POLITECNICO
MILANO 1863

School of Industrial and Information Engineering

Safe ARPOD for under-actuated CubeSat via Reinforcement Learning

Matthieu Paris
ID: 935844

Supervisor: Prof. Pierluigi Di Lizia
Co-supervisor: Michele Maestrini PhD.

A thesis presented for the degree of
Master of Science in Space Engineering

2021

Safe ARPOD for under-actuated CubeSat via Reinforcement Learning

Matthieu Paris

Abstract

From the emergence of commercial applications such as on-demand imagery and global internet service, to the necessity of satellite servicing and active space debris removal, the level of complexity in mission design has skyrocketed. All these different applications have directed the evolution of the technology toward the need for autonomous spacecraft that can operate independently of human control. As such, artificial intelligence has rapidly emerged as being a promising field allowing greater robotic autonomy and innovative decision making. While new autonomous techniques have enabled faster and larger numbers of spacecraft operations, there is still a valid concern for the safety of the missions during proximity manoeuvres.

This Master's thesis investigates the use of the Reinforcement Learning algorithm Proximal Policy Optimization for achieving a planar Autonomous Rendezvous, Proximity Operation, and Docking (ARPOD) manoeuvre with an under-actuated CubeSat. Together with the safety considerations, the different control objectives throughout the three phases reflect the complexity necessary for safe and efficient operations.

The results show both the promise of reinforcement learning methodology and its limitations, which are discussed.

Safe ARPOD for under-actuated CubeSat via Reinforcement Learning

Matthieu Paris

Acknowledgements

First I would like to thanks my supervisor Professor Pierluigi Di Lizia and co-supervisor Michele Maestrini PhD for their support during the six months of work.
I also want to acknowledge the rich content of the DeepMind YouTube channel and 3Blue1Brown YouTube channel. Completing this thesis would have been even more challenging without their online courses and science popularization videos respectively. In addition, the different online contents of Phil Tabor greatly help me to dive into the concrete implementation of reinforcement learning algorithms.

Finally, I want to give a special thanks to John Hughes MA(Cantab.) for his investment of time and intellectual support during my scholarship up to the completion of this work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The space community is in the middle of a complexity paradigm shift. Many are looking to large numbers of heterogeneous small satellites to accomplish missions never attempted before. Commercial interests are looking to constellations of satellites for space-based services like on-demand imagery and global internet service. There are growing companies, new infrastructures and technologies being developed for complicated missions such as satellite servicing and active debris removal [1]. All of this involves a level of complexity never seen before in mission design and operation.

However spacecraft operations today are developed by human teams days in advance to solve a particular task [2] and thus would be difficult to support missions stated above. Indeed, as more complex tasks are introduced, the engineering effort needed to hand-craft solutions may become infeasible.
To close this gap, development in autonomous techniques that are more supportive to evolving complex needs are necessary. This work in particular address the complexity involved in developing autonomy for safe and efficient proximity operations.

While autonomy could enable faster and larger numbers of spacecraft operations, there is a valid concern for the safety of the missions. Rendezvous and docking as an autonomous operation is particularly relevant to the satellite servicing and debris removal applications and it is under continuous development, but mission safety is especially critical and challenging.
The work presented here seeks to meet the above-mentioned needs by providing an overview of Autonomous Rendezvous, Proximity Operation, and Docking (ARPOD) for an under-actuated spacecraft via reinforcement learning. The problem objective and safety constraints introduced reflect the complexity necessary for safe and efficient rendezvous and docking.

1.1 A Brief History of Space Rendezvous and Docking

In some respects, the birth of orbital rendezvous came in the 1960s during the height of the space race between the United States and the Soviet Union. It was during this era that orbital rendezvous transformed from a mere concept to reality. [3]

Some may claim that the first orbital rendezvous occurred on 12 August 1962 when the Russian Vostok 4 spacecraft piloted by Pavel Popovich was launched into orbit and came within 6.5 km of Vostok 3 [4]. However, neither spacecraft had the necessary maneuvering capability to maintain their relative position, and so they eventually drifted over 850 km apart before the end of the day. The Vostok program was analogous to the United States Mercury program, whose primary objective was to place an astronaut into Earth orbit.
Three years later, The NASA's Gemini program served as a bridge between the path-breaking but limited Earth-orbital missions of Project Mercury and the unprecedented lunar missions of Apollo. Consequently, Gemini was first and foremost a project to develop and prove equipment and techniques for orbital rendezvous and docking [5]. The goal was manned orbital rendezvous and at this time, NASA considered autonomy as a nicety, not a necessity. On 15 December 1965, the first ever orbital rendezvous occurred between Gemini VI and Gemini VII. Several months later, on 16 March 1966, the first docking between two spacecraft finally occurred when Neil Armstrong and Dave Scott docked Gemini VIII with an Agena target vehicle [6]. During this program, the astronauts' effective display of detecting and resolving critical mission problems in real time seemed to reinforce NASA's position of using manual control over autonomous systems.
By the spring of 1967, the Soyuz program accomplished the first automated rendezvous and docking between two piloted vehicles [4]. Unlike Gemini, the Soyuz vehicle was designed primarily for automated orbital rendezvous with piloted capabilities generally reserved for contingency operations. After Soyuz 1 crashed during its reentry manoeuvre killing Vladimir Komarov, manned operations came to a temporary halt, automated missions continued to move forward. Under the cover name of Kosmos, in October 1967 the first ever rendezvous and docking between two robotic spaceships was performed [7].

The great accomplishments and technical developments that have been achieved with regard to orbital rendezvous are slowly being overshadowed by their limitations to meet new demands. During the 1990s, the idea of performing rendezvous maneuvers autonomously without necessitating complex communication schemes between spacecraft while incorporating light weight, low power, compact navigation sensors has become a sought-after ideal for a variety of missions. With this in mind, the National Space Development Agency of Japan (NASDA) created the Engineering Test Satellite VII (ETS-VII) flight experiment. And on 7 July 1998, ETS-VII successfully performed the first autonomous rendezvous and docking procedure between uninhabited spacecraft [8].
Finally, commissioned by the U.S. Air Force Research Laboratory and under the direction of the Lockheed Martin Space Systems Company, the Experimental Satellite System-11 (XSS-11) program started. This program had the mandate to develop and verify on-orbit guidance, navigation, and control capabilities to safely and autonomously rendezvous a micro-satellite [9]. It was launched on 11 April 2005 and by the fall of 2005 it had performed over 20 rendezvous maneuvers. Although provisions were made to allow ground controllers to interact with the vehicle, XSS-11's on-board planner could autonomously guide the spacecraft by selecting from a variety of operational modes. The spacecraft was not simply operating automatically, but had the unique capacity to also respond to various situations autonomously.

1.2 Overview of Autonomous Control Techniques

Since XSS-11 program has demonstrated that a spacecraft could be autonomously guided and could respond to various situations autonomously, much research has been made to extend the benefit of autonomous controller.

An algorithm for multiple small spacecraft during simultaneous close proximity operations was then developed [10]. It combines the control effort efficiency of a Linear Quadratic Regulator (LQR) and the robust collision avoidance capability of Artificial Potential Function (APF) methods. The LQR control effort serves as the attractive force toward goal positions, while APF-based repulsive functions provide collision avoidance for both fixed and moving obstacles. This research have been carried on up to a novel autonomous control technique featuring real-time fuel optimization and the previous LQR/APF algorithm [11].
The well-established Linear Quadratic Regulators are methods with a long heritage that has also been implemented in a receding horizon fashion to simulate a closed-loop response. This so-called Model Predictive Control (MPC) is able to use dynamically re-configurable constraints by adjusting the trajectory based on the current state of the vehicle, which is likely to be different than the predicted result with the purely open-loop algorithm.
Hence, algorithms for a three degrees of freedom docking [12] and hybrid rendezvous and docking [13] were developed. These algorithms are able to deal with the notion of fuel minimisation and obstacle avoidance. However, algorithms based on classical control techniques suffer from their close dependency on the mathematical framework. Indeed, the non-linearity of the attitude dynamics, the non-convexity of some of the constraints, and the coupling between the positions and attitudes of all spacecraft make the problem almost impossible to solve with such techniques.

To overcome this limitation, path planner algorithms have been applied to spacecraft control. This technique typically operates by prescribing a set of way points for the satellite to follow, although in some cases the path is defined using a smoothed curve such as a series of splines.
It has been shown that it is possible to design spacecraft reconfiguration manoeuvres for up to four spacecrafts with six degrees of freedom [14] where Rapidly-exploring Random Trees is used as path planner. The algorithm generates a trajectory consisting of a sequence of states, connected by feasible direct trajectories. The result is then passed through a smoother to improve the cost of the trajectory previously found.
Path planner algorithms are open-loop ones in which the trajectory and high-level controls are computed once at the beginning of the manoeuvre and this path is followed until manoeuvre termination. Consequently, evolving constraints such as moving obstacles can not be handled.

Motivated by the previous techniques' limitations, a new approach that builds upon a branch of machine learning has been introduced. This last technique, called reinforcement learning, augments the guidance capabilities of spacecraft for difficult tasks.

1.3 Reinforcement learning in Space

Among the machine learning framework, reinforcement learning instructs the computer by providing feedback to repeated attempts at solving a given problem. Since it is a general, model-free framework, reinforcement learning is potentially advantageous over model-based methods for scenarii where model identification is infeasible or prohibitive, e.g., when environments, dynamics, or disturbances are time-varying. The engineering effort is reduced to specifying a reward system rather than the complete logic. Therefore, by selecting an appropriate reward scheme, complex behaviours can be learned by the Agent without being explicitly programmed.
Moreover, once learned, implementation of the policy requires low amounts of computational effort and memory, making it practically realizable with current spacecraft computing resources.

Most modern algorithms use artificial neural networks to approximate nonlinear functions that decide on the actions to take. Deep reinforcement learning, which uses multilayer artificial neural networks, made headlines in the mid-2010s, in part due to the groundbreaking work of Google Deep Mind.
In 2015 a computer was trained with a deep Q-learning algorithm to play 49 different classic Atari games, 29 of which were at a human or superior skill level [15]. The following year, a computer agent was capable of beating the European champion of the Chinese game of Go by mastering this complex game of an estimated 4.9×10^{359} possible combinations of moves [16].
This was followed by a further development in which the agent demonstrated super-human performance via pure reinforcement learning [17] and testifies to the reinforcement learning's ability to learn novel solutions. These developments demonstrated the potential of reinforcement learning solutions to other seemingly intractable problems that suffer from the curse of dimensionality.

Its extension to spacecraft guidance, navigation, and control is still a source of active research. Recent efforts include the application of the REINFORCE algorithm for asteroid mapping [18] and the use of an reinforcement learning actor-critic framework to solve path constraints in near-rectilinear orbits [19]. Proximal policy optimization was used for producing six degrees of freedom planetary landings and asteroid hovering maneuvers [20].
A lot of work has also been done in the field of rendezvous, proximal optimization and docking. In 2019, J.Broida and R.Linares [21] used a proximal policy optimization to compute three degrees of freedom rendezvous trajectories without considering the attitude dynamic. Later in 2020, C.E. Oestreich, R.Linares, and R.Gondhalekar [22] developed six degrees of freedom docking manoeuvres where the rewards were based on a reference value provided by a linear quadratic regulator feedback law.
Finally, Hovell and Ulrich [23] presented a guidance policy for three degrees of freedom proximity operations using the "distributed distributional deep deterministic policy gradient" (D4PG) algorithm, testing it successfully in granite surface hardware experiments. The hardware implementation distinguishes this work from most research efforts that are limited to simulation.

1.4 Impact and Outline

From the emergence of commercial applications such as on-demand imagery and global internet service, to the necessity of satellite servicing and active space debris removal, the level of complexity in mission design has skyrocketed. All these different applications have directed the evolution of the technology toward the need for autonomous spacecraft that can operate independently of human control.

Recently, the interesting potential of reinforcement learning to solve complex time varying problems has made it an important source of research in guidance, navigation and control. This work aims to explore the capacity of reinforcement learning algorithms to solve a three degrees of freedom rendezvous, proximity operation, and docking manoeuvre for an under-actuated spacecraft system with safety constraints. While research has already been done in this direction, none of it investigates the possibility to perform a full ARPOD with a pure reinforcement learning algorithm.

The remainder of the document will be organized as follows. In chapter 2, a complete explanation of the necessary reinforcement learning theory is provided. Starting from a popularization of the scientific concept, it introduces the different notions in order to end with a clear understanding of the algorithm used in this work: the Proximal Policy Optimization. In chapter 3, an overview of the mechanisms of an artificial neural network is made. It aims to provide the reader with a brief knowledge concerning this specific tool allowing the learning mechanism. Chapter 4 introduces the problem of autonomous rendezvous, proximity operation and docking. Hence, it formalizes the models and dynamics of this specific problem. And it presents the constraints needed to compute a feasible and safe manoeuvre. In chapter 5, the problem is implemented into the framework of chapters 2 and 3. Finally, chapter 6 studies different test cases to conclude on the ability of the Agent to solve this highly constrained problem.

Chapter 2

Reinforcement Learning

Reinforcement learning is learning what to do by interacting with the *environment* to get the best outcome. The learner, also called *Agent* is not told which actions to take, but instead must discover which actions yield the most benefits.

Reinforcement learning is different from supervised learning which learns from a training set of labelled examples provided by a knowledgeable external supervisor. This is an important kind of learning, but alone it is not adequate for learning from interaction. It is also different from what machine learning researchers call unsupervised learning, which is typically about finding structure hidden in collections of unlabelled data. One might be tempted to think of reinforcement learning as a kind of unsupervised learning because it does not rely on examples of correct behaviour; however reinforcement learning tries to optimize a reward signal instead of trying to find hidden structures.

One of the challenges that arise in reinforcement learning, and not in other kinds of learning, is the trade-off between *exploration* and *exploitation*. The *Agent* might want to exploit the *environment* based on its current knowledge to maximise the total outcome. But to discover such actions, it has to try actions that it has not selected before. Hence, it also has to explore in order to make better action selections in the future.
The dilemma is that neither *exploration* nor *exploitation* can be pursued exclusively without failing at the task. The *Agent* must try a variety of actions and progressively favour those that appear to be best. Such a dilemma has been intensively studied by mathematicians for many decades, yet remains unresolved. [24]

A daily life analogy is faced when we want to find the best restaurant in our city. This situation is similar to reinforcement learning framework in the sense that the *environment* is our city and we are the *Agent* who wants to find the best restaurant, meaning the restaurant that yields the best score. To do so, we try them by interacting with our *environment*. However, some of us will favor what they think is the best restaurant. They will then favor *exploitation* to have a high cumulative score in the short-term. On the other hand, another might prefer to try new restaurants by *exploring* the city. This last one might have the best long-term strategy by gathering information in order to make the best overall choice. But by doing so, he/she is going to sacrifice the short-term cumulative score without any certainty on the long-term score.
This is the so-called *exploration/exploitation* dilemma.

