasets.

According to the Adagrad method, the gradient of the objective function $J(\theta)$ can be expressed singularly for each parameter θ_i, at time step t as:

$$g_{t,i} = \nabla_{\theta_t} J(\theta_{t,i}) \tag{4.3.5}$$

Consequently, the SGD update for the $i-th$ parameter at each time step t results in:

$$\theta_{t+1,i} = \theta_{t,i} - \eta g_{t,i} \tag{4.3.6}$$

In eq. (4.3.6), the learning rate η is adjusted at every time step t for each parameter θ_i, based on the past gradients computed for the same parameter. Hence:

$$\begin{aligned} \eta &= \frac{\eta}{\sqrt{G_{t,ii}+\epsilon}} \\ \theta_{t+1,i} &= \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii}+\epsilon}} \times g_{t,i} \end{aligned} \tag{4.3.7}$$

Where, $\mathbf{G}_t$ is a diagonal matrix where each diagonal element (i,i) is the sum of the square of the gradients, computed with respect to all parameters θ up to the current time step t. The other term in the expression of the learning rate update, is ϵ, a smoothing term that avoids zero on the denominator.

The main benefit of Adagrad is the peculiar property of automatically adjust the learning rate. However, in order to do so, the update of η requires to accumulate squared gradients in the denominator. This causes a gradually increasing sum on the denominator itself that tends to shrink the learning rate, which eventually becomes infinitesimally small. As a consequence, the algorithm is no longer able to learn and no more updates are performed.

There exist different algorithms that solve this flaw. One of them is worth mentioning, due to the connection with the Adam Optimizer. The *RMSprop* is an unpublished, adaptive learning rate method [46]. It is inspired to another algorithm, called *Adadelta* [47], that restricts the window of accumulated past gradients to a fixed size. Differently from Adagrad, the sum of these gradients is recursively defined as a decaying average of all past squared gradients eq. (4.3.8). The same approach is followed in the RMSprop algorithm.

$$\begin{aligned} \mathbb{E}[\mathbf{g}^2]_t &= \gamma \mathbb{E}[\mathbf{g}^2]_{t-1} + (1-\gamma)\mathbf{g}_t^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\mathbb{E}[\mathbf{g}_t + \epsilon}} \mathbf{g}_t \end{aligned} \tag{4.3.8}$$

In particular, in the RMSprop the value suggested for γ is 0.9.

4.3.3 Adam Optimizer

Adaptive Moment Estimation, also known as *Adam*[41] is an algorithm that computes adaptive learning rates for each parameter, as done by Adagrad and RMSprop. However, in addition to storing a decaying average of past *squared* gradients $\mathbf{v}_t$, Adam algorithm keeps track of a decaying average of past gradients $\mathbf{m}_t$, similar to momentum:

$$\begin{aligned} \mathbf{m}_t &= \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t \\ \mathbf{v}_t &= \beta_2 \mathbf{v}_{t-1} + (1-\beta_2)\mathbf{g}_t^2 \end{aligned} \tag{4.3.9}$$

In eq. (4.3.9), β_1 and β_2 are the decaying rates applied at the estimates of the first moment $\mathbf{m}_t$, the mean, and the second moment $\mathbf{v}_t$, the variance, of the gradients, respectively. In [41], when the two estimates of the moments of the gradients are initialized as zeros, they are biased towards zero. A solution is proposed by the authors of [41] to counteract these biases. In particular, it is possible to correct the two estimates computing a bias-corrected version as follows:

$$\begin{aligned} \hat{\mathbf{m}}_t &= \frac{\mathbf{m}_t}{1-\beta_1^t} \\ \hat{\mathbf{v}}_t &= \frac{\mathbf{v}_t}{1-\beta_2^t} \end{aligned} \tag{4.3.10}$$

These bias-corrected versions are then used to update the parameters as it is done in Adagrad or RMSprop. Adam update rule then results in:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon} \hat{\mathbf{m}}_t \tag{4.3.11}$$

It is worth adding that the decaying rates are set to default as 0.9 and 0.999, as suggested by the authors of [41]. The same is applied for the smoothing hyperparameter ϵ, which is taken as 10^{-8}. The only hyperparameter to manually select is the learning rate, that will be specified in chapter 5.

Chapter 5

Problem formulation through Meta-RL

In the previous three chapters, the problem formulation (chapter 2) and the mathematical framework (chapter 3, chapter 4) have been set up. This chapter tries to join the contents discussed previously, focusing on the implementation of the Meta-Reinforcement learning algorithm, strongly inspired by [20] and taking into consideration [23] for the adaptation of the Autonomous Rendezvous Proximity Operation and Docking problem as a Markov Decision Process. In particular, the state space, action space and the discretization of the dynamics are taken from [23]. The reward function implemented to teach the Agent is described in a proper section. The last section points out the Agent's hyperparameters manually selected for this work. The chapter 6 will specify more the reward function and the hyperparameters for each case considered in this work.

5.1 ARPOD as a Markov Decision Process

Markov Decision Processes are defined in Sec. 3.2. Recalling the notation used previously, the state space $\mathcal{S}$ and the action space $\mathcal{A}$ of the ARPOD problem seen as a Markov Decision Process are specified hereafter.

5.1.1 State Space $\mathcal{S}$

For the ARPOD problem to be considered as a MDP, the Agent's state has to satisfy the Markov Property, discussed in Sec. 3.2. This means that the transition to the successive state has to be function only of the present state and action. In other words, the state of the Agent must contain enough information for the satisfaction of this requirement, without providing unnecessary information that would end up slowing the learning process.
The dynamics of the Chaser Sec. 2.3, which represents the Agent in this application of Meta-RL to the ARPOD problem, can be fully described by its position, velocity, attitude and angular velocity. Hence, the state can be formalized as:

$$S_t = \left[x_t, y_t, \dot{x}_t, \dot{y}_t, \theta_{N_t}, \dot{\theta}_{N_t}\right] \tag{5.1.1}$$

As done in [23], for an efficient backprogation [48], the state variables given as input to the neural networks are scaled. The scalers considered in this work are inspired from [20] and they are evaluated considering the mean and the variance of each state variable. In particular, the following scaling is adopted:

$$\begin{aligned} s_f &= \frac{1}{\sqrt{var(s_i)}} + \frac{0.1}{3} \\ \tilde{x}_i &= s_f(x_i - \mu(s_i)) \end{aligned} \tag{5.1.2}$$

Where s_f is the scaling factor, $var(s_i)$ is the variance of each state variable s_i and $\mu(s_i)$ is its mean value.
Therefore, the Chaser state S_t fed as input for the neural networks learning process is denoted as $\tilde{S}_t$.

$$\tilde{S}_t = \left[\tilde{x}_t, \tilde{y}_t, \tilde{\dot{x}}_t, \tilde{\dot{y}}_t, \tilde{\theta}_{N_t}, \tilde{\dot{\theta}}_{N_t}\right] \tag{5.1.3}$$

5.1.2 Action Space $\mathcal{A}$

Before illustrating the action space considered for the solution of this problem, it is imperative to recall that the Markov decision process requires, in this case, a discrete-time model. For this reason a new assumption is added to the previous ones, made in Sec. 2.3.

Assumption 6 *Actions from the actuators are instantaneous impulses that are provided at the beginning of each time step*

In this work, as done in [23], the actions that the Chaser can execute consist in a velocity impulse ΔV, given through the thruster, and a commanded acceleration $\dot{\psi}$ to the reaction wheel, that are given as outputs from the Actor neural network. Furthermore, the Actor network uses a *tanh* activation function in the second last layer, which clips the mean of each distribution outputted between -1 and 1. However, the standard deviation of the distribution could bring these values out of the range. For this purpose, the outputs are clipped again to enter inside the range.
Denoting now the real Actor neural network outputs as $\widetilde{\Delta V} \in [-1; 1]$ and $\widetilde{\dot{\psi}} \in [-1; 1]$. The output of the Actor network can be formalized as:

$$\tilde{A}_t = \left[\widetilde{\Delta V}_t, \widetilde{\dot{\psi}}_t\right] \tag{5.1.4}$$

In [23], a convenient strategy is adopted to ease the learning process. In particular, the action space $\mathcal{A}$ is restricted through the definition of lower and upper boundaries, decreasing its size and reducing the computational time. Moreover, the restricted action subspace results to be time

dependent and defined at each state. In this work, the same strategy is followed.

In the action space, two more subspaces can be identified and distinguished: the reaction wheel acceleration space and the thruster impulse space. It is worth mentioning that these two subspaces are not two separate entities. In particular, due to the dynamics of the problem, they are linked in the following way: at each time step t the reaction wheel commands an acceleration impulse to change the attitude of the Chaser and then, with the new orientation, the thruster gives a velocity impulse that brings the Chaser in a new position.

Reaction Wheel Acceleration Space

The reaction wheel uniquely controls the attitude of the Chaser. Recalling that it is necessary to respect the attitude constraints formalized in Sec. 2.4, Table 2.4.1, the reaction wheel action space is directly affected by them. Their impact on the commanded acceleration can be determined using the solution of the attitude dynamics equation 2.3.7. The Table 5.1.1 formalizes the constraints that the reaction wheel action has to satisfy.

#	Description	Expression
2	Maximum RW velocity	$\lvert\dot{\psi}_t\rvert \leq \frac{\psi_{max}-\psi_t}{t_{step}}$
3	Maximum RW acceleration	$\lvert\dot{\psi}_t\rvert \leq \dot{\psi}_{max}$
6	Maximum angular velocity	$\lvert\dot{\psi}_t\rvert \leq -\frac{I_{zz}}{D}\frac{\dot{\theta}_{N_{max}}-\dot{\theta}_{N_t}}{t_{step}}$
7	Maximum angular acceleration	$\lvert\dot{\psi}_t\rvert \leq -\frac{I_{zz}}{D}\ddot{\theta}_{N_{max}}$

Table 5.1.1: Attitude and reaction wheel constraints referred to the commanded acceleration

Where t_{step} is the length of the time step t. Through the constraints, it is now possible to restrict the reaction wheel action space. Indeed, taking into account the most strict constraints, the maximum and minimum accepted values of the commanded acceleration can be computed. Consequently, the upper and lower boundaries of the reaction wheel action space at time step t, $\left[\dot{\psi}_t^{INF}, \dot{\psi}_t^{SUP}\right]$ are determined.

The effective commanded acceleration, given the clipped output of the Actor network and the boundaries to respect, results to be:

$$\dot{\psi}_t = \tilde{\dot{\psi}}_t \frac{\dot{\psi}_t^{SUP} - \dot{\psi}_t^{INF}}{2} + \frac{\dot{\psi}_t^{SUP} + \dot{\psi}_t^{INF}}{2} \tag{5.1.5}$$

Thruster Impulse Space

The same approach followed in the case of the reaction wheel action space is applied to the thruster action space. Recalling the constraints on the velocity and on the thruster itself, #1, #4 and #5 in Table 2.4.1, it is necessary to formalize them such that the upper and lower boundaries of the

thruster action space can be determined.

As done previously, the dynamics equations can be used for this purpose. The matrix notation of the Clohessy-Wiltshire equations [26], reported eq. (2.3.4), is used to ease the explanation.

$$\begin{aligned} \mathbf{r}_{t+1} &= [\mathbf{\Phi}_{rr}(t_{step}]\mathbf{r}_t + [\mathbf{\Phi}_{rv}(t_{step})]\mathbf{v}_t \\ \mathbf{v}_{t+1} &= [\mathbf{\Phi}_{vr}(t_{step}]\mathbf{r}_t + [\mathbf{\Phi}_{vv}(t_{step})]\mathbf{v}_t \end{aligned} \tag{5.1.6}$$

From the constraints on the velocity, the limit velocity can be computed. In particular, at each time step t, the initial velocity $||\mathbf{v}_t + \mathbf{\Delta v}_t||$ and the velocity of the next state $||\mathbf{v}_{t+1}||$ must be inside the acceptable velocity ranges expressed in constraints #4 and #5. For what concerns the recoverable relative velocity limit (#4), as done in [23], a stricter constraint is enforced : $||\mathbf{v}|| \leq v_{max}$. Note that the constraint #4 could be eventually relaxed, being it applied to the components of the velocity vector.

The velocity limit applied to the initial velocity can be temporally computed as the minimum velocity between the limiting ones imposed in constraints #4 and #5

$$v_t^{LIM,temp} = min\left(v_{max}, v_{dock} + f_s\sqrt{\frac{F_{max}}{2m}||\mathbf{r}_t||}\right) \tag{5.1.7}$$

It can be noted that constraint #5 imposes a limit velocity that depends on the relative distance. As explained before, since the velocity limit must be applied also to the relative velocity of the next state, $\mathbf{v}_{t+1}$, it follows that a prediction of the relative distance reached at the next step, $\mathbf{r}_{t+1}$ is needed. Hence, it can be useful to determine the smallest relative distance that can be reached starting from the current position, assuming to have already changed the attitude through the reaction wheel.
To compute $\mathbf{r}_{t+1}$ through the planar CW equations, eq. (5.1.6), the knowledge of the relative velocity at time t is necessary. Since it cannot be computed from the evaluation of the velocity impulse given at time t, which is still unknown, it can be guessed as done in [23]. Considering now the guessed velocity as $v_t \in \{-v_t^{LIM,temp}; 0; v_t^{LIM,temp}\}$, the first equation of the matrix form of the planar CW equations can be solved to find the approximated smallest relative distance that can be reached, r_{t+1}^{MIN}. Therefore, the smallest bounded relative velocity limit at time step $t+1$ is quantified.

$$v_{t+1}^{MAX} = v_{dock} + f_s\sqrt{\frac{F_{max}}{2m}r_{t+1}^{MIN}} \tag{5.1.8}$$

The still missing velocity limit is the one that must be respected at time step t. Nevertheless, by using the CW equations and the knowledge of the position at time step t, $\mathbf{r}_t$, and of the velocity limit at next time step, $\mathbf{v}_{t+1}^{MAX}$, v_t^{MAX} can be determined.
Altogether, the final velocity limit at time step t is formalized as follows.

$$v_t^{LIM} = min(v_t^{LIM,temp}, v_t^{MAX}) \tag{5.1.9}$$

For what concerns constraint #1, it can be easily formalized considering the relation between the thrust and the velocity impulse, that is:

$$\Delta v_t = \frac{F_t}{m} t_{step} \tag{5.1.10}$$

Hence, the velocity impulse constraint is in the form:

$$\Delta v_t = [\Delta v_{tmin}, \Delta v_{tmax}] \tag{5.1.11}$$

At this point it is trivial to define the boundaries of the thruster impulse space at a given time step. In particular:

$$\begin{aligned} \Delta v_t^{SUP} &= min(\Delta v_{tmax}, v_t^{LIM} - v_t) \\ \Delta v_t^{INF} &= max(\Delta v_{tmin}, -v_t^{LIM} - v_t) \end{aligned} \tag{5.1.12}$$

Finally, the effective velocity impulse, given the clipped output of the Actor network and the boundaries to respect, results to be:

$$\Delta v_t = \widetilde{\Delta v}_t \frac{\Delta v_t^{SUP} - \Delta v_t^{INF}}{2} + \frac{\Delta v_t^{SUP} + \Delta v_t^{INF}}{2} \tag{5.1.13}$$

As noted by [23], the constraint on the actuator limit and the constraints on the Chaser velocity are formalized separately. It may happen that respecting one of the two types of constraints results in the non-compliance of the other one. This happens when the impulse constrained by the upper velocity limit v_t^{LIM} is lower than the action space lower bound Δv_t^{INF} or the opposite case. Physically, this implies that the needed velocity impulse to remain inside the acceptable range of velocity and so fulfill the constraints overcomes the physical capacity of the thrusters. In this circumstance, the thruster velocity impulse is set to its physical limitation sacrificing the compliance of the relative velocity constraint.

5.1.3 Discretization of the dynamics

The dynamics problem introduced in Sec. 2.3 is illustrated in continuous form. In this section, the dynamics is discretized in a recursively form, which means that the state in any time step is computed from the state at previous step. For this reason, as it was introduced in Sec. 3.4.3, the Agent must take into account not only the direct consequence of the action it selects but also its

long-term effects. Without going into much detail, it can be said that executing an action without considering the future effects of it could eventually lead the Agent to completely disruptive states, that would result in failing the entire mission.

To discretize the dynamics, let's assume that the state S_t at time step t is known, thus it is possible to compute the successive state S_{t+1}. In addition, the initial condition of the state S_0 is selected manually from the user. Starting from the simpler attitude dynamics equations of the Chaser, the recursive and discretized form results in:

$$\begin{aligned} \dot{\theta}_{N_{t+1}} &= -\tfrac{D}{I_{zz}}\dot{\psi}_t t_{step} + \dot{\theta}_{N_t} \\ \theta_{N_{t+1}} &= \dot{\theta}_{N_{t+1}} t_{step} + \theta_{N_t} \end{aligned} \tag{5.1.14}$$

Where $\dot{\psi}_t$ is the instantaneous acceleration of the reaction wheel.
The instantaneous velocity impulse of the thruster Δv_t can be translated into the velocity at time step t as follows.

$$\begin{aligned} v_{x_t} &= \dot{x}_t + sin(\theta_{N_{t+1}})\Delta v_t \\ v_{y_t} &= \dot{y}_t - cos(\theta_{N_{t+1}})\Delta v_t \end{aligned} \tag{5.1.15}$$

Lastly, the discretized translation dynamics equations, in matrix notation, are formalized through the analytical solution of the CW equations eq. (2.3.5).

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \dot{x}_{t+1} \\ \dot{y}_{t+1} \end{bmatrix} = \begin{bmatrix} 4 - 3\cos nt_{step} & 0 & \frac{\sin nt_{step}}{n} & \frac{2}{n}(1 - \cos nt_{step}) \\ 6(\sin nt_{step} - nt_{step}) & 1 & \frac{2}{n}(\cos nt_{step} - 1) & \frac{1}{n}(4\sin nt_{step} - 3nt_{step}) \\ 3n\sin nt_{step} & 0 & \cos nt_{step} & 2\sin nt_{step} \\ 6n(\cos nt_{step} - 1) & 0 & -2\sin nt_{step} & 4\cos nt_{step} - 3 \end{bmatrix} \begin{bmatrix} x_t \\ y_t \\ v_{x_t} \\ v_{y_t} \end{bmatrix} \tag{5.1.16}$$

The discrete dynamics equations and the constraints show a dependence from the time step length t_{step}. Specifically, decreasing or increasing the size of the time step affects the frequency of control actions, leading to a less or more precise control. Simultaneously, this also affects the time needed to reach the Target and so the number of actions to be taken. In this work, as it is done in [23], the time step length is reduced as the Chaser gets closer to the Target, implying that the control gets more precise as the gap between the two spacecraft decreases. The logic with which the length of the time step is reduced can be found in Table 5.1.2.

Condition	$\mathbf{t_{step}}$ **size**
$r_t > 100\ m$	$10\ s$
$100\ m \geq r_t > 10\ m$	$5\ s$
$10\ m \geq r_t > 5\ m$	$3\ s$
$5\ m \geq r_t$	$1\ s$

Table 5.1.2: Time step size reduction logic as a function of the relative distance

5.2 Overview of the reward logic

To properly teach the Agent which state should be preferred and which one should be avoided, a reward logic must be defined. In Reinforcement learning, and so in Meta-Reinforcement learning, the reward function allows the Agent to learn "good" actions and reach "good" states. To better understand what is the scope of the reward assignment, imagine to train a dog. The dog learns to execute commands as the trainer *rewards* it with a treat. This very simple and everyday life example can be compared to what happens in a generic reinforcement learning algorithm.
However, the selection of a reward function that suggests the Agent how to behave is not a trivial matter. Choosing, for example, an ambiguous or faulty reward logic may lead to completely wrong or too poor solutions of the problem.

A common technique is to define the so-called *shaping reward* function, that typically is a potential function over the state that provides guidance to the Agent, improving so the learning [49].
A different type of rewards are the *sparse rewards*, that consist in positive or negative bonuses, assigned as integer numbers when some conditions are met. However, sparse rewards do not give hints to the Agent and do not improve the learning process if used as the main reward assignment function. This can be explained by the fact that the Agent explores randomly, executing random actions until it gets non null or, eventually, positive rewards. A problem may arise: the sequence of actions that were rewarded positively may be very long and some of those actions could be useless in the assignment of the reward. In other words, not all the actions, if a proper shaping reward function is not introduced, are useful in getting positive rewards. For example, consider the case where a bonus is assigned as a sparse reward when a certain distance from the Target is reached. Assume that the reward logic only relies on the sparse rewards. The Agent may end up going back and forth around the "checkpoint" distance, accumulating positive rewards. As it can be imagined, this behavior is not acceptable. This issue is known as credit assignment problem [50].

In this work, the reward function is taken as the sum of two contributions: the *shaping rewards* R_t^{shape} and the *sparse rewards* R_t^{sparse}.

$$R_t = R_t^{shape} + R_t^{sparse} \tag{5.2.1}$$

The way the two types of rewards are defined for the ARPOD problem is discussed in chapter 6 for each case analysed.

5.3 Agent's Hyperparameters

Reinforcement learning based algorithms can be applied to any sort of environment that respects the Markov property, being them model-free algorithms. Indeed, as opposed to the environment that is peculiar and unique for any problem that is studied, the Agent is a general entity that can be thrown in many different environments with minor changes. It is still important to define the

parameters, usually referred to as hyperparameters to distinguish them from the neural networks' parameters, that build up the Agent and its learning process.