2.1 Elements of Reinforcement Learning

Beyond the *Agent* and the *environment*, one can identify three main sub-elements in a reinforcement learning system: a *policy*, a *reward signal*, and a *value function*. [24]

A *policy* defines the learning Agent's way of behaving at a given time. Roughly speaking, a *policy* is a mapping from perceived states of the environment to actions to be taken when in those states. The *policy* is the core of a reinforcement learning Agent in the sense that it alone is sufficient to determine behaviour. In general, *policies* may be stochastic, specifying probabilities for each action.

A *reward signal* defines the goal of a reinforcement learning problem. At each interaction, the environment sends to the Agent a single number called the *reward*. The Agent's sole objective is to maximize the *total reward* it receives in the long run; if an action selected by the policy is followed by low *reward*, then the action that has been taken is "bad", and the policy may be changed to select some other action in that situation in the future.

Whereas the reward signal indicates what is good in an immediate sense, a *value function* specifies what is good in the long run. Roughly speaking, the value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state.

To continue with the previous analogy, the *policy* is the strategy we follow based on our current knowledge of the restaurants in the city. Each time we go to a restaurant, we get a score based on how good it was, this is the *reward signal* that we seek to maximise. Finally, the *value function* is the cumulative score we expect to have at a given time with a given knowledge. It is then easy to see that maximising the *rewards* favours the short-term strategy, while maximising the *value function* might lead to the best long-term strategy.

2.2 Markov Decision Process

A more formal description of reinforcement learning must start with the *Markov decision process*, which provides the underlying structure for the learning process. [24]

In a *Markov decision process*, time is subdivided into discrete time steps, $t = 0, 1, 2, ...$ The full description of the Agent necessary to define its condition within the environment is known as its *state*, $S_t \in \mathscr{S}$. The *action* that the Agent takes to try and solve the problem at time step t is $A_t \in \mathscr{A}(S_t)$. Notice how the chosen *action* is a function of the *state*. After the Agent selects an *action*, the environment returns a *subsequent state* S_{t+1} and *reward* $R_{t+1} \in \mathscr{R}$. This *state* then becomes the current *state* as the Agent advances into the subsequent time step. This cyclical process is illustrated in figure 2.1.
This process continues until some end condition is met: the Agent may have succeeded in its task, met a user defined failure condition, or simply reached a limit on the number of time steps. Each attempt at solving the problem is known as an *episode*.

The *Markov decision process* and Agent together thereby give rise to a sequence or *trajectory* that begins like this:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \cdots$$

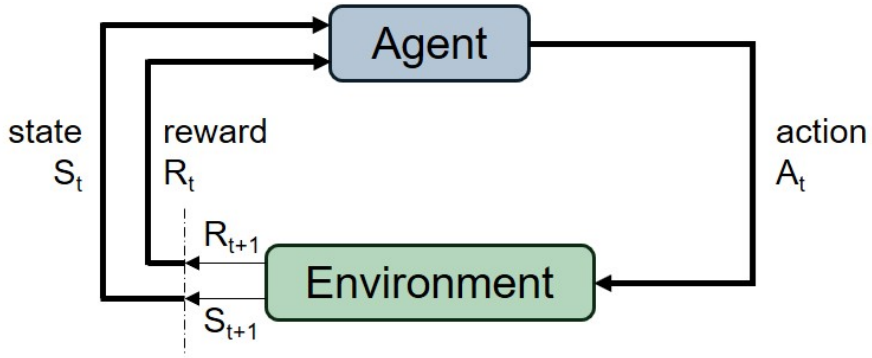


Figure 2.1: Agent - Environment cycle

Common formulations of Markov decision processes make the assumption that they are *finite*. This requires that the sets $\mathscr{S}$, $\mathscr{A}$, and $\mathscr{R}$ all have *discrete*, *finite spaces*. As a result, the random variables for the state and reward at a time step t, S_t and R_t have *discrete probability distributions* dependent only on the preceding state and action. This provides the most important property of a Markov decision process: *the Markov Property*. In other words, the future is independent of the past at a given time step t.

Keeping this in mind, it is possible to define the probability of the agent transitioning to a particular state $\mathbf{s'} \in \mathscr{S}$ and collecting a given reward $r \in \mathscr{R}$ solely as a function of the current state $\mathbf{s} \in \mathscr{S}$ and the action associated with it $\mathbf{a} \in \mathscr{A}(\mathbf{s})$. This is known as the *dynamics* of the Markov decision process. [24]

$$p(\mathbf{s'}, r|\mathbf{s}, \mathbf{a}) = prob(S_{t+1} = \mathbf{s'}, R_{t+1} = r|S_t = \mathbf{s}, A_t = \mathbf{a}) \tag{2.1}$$

The definition 2.1 of the *dynamics function* $p : \mathscr{S} \times \mathscr{R} \times \mathscr{S} \times \mathscr{A} \to [0, 1]$ is an ordinary deterministic function of four arguments. For the sake of clarity, one must remember p as a probability distribution for each choice of $\mathbf{s}$ and $\mathbf{a}$.

$$\forall \mathbf{s} \in \mathscr{S}, \forall \mathbf{a} \in \mathscr{A}(\mathbf{s}), \sum_{\mathbf{s'} \in \mathscr{S}} \sum_{r \in \mathscr{R}} p(\mathbf{s'}, r|\mathbf{s}, \mathbf{a}) = 1 \tag{2.2}$$

From the four-argument *dynamics function*, p, one can compute anything else one might want to know about the environment, such as the *state-transition probabilities* (which it is denoted, with a slight abuse of notation, as a three-argument function $p : \mathscr{S} \times \mathscr{S} \times \mathscr{A} \to [0, 1]$). This provides the probability of a state $\mathbf{s'}$ at the next time step given a state $\mathbf{s}$ and action $\mathbf{a}$ at the current time step. This marginal probability can be found by summing the joint probability defined in equation 2.1 over all possible rewards in the set $\mathscr{R}$.

$$p(\mathbf{s'}|\mathbf{s}, \mathbf{a}) = prob(S_{t+1} = \mathbf{s'}|S_t = \mathbf{s}, A_t = \mathbf{a}) = \sum_{r \in \mathscr{R}} p(\mathbf{s'}, r|\mathbf{s}, \mathbf{a}) \tag{2.3}$$

Similarly, the *expected reward* given a state $\mathbf{s}$ and action $\mathbf{a}$ can be found using equation 2.1 and the definition of an *expected value*; The *expected value* of a finite and countable random variable is defined as the sum of all possible values of that variable weighted by their probabilities. Therefore, the *expected rewards* for state–action pairs is a two-argument function $r : \mathscr{S} \times \mathscr{A} \to \mathscr{R}$.

$$r(\mathbf{s}, \mathbf{a}) = \mathbb{E}[R_t | S_t = \mathbf{s}, A_t = \mathbf{a}] = \sum_{r \in \mathscr{R}} r \sum_{\mathbf{s'} \in \mathscr{S}} p(\mathbf{s'}, r | \mathbf{s}, \mathbf{a}) \tag{2.4}$$

As mentioned previously, this formulation is designed solely for *discrete* state, action, and reward spaces. This is a critical limitation. Fortunately, one can easily modify equations 2.3 and 2.4 for *continuous* random variables [25]. Recall equation 2.3, which gave the probability of a specific state $S_{t+1} = \mathbf{s'}$. This was a conditional probability mass function. For the *continuous* case, this is replaced by a conditional probability density function $T : \mathscr{S} \times \mathscr{A} \times \mathscr{S}' \to [0, 1]$ which denotes the probability that action $\mathbf{a}$ in state $\mathbf{s}$ results in a transition to a state in the region $\mathscr{S}' \subseteq \mathscr{S}$.

$$\int_{\mathscr{S}'} T(\mathbf{s}, \mathbf{a}, \mathbf{s'})\, d\mathbf{s'} = prob(S_{t+1} \in \mathscr{S}' | S_t = \mathbf{s}, A_t = \mathbf{a}) \tag{2.5}$$

For a reinforcement learning problem with a *discrete* action space, an action can be selected by simply selecting the value for A_t that has the highest probability from equation 2.3. With a *continuous* action space, an action can be sampled from the probability distribution T.
Regarding the reward function defined by equation 2.4, it would be possible to calculate a *continuous* version using a probability density function, however, this is not necessary in a practical implementation.

2.3 Policy Gradient Methods

An Agent is not a monolithic system, but rather contains sub-components that control its various functions. When an Agent chooses an action, it does so according to its policy π. This policy maps an observation O_t of the Agent's environment to an action A_t and transforms the Agent into a closed-loop controller.

This section focuses on methods that learn a parameterized policy that can select actions without consulting a value function. In other words, learning a strategy that does not require the knowledge of the expected cumulative rewards. A value function may still be used to learn the policy parameter, but is not required for action selection.

2.3.1 Policy Approximation

For continuous problems, this policy is a neural network that takes the observation input signal O_t and defines a median and standard deviation as an output in order to define a probability distribution. This is written $\pi_\theta(\mathbf{a}|O_t = \mathbf{o})$, where θ are the parameters of the neural network.
The distinction between observation and state is a subtle one: In some scenarii, the observation may contain more information than is included in the state alone. But by assuming that the Markov decision process is fully observable, the state and observation are one and the same. Hence, one can write the probability that action $\mathbf{a}$ is taken at time t given the environment in state $\mathbf{s}$ at time t with parameters θ.

$$\pi_\theta(\mathbf{a}|\mathbf{s}) = \pi(\mathbf{a}|\mathbf{s},\theta) = prob(A_t = \mathbf{a}|S_t = \mathbf{s}, \Theta_t = \theta) \tag{2.6}$$

After some user-selected number of episodes, the policy parameters are updated based on the information contained within the tuples (S_t, A_t, R_{t+1}). After many updates, the Agent eventually converges to an optimal policy denoted π^*.
Before tackling the concepts used to learn this optimal policy, another concept needs to be formalized : the value function $V^{\pi,\gamma}(\mathbf{s})$.

2.3.2 Value Function and Advantage

In order to make intelligent decisions, an Agent must consider not only the immediate consequences of its actions but also their long term ramifications. With a reward being a quantification of only a single state transition, taking decisions by following the highest reward could result in a poor policy. To correct this deficiency, the value of a state is defined as the expected value of the sum of discounted future rewards. Assuming a discounting factor $\gamma < 1$,

$$V^{\pi,\gamma}(\mathbf{s_t}) = \mathbb{E}\left[\sum_{\tau=0}^{\infty} \gamma^\tau r_{t+\tau}\right] \tag{2.7}$$

The value function can be thought of as being a measure of the quality of a trajectory, with the horizon of this estimate controlled by the discounting factor. The discounting factor also controls the greediness of the Agent. In other words, it quantifies how the Agent prioritizes the immediate reward over long term rewards.
In a practical implementation, the recursive form of equation 2.7 is more convenient.

$$V^{\pi,\gamma}(\mathbf{s_t}) = \mathbb{E}\left[r_t\right] + \gamma V^{\pi,\gamma}(\mathbf{s_{t+1}}) \tag{2.8}$$

While the value function translates the expected outcome according to the current policy, it does not determine whether a given action leads to a better or worse outcome than anticipated. Thereby, the Agent has to consider the true value of each state that it encountered using the collected rewards against its current value function. This is known as the advantage function. [26]

$$A^{\pi,\gamma}(\mathbf{s_t}, \mathbf{a_t}) = Q^{\pi,\gamma}(\mathbf{s_t}, \mathbf{a_t}) - V^{\pi,\gamma}(\mathbf{s_t}) \tag{2.9}$$

with $Q^{\pi,\gamma}(\mathbf{s_t}, \mathbf{a_t})$ the state-action value function : it is an estimate of the value function starting from state $\mathbf{s_t}$ and following the selection of action $\mathbf{a_t}$. Since the Agent is exploring, this will likely not be the best action according to its policy. Hence, in equation 2.9, 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.
As such, the advantage would be positive when the chosen action leads to a trajectory that has a better outcome than the Agent's current policy. In other words, the advantage quantifies how much better the taken action is, based on the expectation of what would have happened from the current policy.
In practice, the advantage $A^{\pi,\gamma}$ is not known and must be estimated.

2.3.3 Generalized Advantage Estimation

Unfortunately, the variance of the advantage estimator scales unfavorably with the time horizon, since the effect of an action is confounded with the effects of past and future actions.
A common strategy is to make the use of a value function rather than the empirical returns. Doing so leads to estimators with lower variance at the cost of introducing bias. But while high variance necessitates using more samples, bias is more pernicious and can cause the Agent to fail to converge. Consequently, an important challenge is to find an advantage estimation with an effective variance reduction scheme without introducing too much bias. [26]

Before any further considerations, the TD error has to be introduced. [24]

$$\delta_t^{V^{\pi,\gamma}} = r_t + \gamma V^{\pi,\gamma}(\mathbf{s_{t+1}}) - V^{\pi,\gamma}(\mathbf{s_t}) \tag{2.10}$$

The TD error at each time step represents the error in the estimate made at that time. Hence, $\delta_t^{V^{\pi,\gamma}}$ can be considered as an estimate of the advantage of the action $\mathbf{a_t}$. In fact, if we have the correct value function, then it is an unbiased estimator of $A^{\pi,\gamma}$.

$$A^{\pi,\gamma}(\mathbf{s_t}, \mathbf{a_t}) = \mathbb{E}\left[\delta_t^{V^{\pi,\gamma}}\right] \tag{2.11}$$

One can notice that the TD error in equation 2.10 depends on the next state and next reward and it is not actually available until one time step later. That is, $\delta_t^{V^{\pi,\gamma}}$ quantifies the error in $V^{\pi,\gamma}(\mathbf{s_t})$, available at time $t+1$.
However, in practice only an estimation of the value function is known and it will yield a biased advantage estimate. [26]

$$\hat{A}_t = \delta_t^{\widehat{V}} \tag{2.12}$$

With $\widehat{V}$ standing for an approximation of the value function $V^{\pi,\gamma}$ and $\hat{A}_t$ representing the advantage estimation.

A solution for reducing the bias is to take the sum of k of these $\delta_t^{\widehat{V}}$ terms, which is denoted by $\hat{A}_t^{(k)}$.

$$\begin{aligned} \hat{A}_t^{(k)} &= \sum_{\tau=0}^{k-1} \gamma^\tau \delta_{t+\tau}^{\widehat{V}} \\ &= -\widehat{V}(\mathbf{s_t}) + r_t + \gamma r_{t+1} + \cdots + \gamma^{k-1} r_{t+k-1} + \gamma^k \widehat{V}(\mathbf{s_{t+k}}) \end{aligned} \tag{2.13}$$

$\hat{A}_t^{(k)}$ results in a telescoping sum that involves a k-step estimate of the returns, minus a baseline term $\widehat{V}(\mathbf{s_t})$. Analogously to the equation 2.12, it can be considered to be an estimator of the advantage function. However, note that the bias generally becomes smaller as $k \to \infty$, since the term $\gamma^k \widehat{V}(\mathbf{s_{t+k}})$ becomes more heavily discounted, and the baseline $\widehat{V}(\mathbf{s_t})$ does not affect the bias.