In this section, the hyperparameters that have been introduced throughout chapter 3, are recalled and explained in much detail. The PPO algorithm, discussed in Sec. 3.6, consists in two main steps. First, after the environment generation, a batch of trajectories or, equivalently roll-outs, is stored and then the policy is updated. Each trajectory, that contains sequence of states, rewards and actions, is associated with a single episode. During training, in particular in the forward pass, the policy and value function recurrent layers are unrolled for multiple time steps. After each update, the clipping parameter, used in the PPO algorithm, is adjusted in order to target a KL divergence between the old and new policies. In this work, as done in [20], the degree of exploration is automatically adjusted during the learning process. This is possible because the log probabilities, needed for the evaluation of the Actor objective function are computed using the exploration variance, output of the Actor neural network.

The hyperparameters associated with the PPO algorithm and the experience collection are the following ones.

- **Horizon**: it is the number of trajectories that are stored before the update of the policy. A large horizon grants more stable updates.
- **Epoch**: it corresponds to the number of times that the learning algorithm will work through the entire training dataset. Decreasing the number of epochs, grants more stable updates but slows down the learning.
- **Recurrent steps**: the number of time steps for which the recurrent layers of the neural networks are unrolled. Unrolling the recurrent layers for more time steps results in a policy that can capture longer temporal dependencies.
- γ: the discounting factor that controls the greediness of the Agent. With a small value of γ, immediate rewards are preferred over long term ones.
- ϵ: the clipping parameter, that sets the threshold of divergence between the old and new policies during update.
- **KL target**: a target KL divergence. It is used to dynamically adjust the learning rates of the neural networks and the clipping parameter ϵ.

The last set of hyperparameters regards the two artificial neural networks of the Agent.

- **Number of layers**: it is the number of hidden layers in between the input and output layers. Typically, more layers are needed when facing complex problems.
- **hidden neurons**: it is the number of neurons in the hidden layers.

- **learning rate**: it influences the size of the updates in the policy and value function. The higher is the learning rate, the faster but more unstable the learning process becomes.

All the hyperparameters must be selected manually. Typically, even if some guidelines are present in literature or in existing projects, the best way to properly select these parameters is through a trial and error procedure.

Chapter 6

Experiment setup and results

This chapter focuses on clarifying how the experiments are set up and on the analysis of the results that are obtained. In particular, the implementation in the Meta-Reinforcement learning framework of the ARPOD problem, already introduced in chapter 5, is detailed and specified for every experiment that has been conducted. For any case that is considered, the reward logic, the hyperparameters and the initial and terminal conditions are defined.

It is worth to recap the problem that this work focuses on. A Meta-Reinforcement learning algorithm has to demonstrate its adaptability and robustness in the computation of different trajectories, assuming uncertainties on the dynamics and on the model of the Chaser spacecraft. These trajectories must allow the Chaser to approach a Target spacecraft and dock with it always fulfilling the constraints that have been formalized in Sec. 2.4. As a rule of thumb, a rendezvous trajectory can be divided into three main phases. An initial rendezvous phase from 10 km to 1 km, where the Chaser is expected to get closer to the Target. Then, the proximity operation phase, between 1 km and 100 m, in which the Chaser must approach the Target respecting the docking cone constraint for a safe approach. And finally, the docking phase, where the Chaser has to slow down enough to respect the velocity constraints and allow the docking with the Target.

In this work, two cases have been considered. The first experiment focuses on demonstrating the ability of the Meta-RL Agent to adapt to large uncertainties in the dynamics, ensuring the accomplishment of the rendezvous starting from 1 km of relative distance. The second and last case considers an initial relative distance of 5 km, thus proving the ability and, eventually, the limitations of the implemented Meta-RL Agent to resolve more complex rendezvous missions.

The simulations are executed using Python and Pytorch on a ASUS ROG GL552VW personal computer, with an Intel Core i7-6700 HQ CPU, a NVIDIA GeForce GTX 960M and 16 GB of RAM. For the sake of clarity, the experiments did not use GPU acceleration but relied entirely on the CPU. For this reason, simulations took at least 6 hours of calculations.

6.1 Experiment 1: Proximity Operations and Docking

The first experiment regards the accomplishment of Proximity Operations and Docking phases. The Chaser, subjected to strict constraints for a safe docking, starts the operation phase at 1 km oriented towards the Target. The initial configuration taken into account is the so-called V-bar configuration, according to which the Chaser initial position is along the tangential direction of the Hill's frame Fig. 6.1.1. In addition, the relative translation and angular velocity are null.

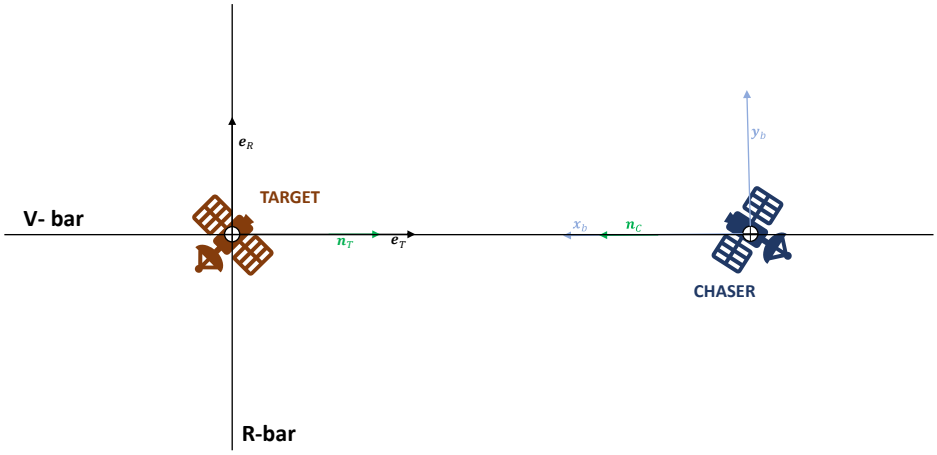


Figure 6.1.1: *Not to scale V-bar configuration. The Chaser approaches the Target along the tangential direction of the Hill's frame.*

$$S_0 = [0, 1000, 0, 0, 0, 0]$$

However, uncertainties are considered on the initial state of the Chaser to demonstrate the strength of a Meta-RL Agent to solve different tasks. The initial state is generated for each episode, considering a normal distribution of values assumed by each of the state variables, excluding the angular velocity. At each episode, from the distribution, a random value is picked and used to generate the initial conditions, or, equivalently, the environment. The maximum and minimum values that can be assumed by each of the uncertain state variables are reported in Table 6.1.1.

State variables	Uncertainty range	Min value	Max value
x_0	$[-50; +50]$ m	-50 m	50 m
y_0	$[-50; +50]$ m	-950 m	1050 m
v_{x_0}	$[-10^{-5}; 10^{-5}]$ $m\ s^{-1}$	-10^{-5} $m\ s^{-1}$	10^{-5} $m\ s^{-1}$
v_{y_0}	$[-10^{-5}; 10^{-5}]$ $m\ s^{-1}$	-10^{-5} $m\ s^{-1}$	10^{-5} $m\ s^{-1}$
θ_0	$[-5; +5]$ deg	-5 deg	$+5$ deg

Table 6.1.1: Maximum and minimum values assumed by the uncertain state variables: relative position, translation velocity and orientation.

6.1.1 Reward logic

The reward logic introduced in Sec. 5.2 is now specified in order to be applied in this first experiment. As mentioned before, two kinds of reward assignment techniques are used: shaping rewards and sparse rewards, that will be treated separately.

Shaping rewards

The shaping rewards are used to provide hints to the Agent on the correct behavior to follow in order to reach the Target. Since both the attitude and the relative distance must be controlled, two contributions are identified for the shaping reward function: a shaping reward associated with the distance and a shaping reward related to the attitude. For the first case, in this work, an Artificial Potential Field [16] is placed in correspondence of the Target position, that coincides with the origin of the Hill's frame. In particular, the value of the potential, that depends on the square of the relative distance from the fictitious attractor, is used inside the reward function.

The potential field is written as:

$$U(\mathbf{r}_t) = \frac{1}{2} k_{att} ||\mathbf{r}_t||^2 \tag{6.1.1}$$

Where k_{att} is the attraction coefficient and is set to 10. The artificial potential, as it can be seen in eq. (6.1.1), increases as the square of the module of the relative distance. It means that, as the Chaser get closer to the Target, this value tends to become null. For this reason, the shaping reward based on this potential field is written in the following form, expressed in eq. (6.1.2). The Fig. 6.1.2 illustrates the behavior of the shaping reward function as a function of the relative distance between the Chaser and the Target.

$$R_t^{shape,1} = -\alpha_1(\tilde{U}_t - 1) \tag{6.1.2}$$

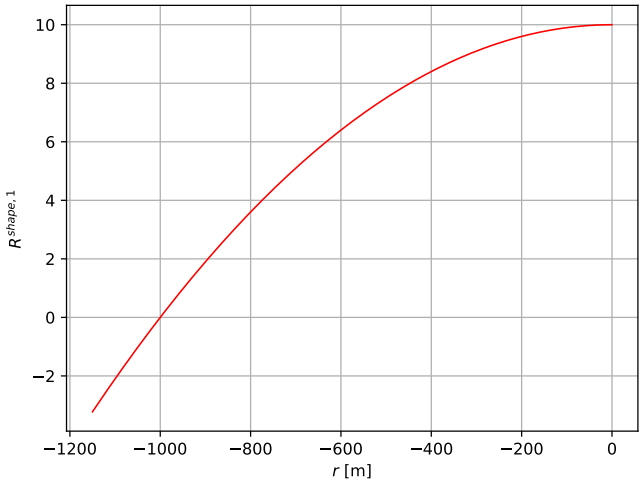


Figure 6.1.2: *Shaping reward as a function of the relative distance. The minus sign on the relative distance is useful for a better understanding of the curve.*

In eq. (6.1.2), α_1 is a numerical coefficient, fixed at a value of 10. The term $\tilde{U}_t$ is the normalized artificial potential. Indeed, due to the high values that the potential could assume, since it is calculated as the square of the relative distance, a normalization with respect to the potential at the nominal initial relative distance is performed.

$$\begin{aligned} \tilde{U}_t &= \frac{U_t}{U_{1000}} \\ U_{1000} &= \frac{1}{2} k_{att} (1000)^2 \end{aligned} \tag{6.1.3}$$

It can be noted that the function expressed in eq. (6.1.2) and illustrated in Fig. 6.1.2 increases up to a value of 10 when the distance is reduced to 0. It only becomes negative as the Chaser moves away from its initial position. For this reason, a penalty must be added to prevent the Chaser to get closer to the Target and move away from it without reaching distances greater than 1000 m.

The other contribution to the definition of the shaping reward comes from the need of controlling the attitude as the Chaser approaches the Target. In this work, the same function adopted in [23] is in part considered. The shaping reward associated with the attitude control then takes the following form.

$$R_t^{shape,2} = \alpha_2 \frac{5 - |\theta_{N_t}|}{180} \tag{6.1.4}$$

Where α_2 is a numerical coefficient that changes with respect to the distance according to the logic expressed in 6.1.5, and θ_{N_t} is the attitude angle wrapped between -180 and 180 degrees.

$$\begin{aligned} \alpha_2 &= 0 \quad if \quad ||\mathbf{r}_t|| > 100m \\ \alpha_2 &= 20 \quad if \quad ||\mathbf{r}_t|| \leq 100m \\ \alpha_2 &= 50 \quad if \quad ||\mathbf{r}_t|| \leq 10m \end{aligned} \tag{6.1.5}$$

The attitude shaping reward is positive as the attitude angle is lower than 5 *deg*. Moreover, the penalization on the attitude become more and more important as the distance is reduced, due to the logic that is described above. It can be also noted that, the maximum positive value for the above reward function is less than 10, which is instead the maximum positive score that can be provided from the position reward function. This condition must be always verified in order to avoid behaviors of the Agent where it prioritizes the attitude control over the approach. Altogether, the shaping reward is given by:

$$R_t^{shape} = R_t^{shape,1} + R_t^{shape,2} \tag{6.1.6}$$

Sparse rewards

Sparse rewards are instead very useful to provide bonus or penalties for good or bad behaviors, respectively. Typically, the assignment of positive rewards (bonuses) can increase the stability of the learning process while the penalties guarantee that bad behaviors are avoided.

In this work, bonuses are given when the Agent reaches some milestones or respects some constraints. On the contrary, penalties, that are negative rewards, suggest the Agent to avoid the exploration of poor parts of the state space.

All bonuses and penalties are listed below, along with the explanation of the conditions that must be met in order to receive them.

- If the Chaser reaches 500 m, it receives a bonus. The bonus is nullified if the Chaser tends to move away and return to 500 m.
- If the Chaser reaches 100 m, the Agent receives a bonus. The bonus is nullified if the Chaser returns to further distances and goes back in the range of the docking phase.
- If the Chaser reaches 50 m, the Agent receives a bonus. The bonus is nullified if the same condition as the previous cases is met.
- A bonus is assigned if the Chaser is 5 m away from the Target. However the bonus is nullified if the Chaser tends to go back and return inside the docking range.
- A bonus is assigned when the Chaser is docked. This condition terminates the episode with success.

$$r_t \leq 1\,m \tag{6.1.7}$$

- A cumulative bonus is assigned when the Chaser is docked with an angle lower than the maximum acceptable one.

$$r_t \leq 1\,m \; and \; |\theta_{N_t}| \leq 5\,deg \tag{6.1.8}$$

- A cumulative penalty is assigned whenever the Chaser starts to move back.

$$r_t > r_{t-1} \tag{6.1.9}$$

- When the Chaser is in the proximity operation phase ($r_t \leq 1000\ m$) and explores a region outside the docking cone (constraint #8), it receives a penalty. However, if it keeps making progress inside the docking cone it receives a cumulative bonus. The condition is formalized as follow:

$$(x_t - y_t > 0 \; or \; x_t + y_t < 0) \tag{6.1.10}$$

  A supplementary penalty is given if the Chaser misses the Target.

$$y_t < -1\,m \tag{6.1.11}$$

- If the Chaser does not respect constraints #4 and #5, it receives a penalty. The constraints mentioned here are explained in Sec. 5.1.2.

- If the Chaser explores a region farther than the initial relative distance, it receives a penalty. A margin of $150\,m$ is considered because of the uncertainties on the initial position. This condition forcefully terminates the episode.

In Table 6.1.2, bonuses and penalties are reported and quantified, along with the assignment conditions.

Condition	**Bonus/Penalty**	**Nullification**
$r_t \leq 500\,m$	$R_t^{sparse} = 10$	$R_t^{sparse} = 0$ if $r_{t-1} > 500\ m$
$r_t \leq 100\,m$	$R_t^{sparse} = 15$	$R_t^{sparse} = 0$ if $r_{t-1} > 100\ m$
$r_t \leq 50\,m$	$R_t^{sparse} = 15$	$R_t^{sparse} = 0$ if $r_{t-1} > 50\ m$
$r_t \leq 5\,m$	$R_t^{sparse} = 20$	$R_t^{sparse} = 0$ if $r_{t-1} > 5\ m$
$r_t \leq 1\,m$	$R_t^{sparse} = 150$	None
$r_t \leq 1\,m$ and $\|\theta_{N_t} \leq 5\,deg$	$R_t^{sparse} = R_t^{sparse} + 50$	None
$r_t > r_{t-1}$	$R_t^{sparse} = R_t^{sparse} - 5$	None
$r_t \leq 1000\,m$ and $(x_t - y_t > 0 \; or \; x_t + y_t < 0)$	$R_t^{sparse} = -10$	None
$r_t \leq 1000\,m$ and $(x_t - y_t < 0 \; or \; x_t + y_t > 0)$	$R_t^{sparse} = R_t^{sparse} + 5$	None
$r_t \leq 1000\,m$ and $y_t < -1\,m$	$R_t^{sparse} = -10$	None
$v_t > v_{max}$ or $v_t > v_{dock} + f_s\sqrt{\frac{F_{max}}{2m} r_t}$	$R_t^{sparse} = -20$	None
$r_t > r_0 + 150\,m$	$R_t^{sparse} = -500$	None

Table 6.1.2: Summary of the sparse reward logic for the assignment of bonuses and penalties

6.1.2 Hyperparameters and Neural Networks setup

The hyperparameters introduced in Sec. 5.3 are now quantified after a process of trial and error. The final set of hyperparameters is reported in Table 6.1.3.

Hyperparameter	Value
Horizon	20
Epoch	16
Recurrent steps	120
γ	0.95
ϵ_0	0.1
KL target	0.0001

Table 6.1.3: Hyperparameters for the first experiment

For the sake of clarity, the clipping parameter ϵ is initialized as $\epsilon_0 = 0.1$. As explained in Sec. 3.5.3, it is changed during learning to target a specified value of KL divergence.

The architecture of the two neural networks, the Actor and the Critic, is inspired by [20]. In particular, a *LSTM cell* is placed in the second hidden layer of the networks, that share the same structure. The other two hidden layers are fully connected and linear. Finally, also the output layer is a linear layer. The number of neurons and the activation functions are reported in Table 6.1.4 and Table 6.1.5.

Layer	Neurons	Activation
Hidden 1	130	tanh
Hidden 2	90	tanh
Hidden 3	60	tanh
Output	2	Linear

Table 6.1.4: Actor Neural Network architecture

Layer	Neurons	Activation
Hidden 1	130	tanh
Hidden 2	90	tanh
Hidden 3	60	tanh
Output	1	Linear

Table 6.1.5: Critic Neural Network architecture

6.1.3 Results

In Meta-RL, it is possible to distinguish two different phases: a training phase and a testing phase. The former consists in the optimization of the policy. This process gives as result an adaptive Agent that is able to accomplish any task similar to the ones it is trained on. During the testing phase, instead, the optimized Agent does not learn. By fixing the policy that is obtained in the previous phase, the Agent is thrown in a precise environment, similar to the ones in which it is trained. Then, by applying the fixed policy, which becomes a deterministic one, it is tested to solve this new task.

Training

The training of the recurrent networks is performed over 30000 episodes, where for each episode a new environment is generated. To reduce computational time, every episode terminates at 100 iterations over the environment dynamics, taking into account the possibility of early termination. In this phase, uncertainties are added also to the inertia properties of the Chaser spacecraft Table 6.1.6. This can be translated in the possibility of applying the algorithm to different spacecrafts for the testing phase.

Variable	Description	Min value	Max value
m	Spacecraft mass	11 kg	13 kg
I_{zz}	Spacecraft mass moment of inertia in z-axis	5×10^{-2} $kg\ m^2$	6.5×10^{-2} $kg\ m^2$

Table 6.1.6: Uncertainties on the inertia properties of the Chaser

Bearing in mind that the ultimate goal of the training is to optimize the adaptive policy and collect the maximum possible reward, the results are shown below.

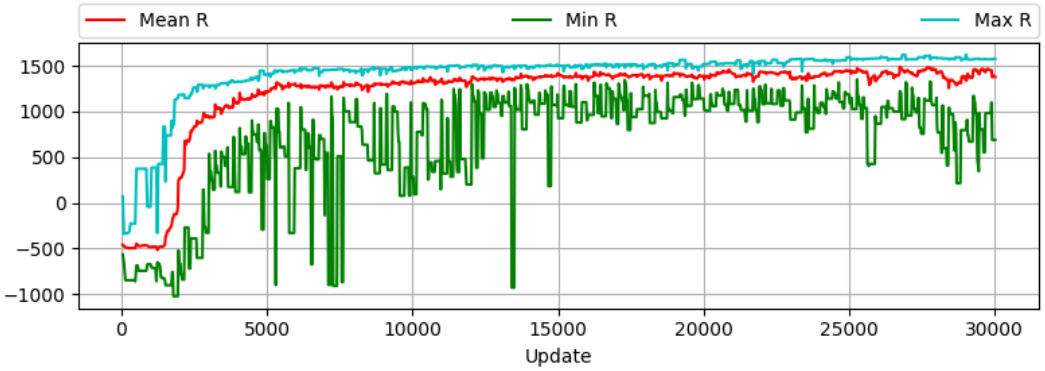


Figure 6.1.3: *Optimization rewards learning curve for the first experiment.*

In Fig. 6.1.3, the mean reward over the horizon is shown in red. The maximum reward that is obtained during each training episode is shown in cyan and it is referred to the best possible

action sequence that grants the highest reward. On the other hand, the green curve illustrates the minimum reward, associated with bad samples. Altogether, even considering some disruptive samples, as for example the one near 15000 episode, it is clear that the learning is successful.
One may note that the behavior at the end of the training becomes unstable. The reason behind this phenomenon could be linked to the correction of the learning rates and clipping parameter, due to the *servo-kl* control, discussed in Sec. 3.5.3. In particular, if the KL divergence decreases becoming too lower than the target one, the Agent may change his behavior, visiting even poor regions of the action-state space and thus collecting lower rewards.

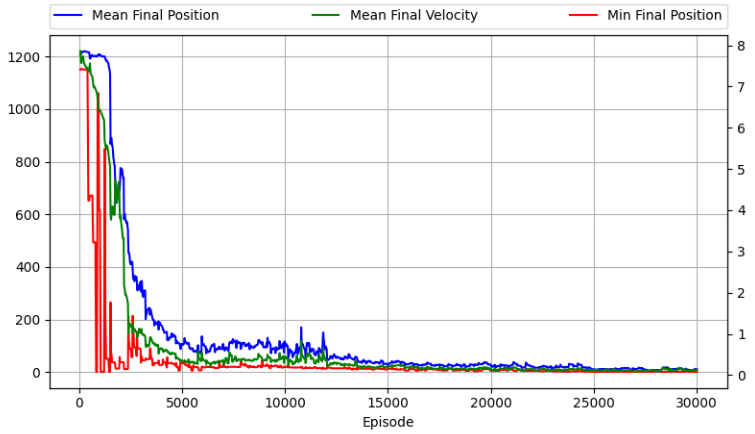


Figure 6.1.4: *Optimization curves of the terminal positions reached by the Chaser, with the associated terminal velocity.* First case.