It can be noticed that a valid variance reduction scheme can be done not just toward any k-step estimator, but toward any average of k-step estimators. One particular way

of averaging is to consider all the k-step estimators, each weighted proportionally to λ^{k-1} (where $\lambda \in [0,1]$), and normalized by a factor of $1-\lambda$ to ensure the weights sum to 1. This estimation scheme, parameterized by γ and λ is called the generalised advantage estimator $GAE(\gamma,\lambda)$. [26]

$$\begin{aligned}\hat{A}_t^{GAE(\gamma,\lambda)} &= (1-\lambda)\left(\hat{A}_t^{(1)} + \lambda\hat{A}_t^{(1)} + \lambda^2\hat{A}_t^{(2)} + \cdots\right) \\ &= \sum_{\tau=0}^{\infty}(\gamma\lambda)^\tau \delta_{t+\tau}^{\widehat{V}}\end{aligned} \tag{2.14}$$

The construction used above is closely analogous to the one used to define TD(λ) [24], however TD(λ) is an estimator of the value function, whereas here it is an estimate of the advantage function.
For the sake of clarity, it is worth analysing the two special cases of this equation 2.14 obtained by setting $\lambda = 0$ and $\lambda = 1$.

$$\hat{A}_t^{GAE(\gamma,0)} = \quad \hat{A}_t \quad = \delta_t^{\widehat{V}} \tag{2.15}$$

$$\hat{A}_t^{GAE(\gamma,1)} = \hat{A}_t^{(\infty)} = \sum_{\tau=0}^{\infty}\gamma^\tau \delta_{t+\tau}^{\widehat{V}} \tag{2.16}$$

$\hat{A}_t^{GAE(\gamma,1)}$ has high variance due to the sum of terms. On the other hand, $\hat{A}_t^{GAE(\gamma,0)}$ induces bias, but it typically has much lower variance. It is then clearer that the generalised advantage estimator makes a compromise between bias and variance, controlled by the parameter λ.

The described advantage estimator uses two separate parameters γ and λ. Both of them contribute to the bias-variance trade-off when using an approximate value function. However, they serve different purposes and work best with different ranges of values : γ most importantly determines the scale of the value function $V^{\pi,\gamma}$ which does not depend on λ. Taking $\gamma < 1$ introduces bias in the estimate. On the other hand, $\lambda < 1$ introduces bias only when the value function is inaccurate.
As a rule of thumb, the best value of λ is lower than the best value of γ because λ introduces far less bias than γ for a reasonably accurate value function. [26]

2.3.4 Policy Gradient Theorem

Policy gradient methods work with a parameterized policy, introduced in section 2.3.1, and optimize an objective function $J(\theta)$ directly over the parameter space of the policy.
These methods seek to maximize performance, so their updates approximate gradient ascent in J. More details about neural network mechanism and gradient ascent are given in chapter 3.

$$\theta_{t+1} = \theta_t + \alpha\widehat{\nabla_\theta J(\theta_t)} \tag{2.17}$$

Where α is the learning rate, and $\widehat{\nabla_\theta J(\theta_t)}$ is a stochastic estimate whose expectation approximates the policy gradient $\nabla_\theta J(\theta_t)$ with respect to its parameters θ.

One can easily guess that the most straightforward objective function is the value function $V^{\pi_\theta,\gamma}(\mathbf{s_0})$ which stands for the expected sum of discounted rewards $\sum_{t=0}^{\infty}\gamma^t r_t$. The gradient is thus defined as follows.

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t r_t\right] \tag{2.18}$$

However, there are several different related expressions for the policy gradient [26]. By using the generalized advantage estimator, one can construct a biased estimator of the policy gradient.

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_{t=0}^{\infty}\nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\hat{A}_t^{GAE(\gamma,\lambda)}\right] \tag{2.19}$$

For the sake of clarity, the gradient should be the derivative of the objective function. Therefore, the objective function corresponding to the biased estimator of the policy gradient in equation 2.19 is worth giving.

$$J^{PG}(\theta) = J(\theta) = \mathbb{E}\left[\sum_{t=0}^{\infty}\log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\hat{A}_t^{GAE(\gamma,\lambda)}\right] \tag{2.20}$$

This objective function is very interesting even though it is less intuitive than the value function; if the advantage estimate is positive, meaning that the action the Agent took in the sample trajectory resulted in a better average return, then the probability of selecting this action will increase. On the other hand, the probability of taking an action will decrease if the advantage estimate is negative. In addition, the use of the generalized advantage estimate offers an effective variance reduction scheme at the cost of adding bias.

2.3.5 Actor-Critic Framework

To compute the generalized advantage estimation introduced in section 2.3.3, the TD error used an approximation of the value function as seen in equation 2.12. Therefore, the Actor-Critic framework is needed to ease the understanding of further concepts.

Indeed, the algorithm presented so far can be referred to as an Actor-Critic algorithm: the policy gradient algorithms, also named Actor-only methods, used an Actor to learn the parametrized policy. At the same time, the Critic approximates and updates the value function. The value function is then used to update the Actor's policy.
Rather than only learning the policy or the value function and then backing out the other, both are optimized simultaneously. While not perfect, the Actor-Critic framework has good convergence properties and is a popular structure for designing reinforcement learning algorithms [27]. One can notice that, like the Actor-only methods, the policy action space can remain continuous.

To implement an Actor-Critic algorithm, two neural networks are required: an Actor and a Critic. The Actor neural network produces the parametrized policy $\pi_\theta(\mathbf{a}|\mathbf{s})$, while the Critic calculates the Agent's best estimate of the value function, denoted by $\widehat{V}_{\mathbf{w}}(\mathbf{s})$. Note that θ and $\mathbf{w}$ respectively stand for the parameters of the Actor and Critic neural network.
Upon completing some number of episodes, the rewards collected from many steps are batched together and the two networks are then updated to maximise their objective functions. The objective function for the policy depends on the algorithm, but so far it has been described as:

$$J_t^{PG}(\theta) = \mathbb{E}_t \left[\log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \hat{A}_t^{GAE(\gamma,\lambda)} \right] \tag{2.21}$$

In order to update the Critic, the objective function is defined as the squared-error between the true values and the Critic's assessment. The true value can be seen as the expected discounted return G_t^π based on the collected reward.

$$V^{\pi,\gamma}(\mathbf{s_t}) = \mathbb{E}\left[G_t^\pi|\mathbf{s_t}\right] \tag{2.22}$$

with

$$G_t^\pi = r_t + \gamma G_{t+1}^\pi \tag{2.23}$$

In a more practical way, one can derive G_t^π from the advantage function, and analogously from the generalized advantage estimation.

$$G_t^\pi = \hat{A}_t^{GAE(\gamma,\lambda)} + V^{\pi,\gamma}(\mathbf{s_t}) \tag{2.24}$$

Hence, the objective function for the value function which tries to optimize the Critic in order to minimize the equation 2.9 is defined as follows. Note that $-J_t^{VF}(\mathbf{w})$ has to be considered since a gradient ascent optimization is performed.

$$J_t^{VF}(\mathbf{w}) = \mathbb{E}_t \left[\left(G_t^\pi - \widehat{V}_{\mathbf{w}}(\mathbf{s}_t) \right)^2 \right] \tag{2.25}$$

2.4 Proximal Policy Optimisation

The problem reinforcement learning suffers from, is that the training data that are generated are themselves dependent on the current policy. Hence, the Agent generates its own training data by interaction with the environment rather than relying on a static data set. It means that the data distribution over observations and rewards is constantly changing as the agent learns. This is a major cause of instability in the learning process: if the policy update is too large, it will push the policy network to a region of the parameter space where it is going to collect the next bunch of data on a very poor policy, causing it to never recover again. In other words, the Agent can lose its understanding of the environment if the Actor suddenly explores the action space leading to very poor performances.

Indeed, repeatedly performing optimization on J_t^{PG} using the same trajectory has been shown to lead to destructively large policy updates [26]. This is solved by limiting the changes in the policy to small updates.

2.4.1 Trust Region Method

A successful approach is to make sure that the updates of the policy never lead too far away from the old policy. This idea is widely introduced in the Trust Region Policy Optimization algorithm (TRPO) [28]. The objective function used in this last algorithm is defined in equation 2.26.

$$J_t^{TRPO}(\theta) = \mathbb{E}_t \left[\frac{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\theta_{\mathbf{old}}}(\mathbf{a}_t|\mathbf{s}_t)} \hat{A}_t \right] \tag{2.26}$$

subject to

$$\mathbb{E}_t \left[KL \left[\pi_{\theta_{\mathbf{old}}}(.|\mathbf{s}_t), \pi_\theta(.|\mathbf{s}_t) \right] \right] \leq \xi \tag{2.27}$$

Here, $\pi_{\theta_{\mathbf{old}}}$ is the vector of policy parameters before the update.
To make sure that the update policy does not move too far away from the current policy, a Kullback-Leibler divergence constraint is added to the optimization objective. This so call Kullback-Leibler divergence is a measure of the difference between two distributions.

One can notice that the only difference with the policy gradient objective function presented in equation 2.20 is the use of the policy ratio $R_t(\theta)$.

$$R_t(\theta) = \frac{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\theta_{\mathbf{old}}}(\mathbf{a}_t|\mathbf{s}_t)} \tag{2.28}$$

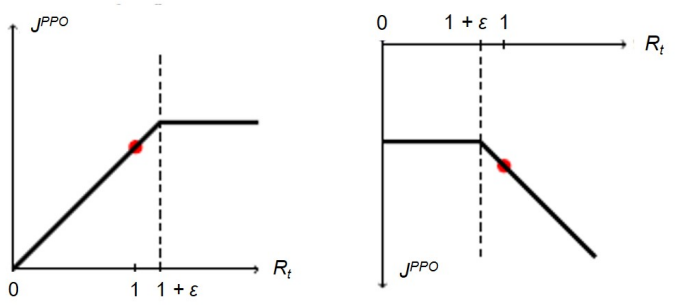
This ratio is the probability of an action occurring under the new policy π_θ, divided by the probability as it occurred under the previous policy $\pi_{\theta_{\mathbf{old}}}$. Then, the algorithm attempts to maximize the ratio for a given action and the advantage calculated for that action. For instance, if an action had a detrimental result, the action should be made less likely: $R_t < 1$. Since the advantage function for an undesirable action is negative, the maximization of equation 2.26 will produce the least negative $R_t(\theta)\hat{A}_t$ possible. In doing so, it will make the probability of the action occurring under the new policy π_θ as small as possible, subject to the constraint on the KL divergence from equation 2.27.

2.4.2 Clipped Surrogate Objective

While TRPO is a powerful algorithm, it adds additional overhead and is also complex to implement among other limitations.
The Proximal Policy Optimization (PPO) has some of the benefits of trust region policy optimization, but it is much simpler to implement, more general, and has better sample complexity [29]. The objective function used in the PPO algorithm is the expectation of the minimum of two terms. The first term is the ratio-advantage product, carried over from TRPO, which pushes the policy toward actions that yield a high positive advantage over the baseline. The second term is a truncated version of the policy ratio obtained by applying a clipping operation.

$$J_t^{PPO}(\theta) = \mathbb{E}_t \left[min \left(R_t(\theta)\hat{A}_t,\ clip(R_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{2.29}$$

To better understand the meaning of this objective, one needs to give more attention to the effect of each term. The effect of the min operator is changed according to the sign of the advantage estimate.



(a) $\hat{A}_t > 0$ (b) $\hat{A}_t < 0$

Figure 2.2: Plots showing one term (i.e. a single time step) of the objective function J_t^{PPO} as a function of the probability ratio R_t, for positive advantages (a) and negative advantages (b).

When the advantage function is positive, the action results in a better than expected outcome. In such a case, the optimization seeks to make this outcome more likely with $R_t > 1$. To avoid any excessive changes to the policy, the change in probability is limited to a value ϵ. For instance, if $\epsilon = 0.2$, R_t may not exceed 1.2 and the action may not become more than 20% more likely (Figure 2.2a). The limit applies in the opposite direction for undesirable actions; the likelihood may not be reduced below 80% of its previous value (Figure 2.2b).

2.4.3 Algorithm

Now that the different objective functions have been properly introduced, one should be knowledgeable enough to tackle the full PPO algorithm.

In practice, a surrogate objective function combining the policy objective function (equation 2.29) and the value function error term (equation 2.25) is used. This objective can further be augmented by adding an entropy bonus H to ensure sufficient exploration [30]. The entropy can be seen as a measure of how unpredictable the outcome of a variable really is. In this case, maximising the entropy of the policy will force a wild spread over all the possible options, and therefore force to have more exploration.

$$J_t^{PPO+VF+H}(\theta, \mathbf{w}) = \mathbb{E}_t \left[J_t^{PPO}(\theta) - c_1 J_t^{VF}(\mathbf{w}) + c_2 H\left(\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\right) \right] \tag{2.30}$$

Finally, the PPO algorithm is presented hereafter.

Algorithm 1 Proximal Policy Optimisation algorithm

Initialization of neural network parameters θ and $\mathbf{w}$
Initialization memory of length T
for episode = 1, 2, ..., E **do**
    Reset the environment
    while not done **do**
        for t = 0, 2, ..., T **do**
            $\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ ▷ Choose an action
            $\mathbf{s}_{t+1}; r_{t+1}; done \leftarrow Env(\mathbf{s}_t; \mathbf{a}_t)$ ▷ play the action
            Record $\mathbf{s}_t$, $\mathbf{a}_t$ and r_{t+1} in memory ▷ store transition
        for epoch = 1, 2, ..., K **do**
            compute $\hat{A}_t^{GAE(\gamma,\lambda)}$ for each recorded state ▷ equation 2.14
            for each batch of size M **do**
                Compute Entropy H
                Compute policy objective function J_t^{PPO} ▷ equation 2.29
                Compute critic objective function J_t^{VF} ▷ equation 2.25
                Compute surrogate objective function $J_t^{PPO+VF+H}$ ▷ equation 2.30
                Perform a gradient ascent on the Actor parameters θ
                Perform a gradient ascent on the Critic parameters $\mathbf{w}$
        Clear memory

Chapter 3

Artificial Neural Networks

Artificial Neural Networks (ANNs) are tools that provide a means for complex, non-linear functions to be modelled purely based on input and output data without any knowledge of the form of the function itself. As such, ANNs are widely used in machine learning. In the PPO algorithm introduced previously, two of them are used: the Actor and the Critic that take as input the current state and return respectively the actions and the best estimate of the value function.
It is worth devoting a chapter to this specific tool, to better understand how these two ANNs manage to learn.

3.1 Artificial Neural Network Structures

An Artificial Neural Network is a machine learning algorithm based on the model of a human brain. The brain consists of millions of interconnected neurons connected with a special structure known as synapses. The same idea is applied in an Artificial Neural Network: units, also referred to as neurons, are interconnected with arcs in order to exchange and process information. Units are organized in sets called layers.
Different types of networks can be found in the deep learning world with different structures. The three most common ones are listed hereafter.

- Feed-Forward Neural Network: input data are processed only in the forward direction.
- Recurrent Neural Network: a recurrent connection on the hidden units is added with respect to the previous structure. This looping constraint ensures that sequential information is captured in the input data.
- Convolution Neural Network: Kernels are used to extract the relevant features from the input using the convolution operation.