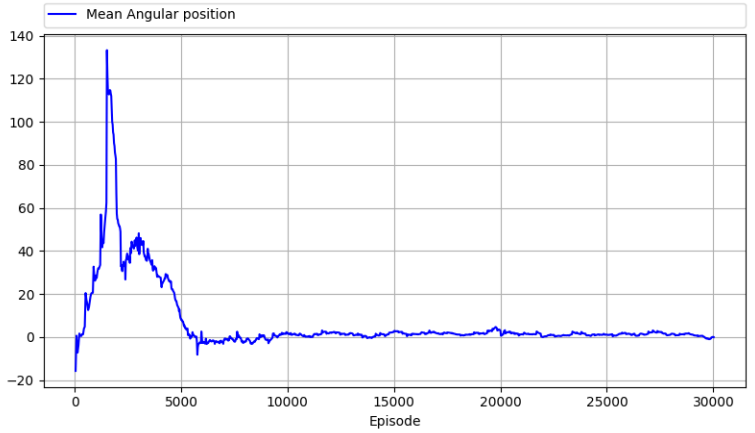


Figure 6.1.5: *Optimization curve of the terminal attitude of the Chaser.* First case.

The above figures, Fig. 6.1.4 and Fig. 6.1.5, show the learning process of the Agent in detail. It can be noted that the mean terminal position (in red) and velocity (in green) over the course of the training phase are optimized. The minimum final position reached by the target is shown in

red and it is associated with the maximum reward that is collected during training, shown in cyan in Fig. 6.1.3. In addition, the Agent is able to adjust its mean final attitude as shown in Fig. 6.1.5. It is worth recalling that these learning curves and optimization results are referred to a vast group of similar environment. In other words, the adaptability of the Meta-RL Agent is verified.

Testing

Once the policy is optimized, the Agent is tested with a fixed policy. The uncertainties on the inertia properties are removed, in order to solve the problem proposed by [22] and described in chapter 2. The Agent is tested over 10000 episodes, each with different initial conditions due to uncertainties. The terminal points, that are shown in Fig. 6.1.6, demonstrate that among the 10000 test trajectories, the 16.58% are acceptable ones. The trajectories that reach less than 10 meters of separation from the Target are shown in Fig. 6.1.7.

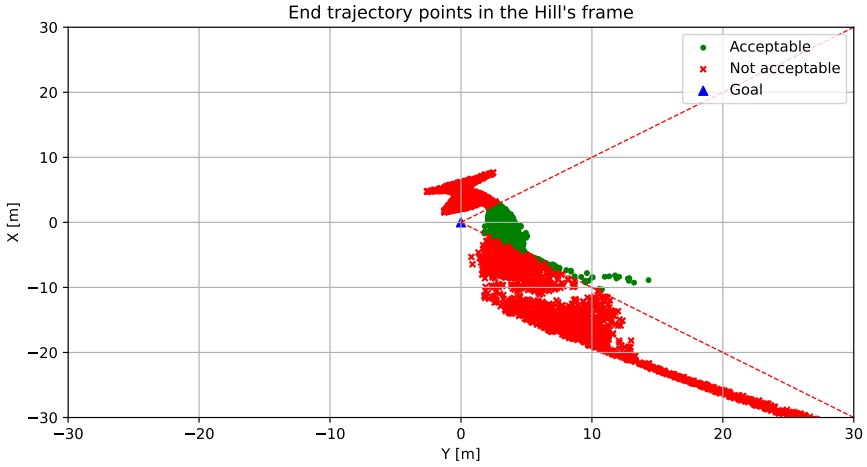


Figure 6.1.6: *Terminal points of the 10000 test trajectories. The green dots represent the acceptable trajectories and the red cross the not-acceptable ones. The docking cone is displayed in red.*

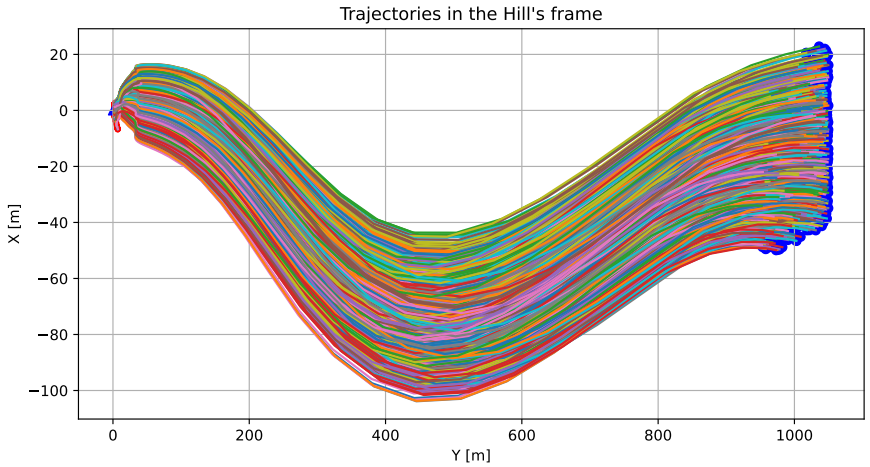


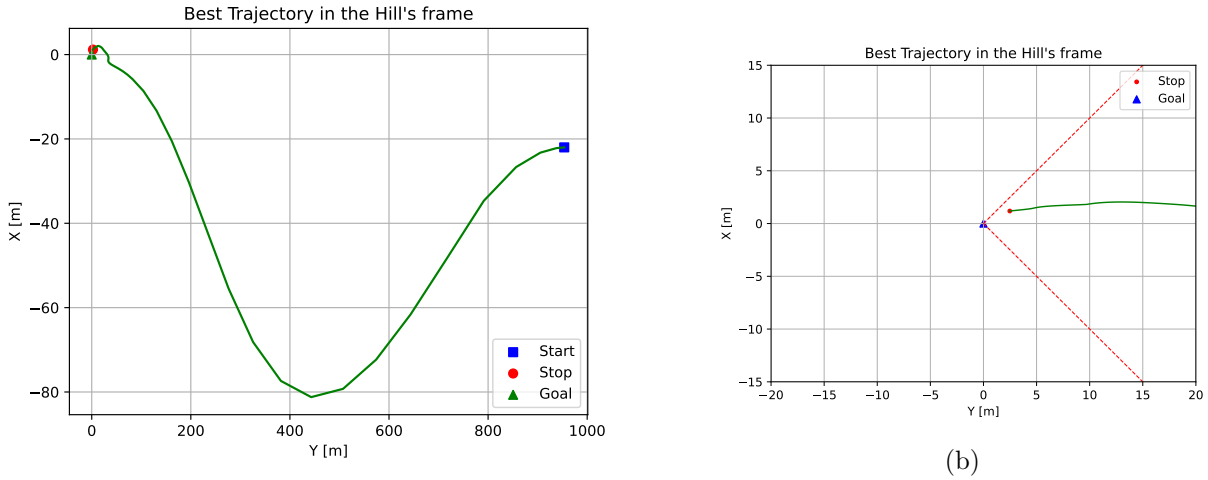
Figure 6.1.7: *Trajectories that end at less than* 10 m *of relative distance from the Target.*

From the Fig. 6.1.7, it can be noted how vast is the area of the starting points. This means that the Agent succeeds in learning the correct behavior from a multitude of initial positions.

For a clear view of the ability of the Agent to respect the constraints imposed in Sec. 2.4, the best trajectory is extracted from the 1642 ones, considering the terminal position reached as a criterion for the selection.

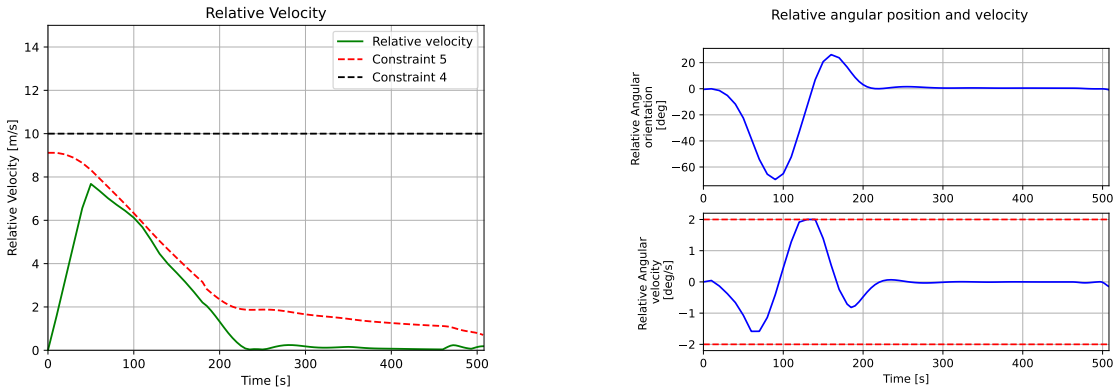
The best selected trajectory is showed in Fig. 6.1.8, with terminal position and final attitude angle equal to:

$$||\mathbf{r}|| = 2.74\,m \quad |\theta_N| = 0.80\,deg$$



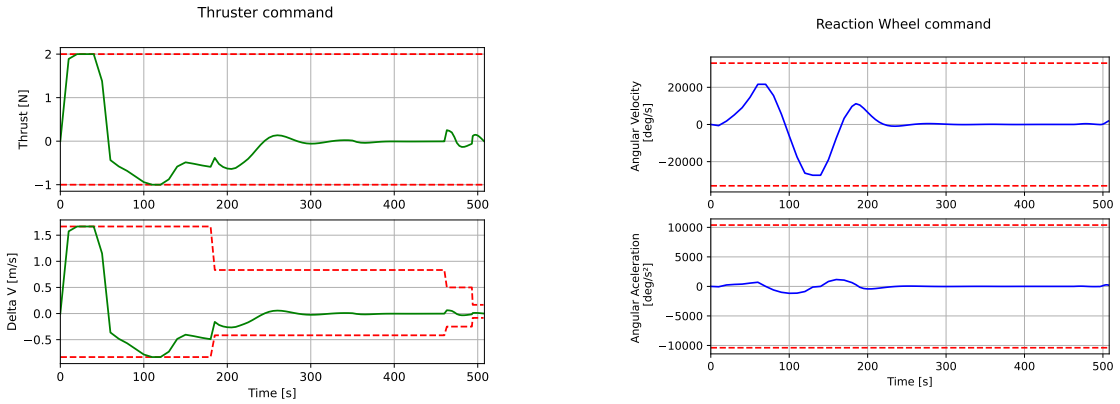
(a) (b)

Figure 6.1.8: *Best trajectory in the Hill's reference frame* (a). *Zommed in* (b), *where the docking cone is displayed.*



(a) (b)

Figure 6.1.9: *Relative velocity* (a) *is always respectful of both constraints. In the proximity operation phase, the constraint on the bounded relative velocity (#5) is dominating the one on the recoverable relative velocity (#4). In* (b) *, the attitude angle and the relative angular velocity are illustrated.*



(a) (b)

Figure 6.1.10: *The commanded thrust and, consequently, the* ΔV*, are shown in* (a)*. The reaction wheel commands, scaled by a factor of 10, are shown in* (b)*.*

As shown in Fig. 6.1.9 and Fig. 6.1.10, the Agent is always respecting the constraint imposed. It can also be noted how smooth is the control obtained as output from the Actor neural network. In other words, the Agent needs only a few manoeuvres to reach the target and adjust its attitude, fulfilling the constraints on the velocity, the attitude and the physical limitations of the actuators. For the sake of clarity, in Fig. 6.1.10 (a), the dashed red lines represent the constraints due to the physical limitations of the thruster. It can be noted that the impulse velocity limits decrease in time, as the time step is reduced. This is justified from eq. (5.1.10) with a fixed maximum thrust.