In this work, the Feed-Forward Neural Network is used. Such a network consists of one input layer, one output layer, and one or more hidden layers that are neither receiving information from the external nor presenting the final network outputs.
Figure 3.1 shows a generic Feed-Forward Neural Network where the units are represented by circles.

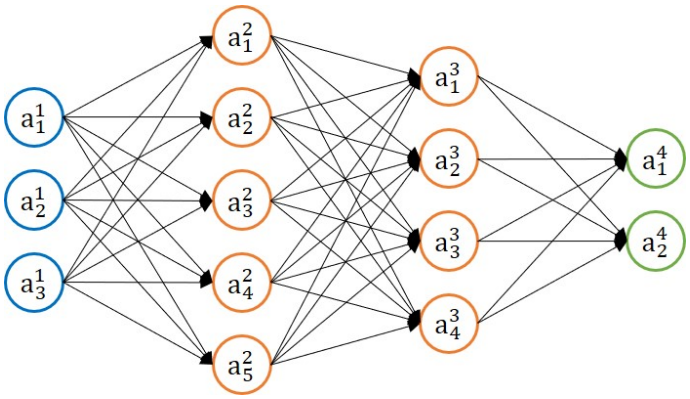


Figure 3.1: A generic feed-forward neural network with an input layer of three neurons (blue), an output layer of two neurons (green), and two hidden layers of respectively five and four neurons (orange).

3.2 Forward Propagation and Execution

Before focusing on the key algorithms allowing neural networks to learn, it is worth discussing how to use them to produce an output given an input.

Neurons are typically semi-linear units represented by scalar values named *activations* and denoted as a. This *activation* is the result of a non-linear function σ, called *activation function*, that has for input a weighted sum computed from a weight matrix $\mathbf{w}$ and a bias matrix $\mathbf{b}$. In figure 3.1, each arrow connecting two neurons represents an arc characterised by a weight. On the contrary, a bias is somehow linked to each neuron.
In a Feed-Forward Neural Network, the neurons' *activation* of layer l is computed from the *activations* of the layer $l-1$ with the following algebra.

$$\begin{aligned} \mathbf{p}^l &= \mathbf{w}^l \mathbf{a}^{l-1} + \mathbf{b}^l \\ \mathbf{a}^l &= \sigma\left(\mathbf{p}^l\right) \end{aligned} \tag{3.1}$$

Here, the weighted sum, also named *preactivation*, is computed separately and denoted as $\mathbf{p}$. This decomposition will make sense in the explanations hereafter.

Different *activation functions* are used, but they are typically sigmoid functions such as the logistic function and tanh, though rectified linear unit (ReLU) is also widely used. The non-linearity of *activation functions* is essential: if all the neurons in a multi-layer Feed Forward Neural Network have linear activation functions, then the entire network cannot model a non-linear behaviour because linear functions of linear functions are themselves linear.

In the next section, it will be necessary to look at equations 3.1 at the scale of each neuron, rather than the layer. In this manner, a new subscript notation is used. In equations 3.2, the *activation* of the j neuron in layer l is computed.

$$\begin{aligned} p_j^l &= \sum_k w_{jk}^l a_k^{l-1} + b_j^l \\ a_j^l &= \sigma\left(p_j^l\right) \end{aligned} \tag{3.2}$$

Note that the weight from the weights matrix $\mathbf{w}$ receives two subscripts to denote the link between two specific neurons in two different layers.

3.3 Backpropagation

One may remember that the overall idea behind the Reinforcement Learning algorithm is to optimize an objective function. In the PPO algorithm presented in section 2.4, this appears as the optimization of the objective function $J_t^{PPO+VF+H}$ by updating the parameter space of both the Actor and the Critic artificial neural network.
Before updating the neural network parameters, the first thing to do is to determine how sensitive the objective function is on the different parameters of the network. This is what stands behind the backpropagation algorithm.

The following explanation considers a single training example with an objective function J and a generic Feed-Forward Neural Network made of L layers. The goal is then to quantify how sensitive is this objective function to small changes in the parameter space. In other words, it is to compute the partial derivative of the objective function with respect to the weights and biases.

Let us first understand the concept with the simple layer notation introduced in equations 3.1. One could interpret it as a network with a single neuron per layer.

Much as forward propagation demonstrates how the inputs work their way through a neural network, backpropagation starts from the outputs and moves back up the network. Therefore, the first consideration must be done on the last layer characterised by the activation $\mathbf{a}^L$ and the preactivation $\mathbf{p}^L$.
By applying the chain rule and equations 3.1, the rate of change of the objective function according to the preactivation of neurons is found.

$$\begin{aligned} \frac{\partial J}{\partial \mathbf{p}^L} &= \frac{\partial J}{\partial \mathbf{a}^L}\frac{\partial \mathbf{a}^L}{\partial \mathbf{p}^L} \\ &= \frac{\partial J}{\partial \mathbf{a}^L}\sigma'\left(\mathbf{p}^L\right) \end{aligned} \tag{3.3}$$

Where $\sigma'\left(\mathbf{p}^L\right)$ stands for the rate of change of the activation function. To provide some insight into the deeper meaning of this equation, the first term represents how sensitive the objective function is to the neuron output. The second term is a measure of the speed with which the activation function is changing.

The name backpropagation takes all its meaning when it comes to computing this partial derivative at any previous layer of the network based on the layer ahead of it. Indeed, this calculation is repeated, starting from the final layer, and propagated backwards towards the input layer. The algebra is shown in equation 3.4 for an arbitrary layer l by differentiating equations 3.1.

$$\begin{aligned}\frac{\partial J}{\partial \mathbf{p}^l} &= \frac{\partial J}{\partial \mathbf{p}^{l+1}} \frac{\partial \mathbf{p}^{l+1}}{\partial \mathbf{a}^l} \frac{\partial \mathbf{a}^l}{\partial \mathbf{p}^l} \\ &= \frac{\partial J}{\partial \mathbf{p}^{l+1}} \mathbf{w}^{l+1} \sigma' \left(\mathbf{p}^l\right)\end{aligned} \tag{3.4}$$

Finally, the sensitivity of the objective function to small changes in the weights and biases is derived by applying two simple chain rules. For any arbitrary layer, equations 3.5 are derived from equations 3.1.

$$\begin{aligned}\frac{\partial J}{\partial \mathbf{w}^l} &= \frac{\partial J}{\partial \mathbf{p}^l} \frac{\partial \mathbf{p}^l}{\partial \mathbf{w}^l} = \frac{\partial J}{\partial \mathbf{p}^l} \mathbf{a}^{l-1} \\ \frac{\partial J}{\partial \mathbf{b}^l} &= \frac{\partial J}{\partial \mathbf{p}^l} \frac{\partial \mathbf{p}^l}{\partial \mathbf{b}^l} = \frac{\partial J}{\partial \mathbf{p}^l}\end{aligned} \tag{3.5}$$

These equations are the final pieces of the backpropagation algorithm. Nevertheless, the same expressions must be derived at the scale of each neuron to be used. Hopefully, the reasoning is the same as previously. Just the full subscript notation introduced in equations 3.2 has to be added. Consequently, the derivation will not be repeated. Note that in this case, a neuron influences the objective function through multiple different paths. So all these paths need to be summed up.
Therefore, the rate of change of the cost function with respect to an arbitrary neuron j in the final layer L or in any arbitrary layers are respectively similar to equation 3.3 and 3.4.

$$\frac{\partial J}{\partial p_j^L} = \frac{\partial J}{\partial a_j^L} \sigma' \left(p_j^L\right) \tag{3.6}$$

$$\frac{\partial J}{\partial p_j^l} = \sum_k \frac{\partial J}{\partial p_k^{l+1}} w_{kj}^{l+1} \sigma' \left(p_j^l\right) \tag{3.7}$$

And the final expressions representing the sensitivity of the objective function to small changes in the weights and biases are similar to equations 3.5.

$$\begin{aligned}\frac{\partial J}{\partial w_{jk}^l} &= \frac{\partial J}{\partial p_j^l} a_k^{l-1} \\ \frac{\partial J}{\partial b^l} &= \frac{\partial J}{\partial p_j^l}\end{aligned} \tag{3.8}$$

In summary, the backpropagation algorithm computes how a training example would like to nudge the weights and biases. Not just in terms of up and down but in terms of what relative proportion to those changes causes the most rapid optimisation in the objective function.

3.4 Gradient Descent and Adam Optimizer

Now that the gradient of the objective function with respect to the parameter space is known, it is time to optimize the neural network.
Gradient descent is one of the most popular algorithms for performing optimization and by far the most common way to optimize neural networks. This section aims at providing the readers with intuitions about the behaviour of the different algorithms of gradient descent optimization up to the one used in this project: the Adam Optimizer.

3.4.1 Gradient Descent Variants

Gradient descent is a way to minimize an objective function $J(\theta)$ parameterized by model parameters θ. To do so, the 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 ends when the gradient is equal to zero. This will at least lead to a locally optimal solution based on the data used to calculate the gradient, where this data are known collectively as a batch.
In other words, we follow the direction of the slope of the surface created by the objective function downhill until we reach a valley.

For the sake of clarity, it is worth specifying that the PPO algorithm used in this work aims to maximise an objective function. Hopefully, one is not without knowing that just a sign convention stands between maximisation and minimisation. Hence, the maximisation is performed with a gradient descent algorithm.

The most straightforward intuition is to use the entirety of a dataset as the batch.

$$\theta = \theta - \eta \nabla_\theta J(\theta) \tag{3.9}$$

This is known as the Batch Gradient Descent, aka Vanilla Gradient Descent. This method can be slow, faces memory issues, and does not enable the model to be updated online. In other words, it is unable to accept new information into the data set as it becomes available. Within the Reinforcement Learning framework, this limitation makes its use impossible.

Stochastic Gradient Descent (SGD) is a variation of this method that is appropriate for neural network optimization. In contrast with the Vanilla Gradient Descent, it performs a parameter update for each randomly selected data point ξ^i to optimize the network to produce ψ^i.

$$\theta = \theta - \eta \nabla_\theta J(\theta, \xi^i, \psi^i) \tag{3.10}$$

With the gradient calculations no longer requiring the entire dataset, new data can easily be added during the process. However, the use of single data points to compute gradients makes individual updates noisy. On one hand, it enables the optimizer to jump to new and potentially better local minima. But on the other hand, the noisiness ultimately complicates convergence to the exact minimum.

Vanilla Mini-batch Gradient Descent, or simply Mini-batch Gradient Descent finally

takes the best of both previous optimizers and performs an update for every small batch of data of size n randomly selected from the complete dataset.

$$\theta = \theta - \eta \nabla_\theta J(\theta, \xi^{i:i+n}, \psi^{i:i+n}) \tag{3.11}$$

In this way, it reduces the variance of the parameter updates, which can lead to more stable convergence; and makes use of highly optimized matrix optimizations that lead to a very efficient gradient computation. [31]

Mini-batch Gradient Descent is typically the algorithm of choice when training a neural network. To stay coherent with the literature, in the rest of this chapter the term SGD is employed to refer to mini-batch optimization. In addition, to ease the notation, the parameters $\xi^{i:i+n}$ and $\psi^{i:i+n}$ in equation 3.11 are left out.

3.4.2 Gradient descent optimization algorithms

SGD optimization appears to be slow where the error basin is long and narrow, i.e. the surface curves much more steeply in one dimension than in another. In such a situation and with a fixed learning rate in the direction of the gradient, the algorithm tends to oscillate back and forth along the semi-major axis [32].
It has been demonstrated that by adding a momentum term based upon the gradient from the previous mini-batch, these oscillations could be dampened [33]. The momentum-based methods use a weighted sum of the previous gradient $\mathbf{v}_{u-1}$ and the current gradient $\nabla_\theta J(\theta)$ to updates the parameters.

$$\begin{aligned} \mathbf{v}_u &= \zeta \mathbf{v}_{u-1} + \eta \nabla_\theta J(\theta) \\ \theta &= \theta - \mathbf{v}_u \end{aligned} \tag{3.12}$$

A simple way to understand this algorithm is to picture a ball going downhill. The ball accumulates momentum as it rolls downhill, becoming faster and faster until it reaches its terminal velocity. The idea is the same with the parameter updates: the momentum term increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change direction. As a result, faster convergence and reduced oscillation are gained. One drawback of these methods is that the momentum of the gradient is built without any regard for the shape of the basin. Additionally, while multiple parameters are being updated, all of the methods mentioned so far use the same learning rate.

The algorithm Adagrad [34] has been developed in order to overcome this. Adagrad algorithm adapts the learning rate to the parameters, performing smaller updates (i.e. low learning rates) for parameters associated with frequently occurring features, and larger updates (i.e. high learning rates) for parameters associated with infrequent features. In its update rule, the algorithm modifies the general learning rate η at each step u for each parameter $\theta_i \in \theta$ based on the past gradients that have been computed for the same θ_i. To achieve this, the algorithm defines a diagonal matrix $\mathbf{G}_u$ where each of the diagonal terms is the sum of the outer product of the gradients from all previous steps.

$$\mathbf{g}_{u,h} = \nabla_{\theta_u} J(\theta_{u,h}) \tag{3.13}$$

$$\mathbf{G}_u = \sum_{\tau=1}^{u} \mathbf{g}_u \mathbf{g}_u^T \tag{3.14}$$

The Adagrad update logic is then displayed in equation 3.15 with the added term κ ensuring that division by zero is impossible.

$$\theta_{u+1,h} = \theta_{u,h} - \frac{\eta}{\sqrt{\mathbf{G}_{u,hh} + \kappa}} \mathbf{g}_{u,h} \tag{3.15}$$

Adagrad's main weakness is its accumulation of the squared gradients in the denominator. Since every added term is positive, the accumulated sum keeps growing during training and causes the learning rate to eventually become infinitesimally small. Consequently, sooner or later, the algorithm is no longer able to acquire additional knowledge.

To solve this flaw, the unpublished neural network optimization algorithm named RMSprop has been developed [35]. Here, the sum of gradients is recursively defined as a decaying average of all past squared gradients. The running average $\mathbb{E}\left[\mathbf{g}^2\right]_u$ at step u then only depends on the previous average and the current gradient.

$$\begin{aligned} \mathbb{E}\left[\mathbf{g}^2\right]_u &= 0.9\mathbb{E}\left[\mathbf{g}^2\right]_{u-1} + 0.1\mathbf{g}_u^2 \\ \theta_{u+1} &= \theta_u - \frac{\eta}{\sqrt{\mathbb{E}\left[\mathbf{g}\right]_u + \kappa}} \mathbf{g}_u \end{aligned} \tag{3.16}$$

Note that the weighting coefficients of 0.9 and 0.1 are recommended values.

3.4.3 Adam Optimizer

Finally, the optimization algorithm used in this work is described: Adaptive Moment Estimation (Adam) [36]. Adam algorithms have become among the most popular optimization methods in Reinforcement Learning research.