6.2 Experiment 2: Rendezvous from 5 km

The second experiment focuses on a Rendezvous Proximity Operations and Docking manoeuvre from $5\,km$ of nominal initial distance along the V-bar. Also in this case, uncertainties are added to the initial state. However, due to the complexity of this case, uncertainties are considerably reduced.
Assume a nominal initial state as:

$$S_0 = [0, 5000, 0, 0, 0, 0]$$

Uncertainties are added to the relative position and velocity, and to the attitude angle as reported incTable 6.2.1. As before, the relative angular velocity is not considered as an uncertain state component only to avoid an increase in the complexity of the problem.

State variables	Uncertainty range	Min value	Max value
x_0	$[-10; +10]$ m	-10 m	10 m
y_0	$[-10; +10]$ m	4990 m	5010 m
v_{x_0}	$[-10^{-7}; 10^{-7}]$ $m\ s^{-1}$	-10^{-7} $m\ s^{-1}$	10^{-7} $m\ s^{-1}$
v_{y_0}	$[-10^{-7}; 10^{-7}]$ $m\ s^{-1}$	-10^{-7} $m\ s^{-1}$	10^{-7} $m\ s^{-1}$
θ_0	$[-5; +5]$ deg	-5 deg	$+5$ deg

Table 6.2.1: Maximum and minimum values assumed by the uncertain state variables: relative position, translation velocity and orientation. Referred to experiment 2.

6.2.1 Reward logic

The reward logic implemented for the second experiment is similar to the one adopted in the case discussed previously. Indeed, the two reward functions differ only for minor changes.

Shaping rewards

The shaping rewards are identical in expression with respect to the Proximity Operations case. The only difference can be found on the normalization of the artificial potential. In particular:

$$\begin{aligned} \tilde{U}_t &= \frac{U_t}{U_{5000}} \\ U_{5000} &= \frac{1}{2} k_{att} (5000)^2 \end{aligned} \tag{6.2.1}$$

As is can be seen in eq. (6.2.1), the reference potential is evaluated at the nominal initial relative distance of 5000 km. The considerations made for the first case still hold.
The expression of the shaping reward referred to the attitude angle has not been modified.

Sparse rewards

With respect to the first case, new conditions are added for the assignment of sparse rewards. The newly introduced bonuses and penalties are listed below.

- If the Chaser reaches $1000\,m$, it receives a bonus. However, the bonus is nullified whenever the Chaser moves away and returns to $1000\,m$.
- A cumulative penalty is assigned when the Chaser starts to move back from relative distances higher than 1000 m.

For the sake of clarity, the Table 6.2.2 sums up all the bonuses and penalties considered in this experiment.

Condition	Bonus/Penalty	Nullification
$r_t \leq 1000\,m$	$R_t^{sparse} = 5$	$R_t^{sparse} = 0$ if $r_{t-1} > 1000\ m$
$r_t \leq 500\,m$	$R_t^{sparse} = 10$	$R_t^{sparse} = 0$ if $r_{t-1} > 500\ m$
$r_t \leq 100\,m$	$R_t^{sparse} = 15$	$R_t^{sparse} = 0$ if $r_{t-1} > 100\ m$
$r_t \leq 50\,m$	$R_t^{sparse} = 15$	$R_t^{sparse} = 0$ if $r_{t-1} > 50\ m$
$r_t \leq 5\,m$	$R_t^{sparse} = 20$	$R_t^{sparse} = 0$ if $r_{t-1} > 5\ m$
$r_t \leq 1\,m$	$R_t^{sparse} = 150$	None
$r_t \leq 1\,m$ and $\lvert\theta_{N_t} \leq 5\,deg$	$R_t^{sparse} = R_t^{sparse} + 50$	None
$r_t > 1000\,m$ and $r_t > r_{t-1}$	$R_t^{sparse} = R_t^{sparse} - 20$	None
$r_t \leq 1000\,m$ and $r_t > r_{t-1}$	$R_t^{sparse} = R_t^{sparse} - 5$	None
$r_t \leq 1000\,m$ and $(x_t - y_t > 0\ or\ x_t + y_t < 0)$	$R_t^{sparse} = -10$	None
$r_t \leq 1000\,m$ and $(x_t - y_t < 0\ or\ x_t + y_t > 0)$	$R_t^{sparse} = R_t^{sparse} + 5$	None
$r_t \leq 1000\,m$ and $y_t < -1\,m$	$R_t^{sparse} = -10$	None
$v_t > v_{max}$ or $v_t > v_{dock} + f_s\sqrt{\frac{F_{max}}{2m} r_t}$	$R_t^{sparse} = -20$	None
$r_t > r_0 + 150\,m$	$R_t^{sparse} = -500$	None

Table 6.2.2: Summary of the sparse reward logic for the assignment of bonuses and penalties in the second experiment.

6.2.2 Hyperparameters and Neural Networks setup

The hyperparameters selected for this experiment are reported in Table 6.2.3.

Hyperparameter	Value
Horizon	30
Epoch	16
Recurrent steps	120
γ	0.95
ϵ_0	0.1
KL target	0.0001

Table 6.2.3: Hyperparameters for the second experiment

As it can be noted, the *horizon* is the only parameter that has been changed for this case, in order to increase the stability of the training process.

The architecture of the Actor and Critic neural networks is identical to the one reported in Sec. 6.1.2. For the sake of clarity, the Table 6.1.4 and Table 6.1.5 are reported here below.

Layer	Neurons	Activation
Hidden 1	130	tanh
Hidden 2	90	tanh
Hidden 3	60	tanh
Output	2	Linear

Table 6.2.4: Actor Neural Network architecture

Layer	Neurons	Activation
Hidden 1	130	tanh
Hidden 2	90	tanh
Hidden 3	60	tanh
Output	1	Linear

Table 6.2.5: Critic Neural Network architecture

6.2.3 Results

The last case shows the complexity of the application of a Meta-RL algorithm to a full ARPOD manoeuvre. Once more, the Agent is trained over a variety of environments, each generated from the uncertainties in the dynamics. Following the training phase, the Agent is tested on a certain number of episodes.

Training

The training of the recurrent networks is performed again over 30000 episodes, that correspond to 30000 environments. The episodes are terminated naturally at 100 iterations over the environment dynamics. Similarly to the first experiment, uncertainties are added to the inertia properties, shown below in Table 6.2.6.

Variable	Description	Min value	Max value
m	Spacecraft mass	11 kg	13 kg
I_{zz}	Spacecraft mass moment of inertia in z-axis	5×10^{-2} $kg\ m^2$	6.5×10^{-2} $kg\ m^2$

Table 6.2.6: Uncertainties on the inertia properties of the Chaser for the second experiment.

The results of the training phase for this experiment are reported in Fig. 6.2.1.

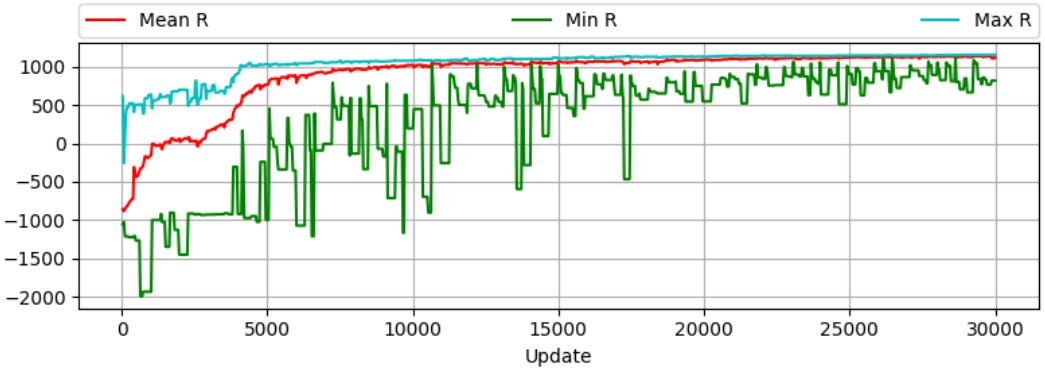


Figure 6.2.1: *Optimization rewards learning curve for the second experiment.*

In this second case, the training is more than successful. The curves of maximum (cyan) and mean (red) rewards are almost overlapped, which means that, on average, the Agent gets as much reward as possible. The same consideration can be done for the curve of minimum reward. Especially, it can be noted how the oscillation of the green curve are limited as the training goes on. To sum up, the policy is optimized.

The curves that show how the terminal position evolves on average and on the best possible case are useful for the reader to justify the success of the training phase, Fig. 6.2.2. The same applies to the curve that shows the mean final attitude angle evolution, Fig. 6.2.3.

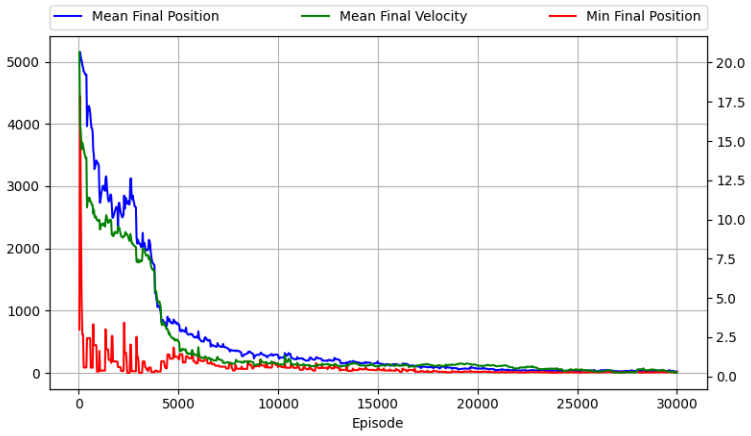


Figure 6.2.2: *Optimization curves of the terminal positions reached by the Chaser, with the associated terminal velocity.* Second case.

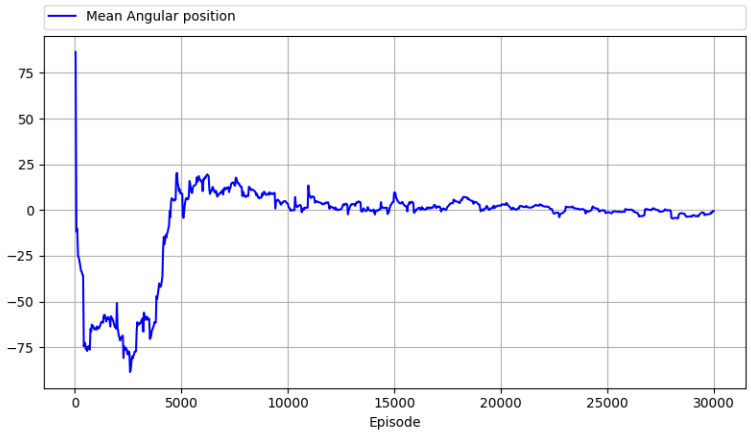


Figure 6.2.3: *Optimization curve of the terminal attitude of the Chaser.* Second case.