In addition to storing an exponentially decaying average of past squared gradients $\mathbf{v}_u$ like Adagrad and RMSprop algorithms, it also keeps an exponentially decaying average of past gradients $\mathbf{m}_u$ similar to momentum-based methods.

$$\begin{aligned} \mathbf{m}_u &= \beta_1 \mathbf{m}_{u-1} + (1 - \beta_1)\, \mathbf{g}_u \\ \mathbf{v}_u &= \beta_2 \mathbf{v}_{u-1} + (1 - \beta_2)\, \mathbf{g}_u^2 \end{aligned} \tag{3.17}$$

$\mathbf{m}_u$ and $\mathbf{v}_u$ are respectively estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients, hence the name of the algorithm.
The uncentered nature of this last estimate is important. These estimates are initialized to zero, and thus are biased towards that value especially during the initial steps. To counteract it, bias correction is necessary.

$$\begin{aligned} \hat{\mathbf{m}}_u &= \frac{\mathbf{m}_u}{1-\beta_1^u} \\ \hat{\mathbf{v}}_u &= \frac{\mathbf{v}_u}{1-\beta_2^u} \end{aligned} \tag{3.18}$$

The parameter update is then accomplished using a form identical to the Adagrad and RMSprop algorithms.

$$\theta_{u+1} = \theta_u - \frac{\eta}{\sqrt{\hat{\mathbf{v}}_u} + \kappa}\hat{\mathbf{m}}_u \tag{3.19}$$

To sum up, the Adam algorithm requires four hyperparameters, which are settings provided by the user to control the behaviour of the learning process:

- The exponential decay rates β_1 and β_2, for which the default values of 0.9 and 0.999 are recommended, respectively.
- The term preventing a division by zero, κ, for which the value 10^{-8} is suggested.
- The learning rate η that will be specified later on for the Actor and the Critic networks.

Chapter 4

Autonomous Rendezvous, Proximity Operation, and Docking

So far, the concept of reinforcement learning was introduced without any consideration of the physics of the problem. Indeed, reinforcement learning algorithms are among the so-called model-free algorithms. This means that the Agent does not require any knowledge of the problem, only the environment in which it is taking actions is specific to a given problem. Hence, problems with very complex or even unknown dynamics could be solved in theory. This chapter starts with a precise description of the problem the Agent has to solve. Then the spacecraft model and the dynamics of a planar Autonomous Rendezvous, Proximity Operation, and Docking (ARPOD) with three degrees of freedom are formalized. Finally, the different constraints leading to a safe and feasible solution are introduced.

Before dealing with the formalization of the problem, one may wonder where are the boundaries between rendezvous, proximity operation and docking manoeuvres. Even if there is no wrong answer as long as the docking is the final manoeuvre, a benchmark is worth being given for clarity. This work considers a benchmark problem for hybrid control during ARPOD [37] described in 2016. It enables techniques to be compared and contrasted with each other by enumerating the phases of the ARPOD mission, including the sensors and dynamics available at each phase.

- The far rendezvous phase describes the approach of one spacecraft to another usually in the range of 10 km to 1 km.
- The close rendezvous phase, which can also be found as the proximity operation phase, takes place in the range of 1 km to 100 m. This phase must end in the line of sight region of the Target docking port.
- Then the docking phase describes the final manoeuvre executed to reach the docking port within the line of sight region. It covers the range from 100 m to 0 m. During this last phase, the attitude must be controlled to perform a successful docking.

4.1 Problem Formulation

The problem considers two spacecraft orbiting around the Earth, with the overall goal to dock both satellites together. One spacecraft, called the Target will be the subject of docking. The other, called the Chaser, is controlled to approach the Target and docks with it by ensuring the safety of itself and the Target at all times.
In this work, a successful docking is achieved when the Chaser reaches a distance of $1\,m$ from the Target within the Target docking port line of sight. To safely dock, the Chaser must reach the Target with a maximum absolute relative velocity of $0.2\,m\,s^{-1}$ and with an absolute relative angle lower than $5\,degrees$.
In addition, it is assumed that the Target does not rotate during the entire manoeuvre, and the Chaser knows its relative position and attitude with respect to the Target.

Given the above statements, the problem can be summarized as follows. Note that the notation will be properly introduced later on.

An omniscient Chaser satellite and a still Target satellite must achieve the following:

- *The Chaser must remain within the Target docking port line of sight during the entire docking phase.*
- *Safety of both the Chaser and the Target must be maintained throughout the duration of the manoeuvre.*
- *The Chaser must asymptotically dock with the Target.*

$$\|\mathbf{r}\| \leq 1\,m \text{ and } |\theta| \leq 5\,deg$$

4.2 Model of the Chaser

The model of the controlled Chaser used in this work is a 6U CubeSat measuring $10cm \times 20cm \times 30cm$, as pictured in figure 4.1.

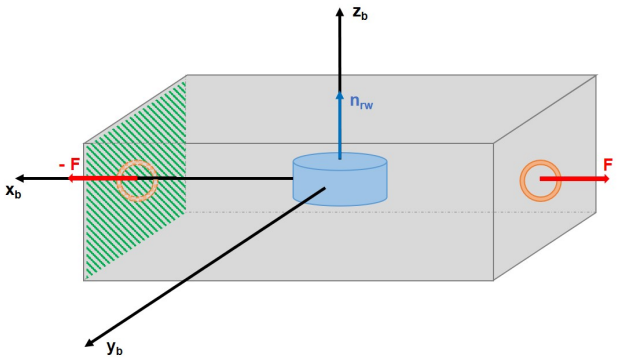


Figure 4.1: 6U CubeSat Chaser with thrusters (red) aligned with the positive and negative body x-axes, a reaction wheel (blue) aligned with the body z-axis, and the docking port (green) normal to the positive body x-axes

In this model, thrust is only possible from two thrusters assumed to be perfectly aligned with the spacecraft body x-axis $\mathbf{x_b}$ on either side of the spacecraft. By convention, a positive total thrust F induces a displacement in the direction of $\mathbf{x_b}$. The attitude of the Chaser about its z-axis $\mathbf{z_b} \equiv \mathbf{n_{rw}}$ is controlled by a reaction wheel. And finally, the docking port of the Chaser is considered as being the surface with outward normal vector $\mathbf{x_b}$.
Specific inertial values for the Chaser can be found in Table 4.1. Note that while numeric values are given, the techniques developed can be used for other similar satellite configurations.

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	$12\, kg$

Table 4.1: Inertia properties of the Chaser

4.3 Dynamics

To begin with, a set of assumptions are made to derive the equations of motion.

Assumption 1 *Both satellites are rigid bodies.*

Assumption 2 *The mass of the Earth is significantly greater than the mass of the satellites.*

Assumption 3 *The mass loss of the Chaser is significantly smaller than the total mass of the spacecraft.*

Assumption 1 applies to most modern spacecraft as fuel slosh and moving mass is typically not a significant part of spacecraft vehicle dynamics. *Assumption 2* consolidates the magnitude of gravity into the gravity parameter μ. *Assumption 3* results in the mass remaining constant and is reasonable as propellant usage over short time intervals is not tremendous.
Under these assumptions, both the Target and Chaser revolve around the Earth, governed by the following equation of motion in the inertial geocentric equatorial frame.

$$\ddot{\mathbf{R}}_j = -\frac{\mu}{R_j^3}\mathbf{R}_j \tag{4.1}$$

Where $j \in \{t, c\}$ is the subscript denoting respectively the Target and the Chaser, $\mathbf{R}_j$ is the position vector of the spacecraft, and $R_j = ||\mathbf{R}_j||$.
The equation of motion 4.1 describes a stable elliptical orbit about the centre of gravity of the frame, which due to *assumption 2*, is approximately the centre of the Earth.

The following additional assumptions are made to linearize the spacecraft dynamics.

Assumption 4 *The Target is in a circular orbit with a radius R_t.*

Assumption 5 *The distance from the Target to the Chaser is significantly less than that of the distance from the Target spacecraft to the centre of the Earth.*

Assumption 4 is reasonable as a good number of satellites are on near-circular orbits. In this work, the altitude of the Target is fixed at 500 km. Concerning *assumption 5*, rendezvous typically happens on the order of tens of kilometres, while the distance to satellites from the Earth is on the order of at least thousands of kilometres.

With the Target in an equilibrium orbit due to *assumption 4*, a non-inertial frame is attached to it, specifically for the purposes of rendezvous: *the Hill's Frame* [38]. The axes of the frame are defined as follow:

- $\mathbf{e_r}$ for the radial direction that points outward from the Earth's centre.
- $\mathbf{e_N}$ for the normal direction that is aligned with the angular momentum vector of an orbit which is constant and always points orthogonal to the orbital plane.
- $\mathbf{e_t}$ for the tangential direction which completes an orthogonal coordinate system with $\mathbf{e_r}$ and $\mathbf{e_N}$. Note that for a circular orbit, $\mathbf{e_t}$ and the inertial orbital velocity are aligned.

In such a frame, the relative position vector of the Chaser with respect to the Target is denoted as $\mathbf{r}$.

$$\mathbf{r} = x\,\mathbf{e_r} + y\,\mathbf{e_t} + z\,\mathbf{e_N} \tag{4.2}$$

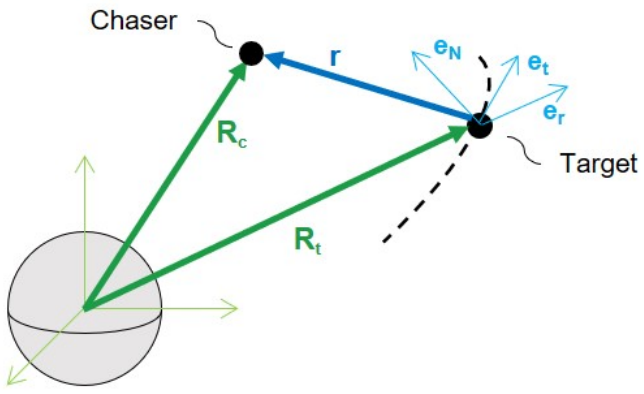


Figure 4.2: Position of the Chaser and the Target in both frames: the inertial frame in green associated with capital letters, and the Hill's frame in blue associated with small letters.

In the Hill's frame, the equation of motion of the Chaser can be written in the following simple form.

$$\begin{aligned} \ddot{x} - 2\sqrt{\frac{\mu}{R_t^3}}\,\dot{y} - 3\frac{\mu}{R_t^3}\,x = 0 \\ \ddot{y} + 2\sqrt{\frac{\mu}{R_t^3}}\,\dot{x} = 0 \\ \ddot{z} + \frac{\mu}{R_t^3}\,z = 0 \end{aligned} \tag{4.3}$$

It is furthermore true for circular orbits that the mean motion of the Target $n = \sqrt{\frac{\mu}{R_t^3}}$ is constant. Note that the linear equations of motion 4.3 can be decoupled into the orbital in-plane motion which lives in the ($\mathbf{e_r}$, $\mathbf{e_t}$) plane, and out-of-plane motion.
While under more complex modelling, these equations of motion become re-coupled, it is outside the scope of this problem [39]. In this work, only in-plane motion is considered. Therefore, equations 4.3 may be rewritten as follows.

$$\begin{aligned} \ddot{x} - 2n^2\,\dot{y} - 3n\,x = 0 \\ \ddot{y} + 2n\,\dot{x} = 0 \end{aligned} \tag{4.4}$$

These are known as the *planar Clohessy-Wiltshire equations*. The coefficients in these equations are constant. Consequently, a straightforward analytical solution exists. Note that the subscript $\cdot_0$ refers to the quantity at $t = 0$.

$$\begin{aligned} x &= (4 - 3\cos nt)\,x_0 + \frac{\sin nt}{n}\,\dot{x}_0 + \frac{2}{n}(1 - \cos nt)\,\dot{y}_0 \\ y &= 6(\sin nt - nt)\,x_0 + y_0 + \frac{2}{n}(\cos nt - 1)\,\dot{x}_0 + \frac{1}{n}(4\sin nt - 3nt)\,\dot{y}_0 \\ \dot{x} &= 3n\sin nt\,x_0 + \cos nt\,\dot{x}_0 + 2\sin nt\,\dot{y}_0 \\ \dot{y} &= 6n(\cos nt - 1)\,x_0 - 2\sin nt\,\dot{x}_0 + (4\cos nt - 3)\,\dot{y}_0 \end{aligned} \tag{4.5}$$

The rotational equation of motion is dictated by the conservation of the angular momentum. With regards to the above planar case, the rotational degree of freedom is only about the $\mathbf{e_N}$ axis. As the inertia of the Chaser about that axis is I_{zz}, the inertia of the reaction wheel about that spin axis is then D.
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. In this work the Target does not rotate. Consequently, θ_N is also the relative rotation of the Chaser with respect to the Target. By convention, θ_N is measured from the inward normal vector of the Target docking port $-\mathbf{n_t}$.

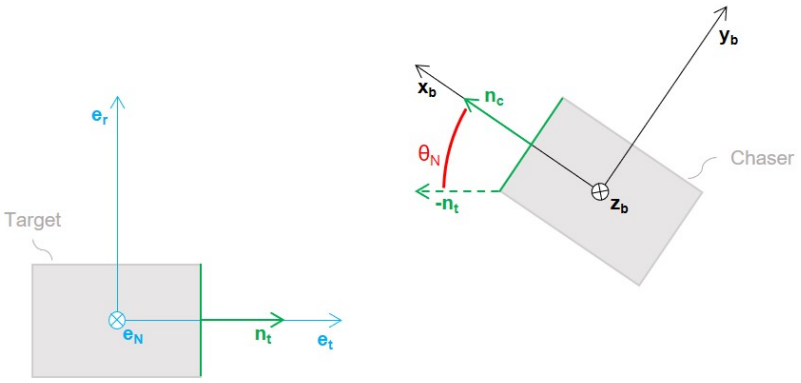


Figure 4.3: Convention of the attitude angle θ_N (red) in the Hill's frame (blue). Both spacecraft are characterised by their docking port and their normal vector $\mathbf{n_j}$ (green). The representation displays a generic Target with a docking port normal to $\mathbf{e_t}$.

The Chaser rotational equation of motion is then derived.

$$I_{zz}\ddot{\theta}_N = -D\dot{\psi} \tag{4.6}$$

Where $\dot{\psi}$ is the commanded acceleration given to the reaction wheel to produce a required torque.
Finally, the analytical solution of equation 4.6 can easily be found.

$$\begin{aligned} \dot{\theta}_N &= -\frac{D}{I_{zz}}\dot{\psi}\,t + \dot{\theta}_{N\,0} \\ \theta_N &= \dot{\theta}_N\,t + \theta_{N\,0} \end{aligned} \tag{4.7}$$

4.4 Constraints

To assure the safety and feasibility of the trajectory, a set of constraints is implemented. The following constraints are strongly inspired by a challenge published in 2021 aiming to find trajectories for safe proximity operations. [40]

The first constraints that must be respected are those about the physical limitations of the actuators. In the case where one of these constraints is violated, the solution would not be feasible anymore.

Constraint 1	*Asymmetric bounded thrust*
	$F \in [F_{min}; F_{max}]$
Constraint 2	*Maximum reaction wheel velocity*
	$\lvert\psi\rvert \leq \psi_{max}$
Constraint 3	*Maximum reaction wheel acceleration*
	$\lvert\dot{\psi}\rvert \leq \dot{\psi}_{max}$

Constraint 1 implies that the Chaser should obey reasonable thrust limitations. In some cases, thrust capabilities may not be equal in all directions or may be limited in some

directions. As such, specific thrust limits are assigned for each direction.
Constraint 2 and *constraint 3* consider the reaction wheel physical limitations and actuator longevity. Indeed, to improve longevity one might prefer to stay below a given velocity to reduce the overall wear. In addition, excessive wear leading to premature failures can also be due to repeated and extended operation of reaction wheels at the extreme ends of its feasible acceleration.