Testing

The optimized Agent is again tested over 10000 episodes, each with different initial conditions due to uncertainties, that, as done before, are removed from the inertia properties.
In Fig. 6.2.4, it can be noted that all the 10000 test trajectories end up on a small region, which is close to the target and inside the docking cone. Therefore, all trajectories are acceptable and are showed in Fig. 6.2.5.

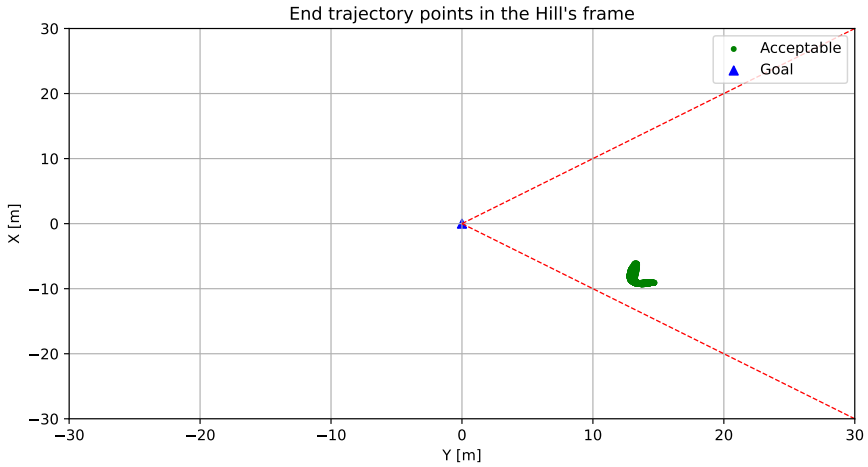


Figure 6.2.4: *Terminal points of the 10000 test trajectories. The green dots represent the acceptable trajectories. The docking cone is displayed in red.* Second experiment.

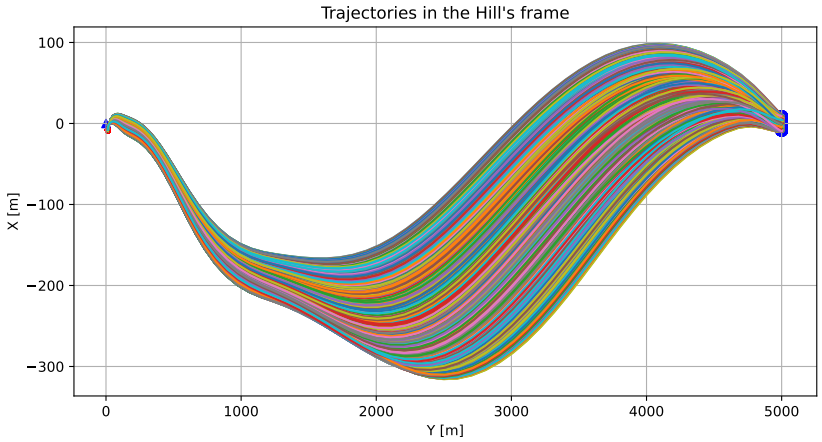
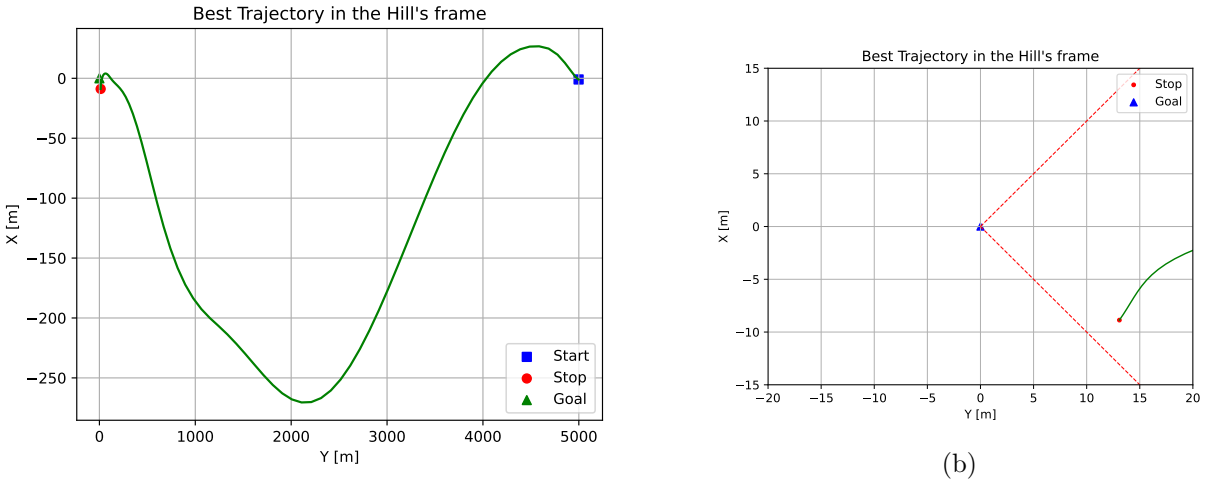


Figure 6.2.5: *Acceptable trajectories for the second experiment.*

With respect to the first experiment, here all the trajectories start from a smaller region, due to the reduction of the uncertainties. However, it can be noted how all of them end up at 15 m from the Target. Even if the mission can not be considered completed, this shows the robustness of a Meta-Reinforcement learning algorithm, when a proper training is conducted.

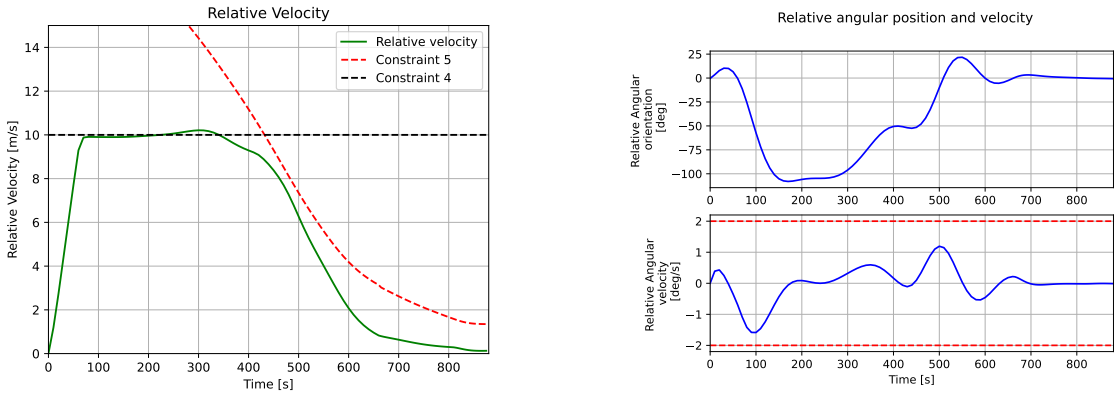
In order to visualize how the velocity and the control commands evolve, it is convenient to select the best trajectory among the 10000 acceptable ones. The criterion for the selection is still based on the closest relative distance from the Target. Hence, the best trajectory for this second experiment is showed in Fig. 6.2.6. The terminal position and the final attitude angle are:

$$||\mathbf{r}|| = 15.8\,m \quad |\theta_N| = 0.63\,deg$$



(a) (b)

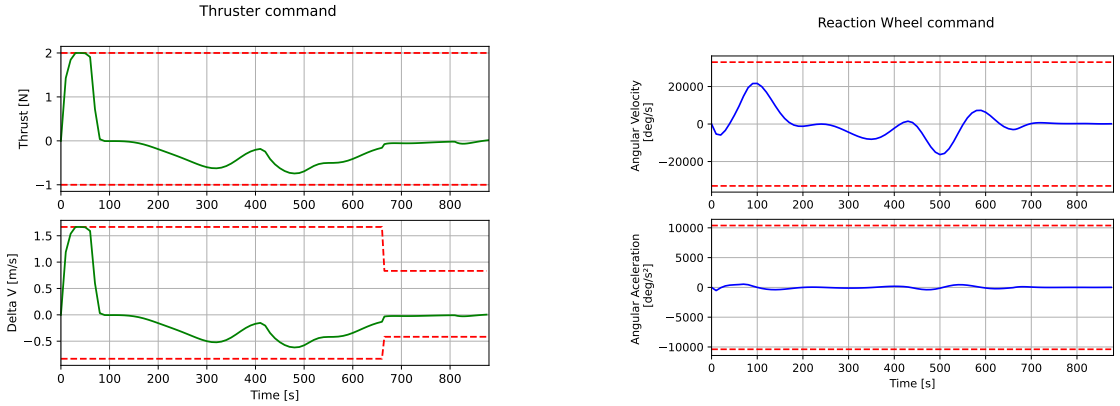
Figure 6.2.6: *Best trajectory in the Hill's reference frame* (a). *Zommed in* (b), *where the docking cone is displayed.*



(a) (b)

Figure 6.2.7: *Relative velocity* (a). *In* (b) *, the attitude angle and the relative angular velocity are illustrated.*

The relative velocity showed in Fig. 6.2.7 is respectful of the constraints almost everywhere. It can be noted, however, that the *constraint* #4 on the recoverable relative velocity is violated before reaching the close proximity operation phase, at $1000\,m$. This implies that the thruster velocity impulse needed to respect the constraint overcomes the physical limitation of the actuator. From a safety viewpoint, the effects of the non compliance of this constraint are not relevant, being the Chaser far enough from the Target.
Finally, the behavior of the actuators in shown in Fig. 6.2.8 (a), for the thruster, and in Fig. 6.2.8 (b), for the reaction wheel.



(a) (b)

Figure 6.2.8: *The commanded thrust and, consequently, the* ΔV*, are shown in* (a). *The reaction wheel commands, scaled by a factor of 10, are shown in* (b).

In this second experiment, the Agent still learned how to behave for a multitude of similar environments. From the graph in terms of commanded controls, it can be noted how the Agent needs only few manoeuvres to properly reach the Target. This is justified from the fact that during

training the Agent explores and performs a longer sequence of actions. After training, however, the policy is fixed and the Agent knows exactly what is the best sequence of actions to take.

Conclusion

The results obtained from the application of Meta-Reinforcement learning for an Autonomous Rendezvous, Proximity Operation and Docking (ARPOD) manoeuvre problem are very promising. Considering the complexity behind the problem in question and the safety requirements, Meta-RL performed discreetly.