Additional constraints are introduced to enforce the relative velocity of the Chaser to respect some safety considerations.

Constraint 4 *Recoverable relative velocity limit*
$|\dot{x}| \leq v_{x\,max}$ and $|\dot{y}| \leq v_{y\,max}$
Constraint 5 *Bounded relative velocity limit*
$\|\mathbf{v}\| \leq v_{dock} + f_s \|\mathbf{r}\|/T_c$

Constraint 4 translates the idea that spacecraft manoeuvres shall maintain recoverable relative motion. Indeed, it is possible for the relative velocity to be so high that it exceeds the capability of the actuators to arrest motion within a limited time frame. In this work, v_{max} is considered as the maximum velocity that the spacecraft can travel in the x or y direction and stop within one minute given the available thrust.

$$v_{x\,max} = v_{y\,max} = v_{max} = \frac{F_{max}}{m} t_{stop} \tag{4.8}$$

Constraint 5 enhances the fact that the Chaser spacecraft should not be travelling exceedingly fast relatively to its distance from the Target. In the cases where spacecraft formation flying or docking is intended, the magnitude of the acceptable relative velocity decreases as the relative distance decreases. This concept is based on the idea of time-to-collision T_c as a function of the available thrust, spacecraft mass, and distance from the Target.

$$T_c = \sqrt{\frac{2m}{F_{max}} \|\mathbf{r}\|} \tag{4.9}$$

The relative velocity limit of the Chaser is then defined by considering a safety factor f_s and the maximum final velocity v_{dock} allowed to ensure a safe docking. From the mission statement: $v_{dock} = 0.2\,m\,s^{-1}$.

The same considerations are made on the attitude of the Chaser and translated through the following constraints.

Constraint 6 *Maximum angular velocity*
$|\dot{\theta}_N| \leq \dot{\theta}_{N\,max}$
Constraint 7 *Maximum angular acceleration*
$|\ddot{\theta}_N| \leq \ddot{\theta}_{N\,max}$

Constraint 6 is in place for the same reason as the translational velocity limits: to enable the Chaser to react or recover from commands in a reasonable time frame.
On the other hand, *constraint 7* deals with the structural integrity of the Chaser because excessive rotational acceleration may cause damage to the spacecraft structure, payload (which may be a sensitive instrument), or one of many possible deployable appendages.

A final constraint is implemented to oblige the Chaser to enter and remain in the line-of-sight of the Target docking port sensors during the entire docking phase. The line-of-sight is, in a planar problem, nothing more than the section of disk characterised by the semi-angle α_{LoS} with respect to the outward normal vector of the Target docking port $\mathbf{n_t}$.

Constraint 8 *Docking cone*
$\alpha_c \leq \alpha_{LoS}$

A summary of the safety and physical constraints formalization is presented in table 4.2. Note that while numeric values are given, the techniques developed can be applied for other similar constraints.

ID	Description	Formalization	Value
1	Asymmetric bounded thrust	$F \in [F_{min}; F_{max}]$	$F_{min} = -1\ N$ $F_{max} = 2\ N$
2	Maximum RW velocity	$\lvert\psi\rvert \leq \psi_{max}$	$\psi_{max} = 576.0\ rad\, s^{-1}$
3	Maximum RW acceleration	$\lvert\dot{\psi}\rvert \leq \dot{\psi}_{max}$	$\dot{\psi}_{max} = 181.3\ rad\, s^{-2}$
4	Recoverable relative velocity limit	$\lvert\dot{x}\rvert \leq v_{max}$ and $\lvert\dot{y}\rvert \leq v_{max}$	$v_{max} = 10\ m\, s^{-1}$
5	Bounded relative velocity limit	$\lVert\mathbf{v}\rVert \leq v_{dock} + f_s \sqrt{\frac{F_{max}}{2m}\lVert\mathbf{r}\rVert}$	$v_{dock} = 0.2\ m\, s^{-1}$ $f_s = 1$
6	Maximum angular velocity	$\lvert\dot{\theta}_N\rvert \leq \dot{\theta}_{N\,max}$	$\dot{\theta}_{N\,max} = 2\ deg\, s^{-1}$
7	Maximum angular acceleration	$\lvert\ddot{\theta}_N\rvert \leq \ddot{\theta}_{N\,max}$	$\ddot{\theta}_{N\,max} = 1\ deg\, s^{-2}$
8	Docking cone	$\alpha_c \leq \alpha_{LoS}$	$\alpha_{LoS} = 45\ deg$

Table 4.2: Constraints of the problem with their respective numerical values

Chapter 5

Reinforcement Learning Implementation

Chapters 2 and 3 explained the algorithm and the mathematical tools needed to understand the reinforcement learning concept. These two chapters are generic and explore the different concepts without any specification on the problem to solve. Then, chapter 4 introduced the problem to solve: a safe three degrees of freedom autonomous rendezvous, proximity operation, and docking. While the previous chapters can be read and understood independently of each other, this chapter is built on all the concepts and ideas seen so far. As such, it focuses on the implementation of the reinforcement learning framework to solve the problem of this work.

First, the environment in which the learning Agent evolves is defined within the Markov decision process framework. At this point, the state space and the action space are formalized. Then, the discrete dynamics respecting the different modelisation choices is formalized. Concerning the reward function, a proper section describes the logic used to teach an Agent. Finally, the last section is dedicated to the Agent's hyperparameters introduced throughout the three previous chapters. Note that the implemented reward logic and set of hyperparameters are specified in chapter 6.

5.1 ARPOD as a Markov Decision Process

For the sake of clarity, the definition and notations of a Markov decision process presented in section 2.2 are briefly recalled below.

A Markov decision process is the underlying structure for a decision-making algorithm providing a discrete-time mathematical framework. At each time step $t = 0, 1, 2, ...$, the Agent is in a state $S_t \in \mathscr{S}$, and it chooses an action $A_t \in \mathscr{A}(S_t)$. The environment responds at the next time step by moving into a subsequent state $S_{t+1} \in \mathscr{S}$, and by giving the Agent a corresponding reward $R_{t+1} \in \mathscr{R}$. This process goes on until an end condition is met.

It follows that the different spaces have to be formalized to solve the specific problem of this work. While the state space $\mathscr{S}$ and the action space $\mathscr{A}$ are described in this section, the reward function, or reward logic, is presented in section 5.2.

5.1.1 State Space

To satisfy the requirements of the Markov property, the Agent's state must provide sufficient information such that the transition to the following state is only a function of the present state and the action. On the other hand, while unneeded information can be added into the state without preventing the learning process, it slows it down considerably.

The Chaser dynamics introduced in equations 4.5 and 4.7 can be fully defined by its position, velocity, attitude and angular velocity: $x, y, \dot{x}, \dot{y}, \theta_N, \dot{\theta}_N$. These accordingly make up the six variables in the state.

$$S_t = \left[x_t,\ y_t,\ \dot{x}_t,\ \dot{y}_t,\ \theta_{N\,t},\ \dot{\theta}_{N\,t}\right] \tag{5.1}$$

In addition, a common trick to enhance the backpropagation process of artificial neural networks is to scale the inputs so that their covariance is about the same and each input variable has the same importance [41]. Therefore, six new variables taking their values between -1 and 1 are built.

$$\widetilde{x} = \frac{x}{r_0}; \quad \widetilde{y} = \frac{y}{r_0}; \quad \widetilde{\dot{x}} = \frac{\dot{x}}{v_{max}}; \quad \widetilde{\dot{y}} = \frac{\dot{y}}{v_{max}}; \quad \widetilde{\theta_N} = \frac{\theta_N}{\pi}; \quad \widetilde{\dot{\theta}_N} = \frac{\dot{\theta}_N}{\dot{\theta}_{N\,max}}$$

Where r_0 is the initial distance of the Chaser with respect to the Target, and θ_N is the Chaser attitude wraps in $[-\pi;\ \pi]$.
Consequently, for each time step t, the Chaser state S_t used as input for the artificial neural networks is denoted $\widetilde{S}_t$ and defined as follows.

$$\widetilde{S}_t = \left[\widetilde{x}_t,\ \widetilde{y}_t,\ \widetilde{\dot{x}}_t,\ \widetilde{\dot{y}}_t,\ \widetilde{\theta_N}_t,\ \widetilde{\dot{\theta}_N}_t\right] \tag{5.2}$$

5.1.2 Action Space

While the state is the input of both the Actor and the Critic artificial neural networks exploited by the Agent during the learning process, the output of the Actor network is the action vector.
In this work, the actions of the Chaser are a velocity impulse Δv from the thruster and a commanded acceleration $\dot{\psi}$ given to the reaction wheel. In addition, due to the discrete-time model of a Markov decision process, a further assumption is made.

Assumption 6 *Actuator actions are instantaneous impulses occurring at the beginning of each time step.*

Until now, the action vector was introduced as the output of the Actor artificial neural network. This vision of the action vector was used to provide a simple explanation but, at the same time, it shortcuts important details that are explained in this section.
The final layer of the Actor network uses a *tanh* activation function so that the mean of each dimension of the distribution is between -1 and 1. However the standard deviation of the distribution allows for values to be outside of this range, so the outputs are clipped

element-wise back within this range if necessary. Let a tilde denote the real Actor artificial neural network outputs $(\widetilde{\Delta v},\ \widetilde{\dot{\psi}}) \in [-1;\ 1]^2$. Consequently, at each time step t, the output of the Actor network is defined as follows.

$$\widetilde{A}_t = \left[\widetilde{\Delta v}_t,\ \widetilde{\dot{\psi}}_t\right] \tag{5.3}$$

The goal is then to decrease the size of the action space $\mathscr{A}$ to ease the learning process. Even if in theory, having an infinite action space and its proper reward logic is a feasible strategy, in practice, it increases the complexity of the problem to such an extent that the computing time becomes unfeasible. Consequently, the action space must be large enough to include all the feasible actions, but also as small as possible to ease the learning process.
Due to the time dependency of the problem, at each time step t the restricted sub-action space $\mathscr{A}_t \subset \mathscr{A}$ is defined from the state S_t. This sub-action space is divided into two more sub-spaces: the reaction wheel acceleration space, and the thruster velocity impulse space.

Before restricting these two action spaces, a few words must be devoted to the modelisation of the actions. At each time step t, first the instantaneous reaction wheel action $\dot{\psi}_t$ changes the attitude of the Chaser to $\theta_{N\,t+1}$, then with this new orientation, the thruster creates an instantaneous velocity impulse Δv_t that leads the Chaser to its subsequent state S_{t+1}. With this sequence of actions in mind, it makes sense to first consider the reaction wheel action space, and then the thruster action space.

Reaction Wheel Acceleration Space

The reaction wheel is the only actuator impacting the attitude of the Chaser and therefore its action space is directly related to the constraints on both the attitude and the actuator limitation. For the sake of clarity, the different constraints directly impacting the attitude of the Chaser are recalled below.

ID	Description	Formalization
2	Maximum RW velocity	$\lvert\psi\rvert \leq \psi_{max}$
3	Maximum RW acceleration	$\lvert\dot{\psi}\rvert \leq \dot{\psi}_{max}$
6	Maximum angular velocity	$\lvert\dot{\theta}_N\rvert \leq \dot{\theta}_{N\,max}$
7	Maximum angular acceleration	$\lvert\ddot{\theta}_N\rvert \leq \ddot{\theta}_{N\,max}$

Table 5.1: Constraints on the attitude and the reaction wheel of the Chaser

By using the attitude dynamics described in equation 4.6 and its analytical solutions in equations 4.7, one can easily determine the direct impact of these constraints on the commended acceleration. Note that the discrete-time notation is used to stay coherent with the Markov properties.

ID	Description	Formalization
2	Maximum RW velocity	$\lvert\dot{\psi}_t\rvert \leq \frac{\dot{\psi}_{max}-\dot{\psi}_t}{t_{step}}$
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 5.2: Constraints on the attitude of the Chaser and their impact on the commended reaction wheel acceleration

where t_{step} refers to the length of the time step t. Consequently, the reaction wheel action space at a given time step can be restricted by knowing the Chaser's state and the reaction wheel velocity. Hereafter, the boundaries of the reaction wheel action space at time step t are denoted by $\dot{\psi}_t^{SUP}$ and $\dot{\psi}_t^{INF}$.

Finally, the transition between the clipped output of the Actor network $\widetilde{\dot{\psi}}_t$ and the reaction wheel acceleration $\dot{\psi}_t$ is formalized.

$$\dot{\psi}_t = \widetilde{\dot{\psi}}_t \frac{\dot{\psi}_t^{SUP} - \dot{\psi}_t^{INF}}{2} + \frac{\dot{\psi}_t^{SUP} + \dot{\psi}_t^{INF}}{2} \tag{5.4}$$

Thruster Velocity Impulse Space

The thrust action space is directly impacted by the constraints on the velocity of the Chaser and the actuator limit. For the sake of clarity, the related constraints are recalled below.

ID	Description	Formalization
1	Asymmetric bounded thrust	$F \in [F_{min}; F_{max}]$
4	Recoverable relative velocity limit	$\lvert\dot{x}\rvert \leq v_{max}$ and $\lvert\dot{y}\rvert \leq v_{max}$
5	Bounded relative velocity limit	$\lVert\mathbf{v}\rVert \leq v_{dock} + f_s\sqrt{\frac{F_{max}}{2m}\lVert\mathbf{r}\rVert}$

Table 5.3: Constraints on the velocity and the thruster of the Chaser

First, the limit velocity must be determined. While the attitude dynamics is relatively straightforward, the translational dynamics is more complex. Indeed, the latter is described by the planar Clohessy-Wiltshire (CW) equations 4.4, which couple the velocity and the position. To ease the following explanations, the analytical solution presented in equations 4.5 is written with a matrix notation [39].

$$\begin{aligned} \mathbf{r}_{t+1} &= [\mathbf{\Phi}_{rr}(t_{step})]\,\mathbf{r}_t + [\mathbf{\Phi}_{rv}(t_{step})]\,\mathbf{v}_t \quad (a) \\ \mathbf{v}_{t+1} &= [\mathbf{\Phi}_{vr}(t_{step})]\,\mathbf{r}_t + [\mathbf{\Phi}_{vv}(t_{step})]\,\mathbf{v}_t \quad (b) \end{aligned} \tag{5.5}$$

To respect *constraints 5* and *4* at each time step t, it is needed to ensure that both the initial velocity $\lVert\mathbf{v}_t + \Delta\mathbf{v}_t\rVert$ and the subsequent velocity $\lVert\mathbf{v}_{t+1}\rVert$ are lower than the bounded

and recoverable relative velocity limits at their respective time step. In this work, the limit imposed on the components of the velocity by *constraint 4* is enforced by considering the stricter constraint : $\|\mathbf{v}\| \leq v_{max}$.
In the case of the initial velocity, a temporary velocity limit $v_t^{LIMtemp}$ is directly computed from *constraints 4* and *5*.

$$v_t^{LIMtemp} = \min\left(v_{max},\, v_{dock} + f_s\sqrt{\frac{F_{max}}{2m}\|\mathbf{r}_t\|}\right) \tag{5.6}$$

In addition, *constraint 5* is a function of the relative distance. Therefore the final velocity limit v_t^{LIM} must also ensure that the velocity $\mathbf{v}_{t+1}$ respects the constraints. To do so, it is worth knowing the smallest relative distance r_{t+1} that could be reached starting from $\mathbf{r}_t$ with an attitude $\theta_{N\,t+1}$. Since the velocity impulse Δv_t is still unknown, the CW equation 5.5.a is applied with a guessed velocity. Hence, by considering $v_t \in \{-v_t^{LIMtemp};\, 0;\, v_t^{LIMtemp}\}$, an approximation of the smallest reachable distance r_{t+1}^{MIN} is found. Consequently, the smallest bounded relative velocity limit v_{t+1}^{MAX} is known.

$$v_{t+1}^{MAX} = v_{dock} + f_s\sqrt{\frac{F_{max}}{2m}r_{t+1}^{MIN}} \tag{5.7}$$

Then, the velocity that must not be overcome at time step t to respect the constraints at time step $t+1$ is computed by applying the CW equation 5.5.b with $\mathbf{v}_{t+1}^{MAX}$ and $\mathbf{r}_t$. This velocity is denoted v_t^{MAX} in the following equation.
Finally, the final velocity limit at time step t is derived.

$$v_t^{LIM} = \min\left(v_t^{LIMtemp},\, v_t^{MAX}\right) \tag{5.8}$$

To sum up, at this point, the velocity that must not be overcome to respect *constraints 4* and *5* is known. Now, the thruster velocity impulse space can be restricted by considering the full set of constraints of table 5.3. Note that the bounded thrust constraint can be translated into a velocity impulse constraint by equation 5.9.

$$\Delta v_t = \frac{F_t}{m} * t_{step} \tag{5.9}$$

where t_{step} is the length of time step t. In that case, the formalization of *constraint 1* can be rewritten: $\Delta v_t \in [\Delta v_{t\,min}; \Delta v_{t\,max}]$.

Finally, the boundaries of the thruster velocity impulse space at a given time step can be derived.

$$\begin{aligned} \Delta v_t^{SUP} &= \min\left(\Delta v_{t\,max},\, v_t^{LIM} - v_t\right) \\ \Delta v_t^{INF} &= \max\left(\Delta v_{t\,min},\, -v_t^{LIM} - v_t\right) \end{aligned} \tag{5.10}$$

Then the transition between the clipped output of the Actor network $\widetilde{\Delta v_t}$ and the thruster velocity impulse Δv_t is formalized.

$$\Delta v_t = \widetilde{\Delta v_t} \frac{\Delta v_t^{SUP} - \Delta v_t^{INF}}{2} + \frac{\Delta v_t^{SUP} + \Delta v_t^{INF}}{2} \tag{5.11}$$

Note that the constraint on the actuator limit and the constraints on the Chaser state have been handled separately, and in some cases, both limits cannot be respected simultaneously. This case is faced when the impulse constrained by the upper velocity limit v_t^{LIM} is lower than action space lower bond Δv_t^{INF} or the other way round. In other words, this case happens when the velocity impulse needed by the Chaser to remain in the acceptable velocity range overcomes the physical capacity of its thruster.
When this situation is faced, the thruster velocity impulse is limited to its physical limitation at the expense of the relative velocity constraints.

5.1.3 Discrete Dynamics

The set of equations describing the dynamics of the problem were introduced in section 4.3 in their continuous form. This section aims to provide the reader with a clear description of the discrete dynamics of the Chaser within the chosen modelisation of the problem. The discrete dynamic equations are defined recursively, which means that the state at any arbitrary time step is computed from the state at the previous time step. This recursive behaviour highlights the complexity behind this problem: the final state is the result of a sequence of actions. Therefore, when the Agent has to choose an action at the beginning of the manoeuvre, it must take into account its consequences not only in the next time step but after a sequence of steps. This choice-making mechanism is formalized in chapter 2 and no additional words will be spent on it. This section dives into the mathematical formalisation of this recursive problem.

The following description assumes that the state S_t at time step t is known and computes the subsequent state S_{t+1} where the initial state S_0 is defined by the user. The description of the state at any arbitrary time step is as follows.

$$S_t = [x_t, y_t, \dot{x}_t, \dot{y}_t, \theta_{N\,t}, \dot{\theta}_{N\,t}]$$

First, the subsequent attitude of the Chaser is computed from the previous one and the instantaneous acceleration of the reaction wheel $\dot{\psi}_t$.

$$\begin{aligned} \dot{\theta}_{N\,t+1} &= -\frac{D}{I_{zz}} \dot{\psi}_t \, t_{step} + \dot{\theta}_{N\,t} \\ \theta_{N\,t+1} &= \dot{\theta}_{N\,t+1} \, t_{step} + \theta_{N\,t} \end{aligned} \tag{5.12}$$

Then, the instantaneous velocity impulse of the thruster Δv_t is considered.

$$\begin{aligned} v_{x\,t} &= \dot{x}_t + \sin(\theta_{N\,t+1}) \, \Delta v_t \\ v_{y\,t} &= \dot{y}_t - \cos(\theta_{N\,t+1}) \, \Delta v_t \end{aligned} \tag{5.13}$$

And finally, the remaining terms of the subsequent state are computed with the analytical

solution of the CW equations.

$$
\begin{aligned}
x_{t+1} &= [4 - 3\cos(nt_{step})]\, x_t + \frac{\sin(nt_{step})}{n}\, v_{x\,t} + \frac{2}{n}[1 - \cos(nt_{step})]\, v_{y\,t} \\
y_{t+1} &= 6[\sin(nt_{step}) - nt_{step}]\, x_t + y_t + \frac{2}{n}[\cos(nt_{step}) - 1]\, v_{x\,t} + \frac{1}{n}[4\sin(nt_{step}) - 3nt_{step}]\, v_{y\,t} \\
\dot{x}_{t+1} &= 3n\sin(nt_{step})\, x_t + \cos(nt_{step})\, v_{x\,t} + 2\sin(nt_{step})\, v_{y\,t} \\
\dot{y}_{t+1} &= 6n[\cos(nt_{step}) - 1]\, x_t - 2\sin(nt_{step})\, v_{x\,t} + [4\cos(nt_{step}) - 3]\, v_{y\,t}
\end{aligned}
\tag{5.14}
$$

This recurrence relation continues until an end condition is met. Hence, the episode ends if the Chaser reaches the Target (i.e. $r \leq 1\ m$), if a terminal condition is met, or simply if the limit on the number of time steps is reached. Note that more details about terminal conditions are given during the reward logic design.

One may have noticed the strong importance of the time step length t_{step} on the dynamic equations and the action spaces. Decreasing the time length implies a higher control frequency leading to a more precise control of the Chaser. However, it increases the sequence of actions needed to reach the Target and so the complexity of the problem. Consequently, when the Chaser gets closer to the Target, precise control is preferable. Hence, the length of the time step should decrease. A solution to realise it is to add the time length into the Actor artificial neural network outputs. By doing so, the Agent has the possibility to adjust it on its own according to its needs. However, it increases the complexity of the learning process and therefore the computation time.
For this work, the will to keep the computation time as small as possible leads to a decrease in the time length according to the logic presented in table 5.4.

Condition	**length of time step** t
$r_t > 1000\ m$	$t_{step} = 10\ s$
$1000\ m \geq r_t > 100\ m$	$t_{step} = 5\ s$
$100\ m \geq r_t > 10\ m$	$t_{step} = 2\ s$
$10\ m \geq r_t > 5\ m$	$t_{step} = 1\ s$
$5\ m \geq r_t$	$t_{step} = 0.5\ s$

Table 5.4: Switching logic of the time step length according to the relative distance

5.2 Reward logic

Central to reinforcement learning is the idea of a reward function, which indicates to the learning Agent what states are preferred, and what states should be avoided. To make reinforcement learning algorithms run in a reasonable amount of time, it is necessary to use a well-chosen reward function that gives appropriate indications to the Agent. A faulty reward logic or a too complex one often changes the problem in an unanticipated way that leads to poor solutions.
The most common idea is to use so-called shaping rewards. These rewards are based on potential functions over the state and give the Agent hints at each time step on how well

it is performing [42]. These functions commonly take the form of linear and exponential ones.
On the other hand, sparse rewards do not lead to the desired solution. It is explained by the fact that, if the Agent does not receive any reward, then it does not know how to update its parameters. So it continues to take random actions with the current set of parameters until it gets nonzero rewards. The sequence of actions that resulted in the reward might be very long, and it is not clear which of those actions were useful in getting the reward. This problem is known as credit assignment in reinforcement learning [43].
However, sparse rewards are used in addition to shaping rewards. As such, the rewards do not suffer from the credit assignment problem since shaping rewards tackle it. In this case, sparse rewards are used to give the Agent strong positive or negative indications when some user-defined conditions are met.

Consequently, the reward logic can be divided into two categories: shaping rewards $R_t^{shaping}$, and sparse rewards R_t^{sparse}.

$$R_t = R_t^{shaping} + R_t^{sparse}$$

When designing these two functions, one must be careful not to create local maxima that might teach the Agent a wrong solution. Hence, the general idea used in this work is to give positive rewards for "good" actions and negative rewards otherwise.

5.3 The Agent's Hyperparameters

So far in this chapter, only the environment in which the learning Agent interacts has been designed. Unlike the Agent, the environment reflects the dynamics and the constraints of the problem. As such, each environment is unique and must be specifically designed for every problem. On the other hand, the Agent is built on a generic algorithm that could be used in many different problems with just a minimum of changes. Nevertheless, Agent behaviour relies on parameters that have a major impact on the learning process. These parameters are commonly called hyperparameters to differentiate them from the learned parameters of artificial neural networks (i.e. weights and biases).
The hyperparameters of the Proximal Policy Optimisation algorithm (PPO) were introduced throughout chapters 2 and 3. For the sake of clarity, they are recalled with their meanings.

The PPO algorithm introduced in chapter 2 is a policy gradient method. Policy gradient algorithms have two steps: first, transitions are gathered, and then the policy is improved. In other words, policy gradient methods gather sequences of states, rewards, and actions called transitions, and use them to update the policy. Then, the old transitions are discarded and new transitions are gathered using the new policy. This leads to the first grouping of hyperparameters that deal with experience collection:

horizon The algorithm gathers trajectories as far out as the horizon limits before gradient descent is performed. Typically a longer horizon corresponds to more stable training updates.

mini-batches The mini-batch size corresponds to how many transitions are used for each gradient descent update. This should always be a fraction of the horizon.

epoch The number of epoch is the number of passes through the transition buffer during gradient descent. The larger the mini-batch size, the larger it is acceptable to do this. Decreasing the number of epoch ensures more stable updates, at the cost of slower learning.

The second set of hyperparameters deals with how the old policy is updated to the new policy. As seen in section 2.3.2, an advantage function translates how far out should rewards in the future influence the policy. The implemented algorithm uses the Generalized Advantage Estimation (section 2.3.3) to alter the reward stream with two parameters: γ and λ. these two parameters perform a bias-variance trade-off of the trajectories and can also be viewed as a form of reward shaping.

γ It is known as the discounting factor. It controls the greediness of the Agent: the smaller γ is, the more immediate rewards are prioritized over long term rewards.

λ It can be seen as a parameter performing a time average of the advantage function. Decreasing it induces bias when the value function is inaccurate, but also it reduces the variance. As a rule of thumb, the best value of λ is lower than the best value of γ.

In addition, to avoid too large updates of the policy that could make the learning process collapse, the PPO algorithm uses a surrogate loss function. This work uses the clipped loss version explained in section 2.4.2 to keep the step from the old policy to the new policy within a safe range.

ϵ It corresponds to the acceptable threshold of divergence between the old and new policies during a gradient update. Setting this value low results in more stable updates, but it also slows the learning process.

Finally, the surrogate objective function formalized in equation 2.30 includes an entropy term. It works as a regularizer: a policy has maximum entropy when all policies are equally likely and minimum when one action probability of the policy is dominant.

entropy coefficient It acts on the exploitation/exploration dilemma by preventing premature convergence of one action probability. It commonly decreases during training to ensure exploitation at the end of the process.

The last set of hyperparameters, introduced in chapter 3, refers to the two different artificial neural networks used by the Agent. Both networks are characterised by their

structure and their learning rate. Note that the latter is closely related to the learning process through the gradient descent algorithm.

number of layers	It corresponds to how many hidden layers are present between the input layer and before the output layer. While fewer layers are likely to train faster and more efficiently. More layers may be necessary for complex control problems.
hidden neurons	It represents how many units are in each fully connected layer of the neural network. For problems where the action is a complex interaction between the state variables, the number of hidden neurons should be large.
learning rate	It is the strength of each gradient descent update step. This should typically be decreased if training is unstable, and the reward does not consistently increase.

Together with the reward function, the hyperparameters need to be tuned to achieve a desired level of performance. While some rules of thumb are given above to tune them, it is not enough to reach values leading to correct behaviour. Therefore, most of the tuning comes from a long series of trial and error.

Chapter 6

Test Cases

Finally, this chapter aims to study test cases based on the reinforcement learning implementation of an ARPOD problem presented in chapter 5. While the state space and the action space are clearly defined in the previous chapter, the specific reward logic and set of hyperparameters are specified for each test case. This structure emphasizes the close relationship between a given problem and the design of the reward function and hyperparameters.

The full description of the problem was presented throughout chapter 3. However, a few words are worth being devoted to recall the main objectives of the problem.
The algorithm has to compute a trajectory allowing the Chaser to dock with the Target by respecting a set of constraints modelling the limitation of the actuators and the safety of the manoeuvre. This trajectory is made of three phases. First the rendezvous phase between 10 km and 1 km where the Chaser has to get closer to the Target. Then, the proximity operation phase between 1 km and 100 m in which the Chaser still has to go closer but must also reach the docking cone by the end of the phase. Finally, the docking phase where the Chaser has to safely dock with the Target while remaining inside the docking cone.

The first test case with an initial relative distance of 1 km is analyzed. This case focuses on the proximity operation and the docking phase to have a close look at the ability of the algorithm to solve highly constrained problems. A second case studies the behaviour of the algorithm when solving a full ARPOD problem.

The following results were produced using Python and Pytorch on a HP Pavilion 15 with an Intel Core i7 CPU, a NVIDIA GeForce 840M, and 6 GB of RAM.

6.1 Case 1: Proximity Operation and Docking

This case focuses on the ability of reinforcement learning algorithms to solve highly constrained problems. Hence, by restricting the problem to the proximity operation and the docking phases, the Agent has to solve the problem with the full set of constraints presented in section 4.4. In addition, by skipping the rendezvous phase, the length of the trajectory and therefore the length of the sequence of actions leading to docking is kept to a minimum.