The work inherited from [23] stood as a guideline for the implementation of Meta-RL for such a highly constrained and complex problem. The definition of a proper reward function helped in the achievement of a discrete learning. The Meta-reinforcement learning Agent was properly trained to solve the ARPOD problem from a restricted range of possible initial positions. The robustness and adaptability demonstrated in this work justifies the strength of Meta-RL in the solution of different tasks, without the need of a specific training for each of the problems. However, even if the training can be considered successful, the mission should not be considered as achieved. In particular, it was shown how the Agent was able to get very close to the Target spacecraft yet without reaching the final docking phase.
In order to finalize this work, different paths can be followed. The first possibility is to tune properly all the hyperparameters. Indeed, a proper tuning, selected after a high number of trials and errors, could eventually lead to a better solution than the one showed in this master thesis.
Another solution could be to select a different reward function that uniquely teaches the Agent the best possible sequence of actions to take to reach the goal. Finally, a very peculiar solution could be to execute a Meta-RL algorithm to teach the Agent how to reach different locations on the way to the Target spacecraft. Then, after training and testing, patch all the trajectories obtained to finalize a full trajectory. A solution like the one proposed here is surely feasible but very complex, due to the level of randomness that characterizes reinforcement learning algorithms.
To conclude, Meta-RL can surpass classical techniques and other machine learning methods for the solution of very complex problem. This work acts as a proof of the high level of flexibility and robustness that Meta-RL can provide. For future works, the addition of actuator failures or more uncertainties on the dynamics can bring to more complex and realistic problems that can be tackled using a Meta-Reinforcement learning algorithm.

Bibliography

[1] Jonathan R. Behrens and Bhavya Lal. "Exploring Trends in the Global Small Satellite Ecosystem". In: *New Space, The Journal of Space Entrepreneurship and Innovation* 7.3 (2019). URL: https://doi.org/10.1089/space.2018.0017.

[2] Barton C. Hacker. "The Idea of Rendezvous: From Space Station to Orbital Operations in Space-Travel Thought". In: *Technology and Culture* 15.3 (1974). URL: https://www.jstor.org/stable/3102953.

[3] David C. Woffinden and David K. Geller. "Navigating the Road to Autonomous Orbital Rendezvous". In: *Journal of Spacecraft and Rockets* 44.4 (July 2007).

[4] M.S. Smith. *Soviet Space Programs, 1971–75: Overview, Facilities and Hardware, Manned and Unmanned Flight Programs, Bioastronautics, Civil and Military Applications, Project of Future: Program Details of Man-Related Flights*. Ed. by Science Policy Research Division. Vol. 1. Washington : U.S. Govt. Print. Off., 1976. Chap. 3, pp. 173–242.

[5] A. A. Siddiqi. *Challenge to Apollo: The Soviet Union and the Space Race, 1945-1974: Space Politics*. NASA History Division, Rept. SP-4408, 2000. Chap. 9, pp. 351–409.

[6] B. C. Hacker and J. M. Grimwood. *On the Shoulders of Titans: A History of Project Gemini*. Ed. by Rept. SP-4203 NASA. 1977. Chap. Preface, p. 16.

[7] B. C. Hacker and J. M. Grimwood. *On the Shoulders of Titans: A History of Project Gemini: Agena on Trial*. Ed. by Rept. SP-4203 NASA. 1977. Chap. 13, pp. 297–323.

[8] A. A. Siddiqi. *Challenge to Apollo: The Soviet Union and the Space Race, 1945-1974: Getting Back on Track*. NASA History Division, Rept. SP-4408, 2000. Chap. 14, pp. 609–652.

[9] Kasai T. Kawano I. Mokuno M. and Suzuki T. "Result of Autonomous Rendezvous Docking Experiment of Engineering Test Satellite-VII". In: *Journal of Spacecraft and Rockets* 38.1 (Jan. 2001), pp. 105–111.

[10] Mitchell I. T. et al. "GNC Development of the XSS-11 Micro-Satellite for Autonomous Rendezvous and Proximity Operations". In: *AAS Paper 06-014* (2006).

[11] Ya-zhong Luo and Guo-jin Tang. "Spacecraft optimal rendezvous controller design using simulated annealing". In: *Aerospace Science and Technology* 9 (2005). Ed. by Elsevier SAS.

[12] S. McCamish, M. Romano, and X. Yun. "Autonomous Distributed LQR/APF Control Algorith for Multiple Small Spacecraft during Simultaneous Close Proximity Operations". In: Hilton Head, South Carolina: AIAA Guidance, Navigation, Control Conference, and Exhibit, 2007. URL: https://doi.org/10.2514/6.2007-6857.

[13] Liuping Wang. *Model Predictive Control System Design and Implementation Using MATLAB®*. Advances in Industrial Control. Springer, London, 2009.

[14] S. Di Cairano, H. Park, and I. Kolmanovsky. "Model Predictive Control Approach for Guidance of Spacecraft Rendezvous and Proximity Maneuvering". In: *International Journal of Robust and Nonlinear Control* (2012). Ed. by Ltd. John Wiley Sons.

[15] Ian Garcia and Jonathan P. How. "Trajectory Optimization for Satellite Reconfiguration Maneuvers with Position and Attitude Constraints". In: Portland, OR, USA: IEEE, 2005. DOI: https://doi.org/10.1109/ACC.2005.1470072.

[16] Qingfeng Yao et. al. "Path Planning Method With Improved Artificial Potential Field - A Reinforcement Learning Perspective". In: *IEEE Access* 8 (2020). DOI: http://dx.doi.org/10.1109/ACCESS.2020.3011211.

[17] Charles E. Oestreich, Richard Linares, and Ravi Gondhalekar. "Autonomous Six-Degree-of-Freedom Spacecraft Docking with Rotating Targets via Reinforcement Learning". In: *Journal of Aerospace Information Systems* 18 (7 2021). DOI: https://doi.org/10.2514/1.I010914.

[18] Brian Gaudet, Richard Linares, and Roberto Furfaro. *Deep Reinforcement Learning for Six Degree-of-Freedom Planetary Powered Descent and Landing*. 2018. DOI: 10.48550/ARXIV.1810.08719.

[19] Lorenzo Federici et al. "Meta-Reinforcement Learning for Adaptive Spacecraft Guidance during Multi-Target Missions". In: 72nd International Astronautical Congress (IAC), Dubai, United Arab Emirates: IAF, 2021.

[20] Brian Gaudet, Richard Linares, and Roberto Furfaro. "Adaptive Guidance and Integrated Navigation with Reinforcement Meta-Learning". In: *Acta Astronauta* (2020).

[21] Brian Gaudet, Richard Linares, and Roberto Furfaro. "Seeker Based Adaptive Guidance via Reinforcement Meta-Learning applied to Asteroid Close Proximity Operations". In: (2020). DOI: https://doi.org/10.48550/arXiv.1907.06098.

[22] Christopher D. Petersen et al. "Challenge Problem: Assured Satellite Proximity Operations". In: 31st AAS/AIAA Space Flight Mechanics Meeting, online: American Astronautical Society (AAS), American Institute of Aeronautics, and Astronautics (AIAA), 2021.

[23] Matthieu Paris. "Safe ARPOD for under-actuated CubeSat via Reinforcement Learning". MA thesis. Politecnico di Milano, 2021.

[24] Christopher Jewison and R. Scott Erwin. "A Spacecraft Benchmark Problem for Hybrid Control and Estimation". In: IEEE 55th Conference on Decision and Control (CDC), ARIA Resort Casino, Las Vegas, USA: IEEE, 2016.

[25] G. W. Hill. "Researches in the Lunar Theory". In: *American Journal of Mathematics* 1.1 (1878). DOI: https://doi.org/10.2307/2369430.

[26] H.D. Curtis. *Orbital Mechanics for Engineering Students*. Ed. by Butterworth-Heinemann. Third. 2014.

[27] Gene W. Sparrow and Douglas B. Price. "Derivation of Approximate Equations for Solving the Planar Rendezvous Problem". In: *NASA Technical Note D-4670* (1968).

[28] K. H. Gross et al. "Run-time Assurance and Formal Methods Analysis Applied to Nonlinear System Control". In: *Journal of Aerospace Information Systems* 14.4 (2017), pp. 232–246.

[29] Jane X Wang et al. "Learning to reinforcement learn". In: (2016). DOI: 10.48550/ARXIV.1611.05763.

[30] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An introduction*. The MIT Press, 2014,2015.

[31] Evan Zheran Liu et al. "Decoupling Exploration and Exploitation for Meta-Reinfocement Learning without Sacrifices". In: (2021). DOI: https://doi.org/10.48550/arXiv.2008.02790.

[32] Hado Van Hasselt. "Reinforcement Learning in Continuous State and Action Spaces". In: (2013). DOI: http://dx.doi.org/10.1007/978-3-642-27645-3_7.

[33] John Schulman et al. *High-Dimensional Continous Control Using Generalized Advantage Estimation*. 2015. DOI: 10.48550/ARXIV.1506.02438.

[34] Peter W Glynn. "Likelihood ratio gradient estimation for stochastic systems". In: *Communications of the ACM* 33.10 (1990), pp. 75–84.

[35] Ivo Grondman et al. "A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients". In: *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)* 42.6 (2012). DOI: 10.1109/TSMCC.2012.2218595.

[36] John Schulman et al. *Trust Region Policy Optimization*. 2015. DOI: 10.48550/ARXIV.1502.05477.

[37] John Schulman et al. *Proximal Policy Optimization Algorithm*. 2017. DOI: 10.48550/ARXIV.1707.06347.

[38] John Schulman. *Approximating KL Divergence*. 2020. URL: http://joschu.net/blog/kl-approx.html.

[39] Ralf C. Staudemeyer and Eric Rothstein Morris. "Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks". In: (2019). DOI: 10.48550/ARXIV.1909.09586.

[40] Jürgen Schmidhuber Seep Hochreiter. "Long Short-Term memory". In: *Neural Computation* 9.8 (1997). URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[41] Diederik P. Kingma and Jimmy Ba. "Adam: a Method for Stochastic Optimization". In: (2014).

[42] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: (2017).

[43] Sutton R. "Two problems with back propagation and other steepest descent learning procedures for networks". In: *Proceeding of the Eight Annual Conference of the Cognitive Science Society* (1986).

[44] Ning Qian. "On the momentum term in gradient descent learning algorithms". In: *Neural Networks: the official journal of the International Neural Network Society* 12.1 (1999).

[45] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methos for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* 12 (2011).

[46] T. Tieleman and G. Hinton. *Lecture 6.5-rmsprop: Divide the Gradient by a Running Average of Its Recent Magnitude*. COURSERA: Neural Networks for Machine Learning. 2012.

[47] Matthew D. Zeiler. *ADADELTA: An Adaptive Learning Rate Method*. arXiv. 2012. DOI: 10.48550/ARXIV.1212.5701.

[48] Yann LeCun et al. "Efficient BackProp". In: *Neural Networks: tricks of the trade* (1998). Ed. by Springer.

[49] Andrew Y. Ng. "Shaping and policy search in Reinforcement learning". PhD thesis. University of California, Berkeley, 2003.

[50] Marvin Minsky. "Steps toward Artificial Intelligence". In: *Proceedings of the IRE* 49.1 (1961), pp. 8–30. DOI: 10.1109/JRPROC.1961.287775.