To do so, the initial relative distance is fixed at 1 km, the Chaser is oriented toward the Target, and both the relative translation and angular velocity are null. In addition, two initial configurations are studied:

- The so-called V-bar configuration where the Chaser initial position is along the tangential direction on the Hill's frame.

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

- And the R-bar configuration where the Chaser initial position is along the radial direction on the Hill's frame.

$$S_0 = [1000,\ 0,\ 0,\ 0,\ -\frac{\pi}{2},\ 0]$$

In both configurations, the Target does not rotate and has its docking port outward normal aligned with the tangential direction $\mathbf{e}_t$.

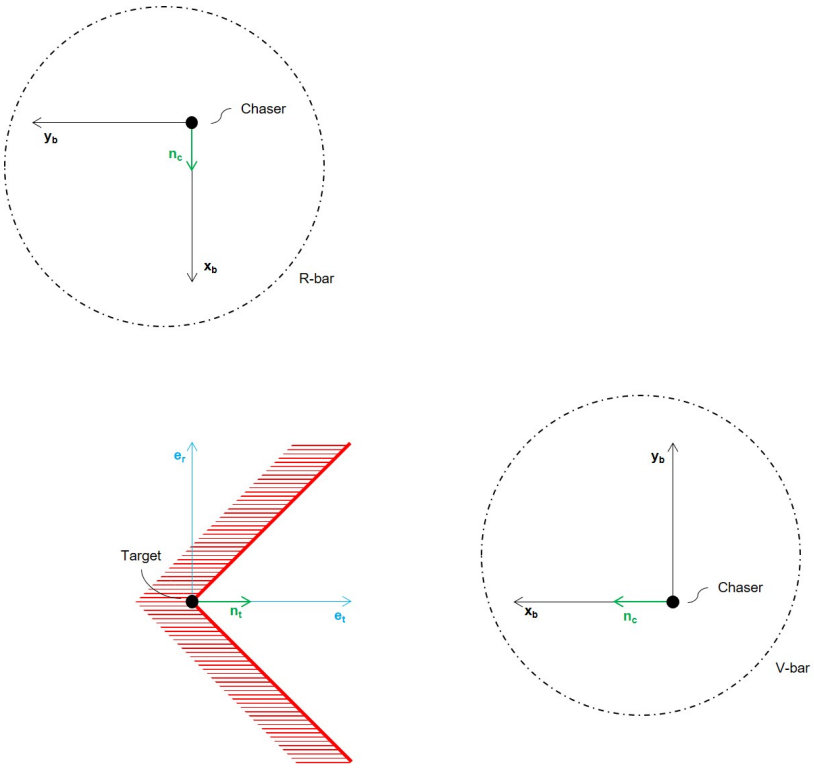


Figure 6.1: Unscaled representation of the R-bar and V-bar initial configurations with the Target docking port outward normal aligned with the tangential direction (i.e. $\mathbf{n}_t \equiv \mathbf{e}_t$). The docking cone is displayed in red.

6.1.1 Reward Logic

During the design process of the sparse and shaping reward functions, a general idea is respected to avoid creating local minima. Hence, the following reward function gives positive rewards for "good" actions and negative rewards otherwise.

Shaping Rewards

The overall goal of the Chaser is to reach the Target. Therefore, the first shaping reward function is used to teach the Agent to decrease the relative distance separating the two spacecraft. To do so, a function that strictly increases when the relative distance decreases is needed. To respect the general design idea, the function also has to be negative when the relative distance is bigger than the initial one. Hence, the function used is as follows. Note that an exponential term is added to enhance the final docking manoeuvre.

$$R_t^{distance} = 10 + 10\,\frac{r_t}{r_0} + 5\,e^{1-0.01 r_t} \tag{6.1}$$

Once the Chaser enters the docking phase (i.e. $r_t \leq 100\ m$) it must control its attitude to dock with a relative angle lower than 5 degrees. The function is then designed by following the same strategy as previously. The only difference is that, in this case, the function must be positive when the absolute attitude angle is lower than 5 degrees. However, it is important that this function does not overcome the previous one otherwise the Agent might try to have a null attitude angle at the expense of reaching the Target.

$$R_t^{attitude} = \begin{cases} 10\,\frac{5-|\theta_{N\,t}|}{180} & if\ \ r_t \leq 100\,m \\ 0 & otherwise \end{cases} \tag{6.2}$$

where $\theta_{N\,t}$ is the attitude angle in degrees clipped between -180 and 180.

Finally, the shaping reward function is then the sum of the two previous functions.

$$R_t^{shaping} = R_t^{distance} + R_t^{attitude}$$

Sparse Rewards

Sparse rewards are used for two distinct purposes: to increase the stability with positive reward named bonus, and to avoid some situations with negative reward named penalty.

To begin with, the bonus logic is explained. Bonuses are positive values used to reward the Agent for reaching some milestones. Hence, it helps the Agent to understand its environment and therefore to solve the problem. In this implementation, a bonus is given for the following reasons.

- When the Chaser is in the docking phase.

$$r_t \leq 100\,m$$

- When the Chaser is in the docking range. This condition ends the episode.

$$r_t \leq 5\,m$$

- When the Chaser is in the docking range with the acceptable attitude angle.

$$r_t \leq 5\,m \ \text{ and } \ |\theta_{N\,t}| \leq 5\,deg$$

Furthermore, the Agent may also receive penalties when it is in some user-defined situation. Concretely, penalties take the form of negative rewards that deter the Agent from exploring a given part of the state space. In this test case, penalties are given for the following reasons.

- When the relative distance increases to dissuade the Chaser from moving away from the Target.
$$r_t > r_{t-1}$$
- When the Chaser explores the region farther than the initial relative distance. A margin of $10\,m$ is added to help the Agent to find the correct direction to follow.
$$r_t > r_0 + 10\,m$$
This is a terminal condition for the episode since there is no use in exploring this region. No other penalty conditions are terminal in order to enable the Agent to learn even if it is in "bad" situations.
- When the Chaser does not respect the velocity constraints *4* and *5*. Indeed, as explained in section 5.1.2, despite the reduction of the thruster action space, it may happen that the velocity impulse needed by the Chaser to remain in the acceptable velocity range overcomes the physical capacity of its thruster.
$$v_t > v_{max} \quad \text{or} \quad v_t > v_{dock} + f_s \sqrt{\frac{F_{max}}{2m} r_t}$$
- When the Chaser is in the proximity operation phase (i.e. $r_t \leq 1000\,m$) and the Agent explores a region outside of the docking cone. This point aims to teach the Agent to respect *constraint 8*.
$$r_t \leq 1000\,m \quad \text{and} \quad (x_t - y_t > 1 \quad \text{or} \quad -x_t - y_t > 1)$$
To enhance this point, an extra penalty is also given if the Chaser misses the Target.
$$r_t \leq 1000\,m \quad \text{and} \quad y_t < -1$$

Table 6.1 summarizes and quantifies the bonus and penalty logic in the same order of their description.

Condition	**Sparse reward**
$r_t \leq \ 100\,m$	$R_t^{sparse} = R_t^{sparse} + 10$
$r_t \leq \quad 5\,m$	$R_t^{sparse} = R_t^{sparse} + 500$
$r_t \leq 5\,m$ and $\lvert\theta_{N\,t}\rvert \leq 5\,deg$	$R_t^{sparse} = R_t^{sparse} + 500$
$r_t > r_{t-1}$	$R_t^{sparse} = -10$
$r_t > r_0 + 10\,m$	$R_t^{sparse} = -50$
$v_t > v_{max}$ or $v_t > v_{dock} + f_s \sqrt{\frac{F_{max}}{2m} r_t}$	$R_t^{sparse} = -10$
$r_t \leq 1000\,m$ and $(x_t - y_t > 1$ or $-x_t - y_t > 1)$	$R_t^{sparse} = 0$
$r_t \leq 1000\,m$ and $y_t < -1$	$R_t^{sparse} = -10$

Table 6.1: Case 1: Sparse reward logic for the attribution of bonuses and penalties

Unlike bonuses, penalties are not added to sparse rewards but replace them. This gives more importance to a penalty when the Chaser is in a closer phase. For instance, if the Chaser is in the docking phase and makes an action that increases the relative distance with the Target, then a sparse reward of -10 will replace one of 10. But if the same situation occurs in the rendezvous phase, then -10 will replace 0. Hence, the Agent receives hints telling it that violating this rule is worse when the Chaser is closer to the Target.

6.1.2 Hyperparameters

While some rules of thumb are given in section 5.3 to tune the hyperparameters, it is not enough to reach values leading to correct behaviour of the learning Agent. Therefore, most of the tuning comes from a long series of trial and error. The selected set of hyperparmeters is presented in the following table.

Name	**Value**
Horizon	64
mini-batch	16
epoch	3
γ	0.99
λ	0.65
ϵ	0.001 if $episode < 10000$ 0.0001 otherwise
entropy coefficient	0.001 if $episode < 10000$ 0.0005 if $10000 \leq episode < 20000$ 0.0001 if $20000 \leq episode < 25000$ 0 otherwise

Table 6.2: Case 1: Hyperparameters

The Actor and the Critic neural networks have the same structure with the only difference in the activation function of their output layer. This structure is inspired by the literature and formalized in tables 6.3 and 6.4. [22].

Layer	**Neurons**	**activation function**
First hidden	130	ReLU
Second hidden	90	ReLU
Third hidden	60	ReLU
Output	2	tanh

Table 6.3: Case 1: Actor artificial neural network structure

Layer	Neurons	activation function
First hidden	130	ReLU
Second hidden	90	ReLU
Third hidden	60	ReLU
Output	1	linear

Table 6.4: Case 1: Critic artificial neural network structure

To avoid too long simulations, the maximum number of time steps per episode is fixed at 128.

6.1.3 Results

This test case focuses on the ability of the Agent to solve the problem while respecting constraints. To do so, the two initial configurations displayed in figure 6.1 are analyzed. The V-bar initial configuration appears as the easiest one since the Chaser starts the manoeuvre within the extension of the docking cone. Hence, the Agent only has to learn how to dock by remaining inside the docking cone. On the other hand, with the R-bar initial configuration, the Agent must also learn how to reach the docking cone.

V-bar initial configuration

The best solution trajectory found by the Agent is displayed in figure 6.2 after training the Agent for 30,000 episodes, which took about 3 hours.
The final state of the trajectory attests that the Chaser docks with the Target.

$$\|\mathbf{r}_{final}\| = 0.89\,m \;\; and \;\; |\theta_{N\,final}| = 2.4\,deg$$

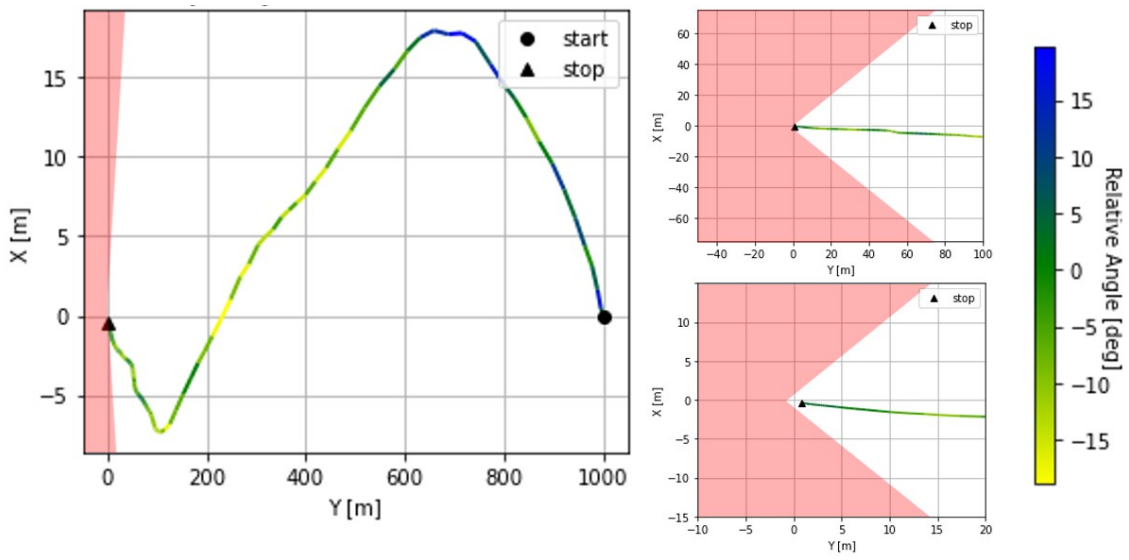


Figure 6.2: Case 1 - V-bar configuration: Full trajectory of the solution found (left), and trajectory zoomed on the Target (right). The docking cone is displayed in red.

It can be seen that the Chaser remains inside the docking cone during the entire docking phase. Therefore, the implemented reward function enables the Chaser to respect the docking cone constraint. In addition, the constraints enforced by the restriction of the action spaces are also respected. This result can be observed in figure 6.3 for the actuators physical limitation, and in figure 6.4 for the safety constraints.
Consequently, the solution found by the Agent represents a feasible trajectory leading to a safe docking of the Chaser with the Target.

One may remember that the time step length is decreased when the Chaser gets closer to the Target in order to have a more precise control. This can be observed in the following figures by noticing the increase in the variation frequency of the different quantities at the end of the manoeuvre. In addition, The variations in the velocity impulse limits are also due to the reduction of the time step length. Indeed, since the thrust limits are fixed, from equation 5.9 it can be understood that a decrease in the time step length implies a decrease in the velocity impulse limits.

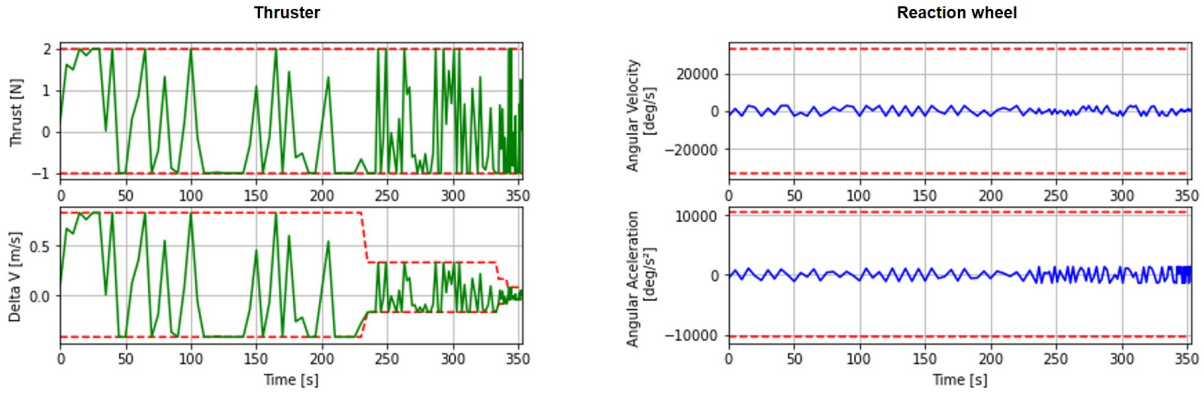


Figure 6.3: Case 1 - V-bar configuration: Behaviour of the Thruster in terms of thrust and commanded velocity impulse (left), and of the Reaction Wheel in terms of angular velocity and commanded angular acceleration (right).

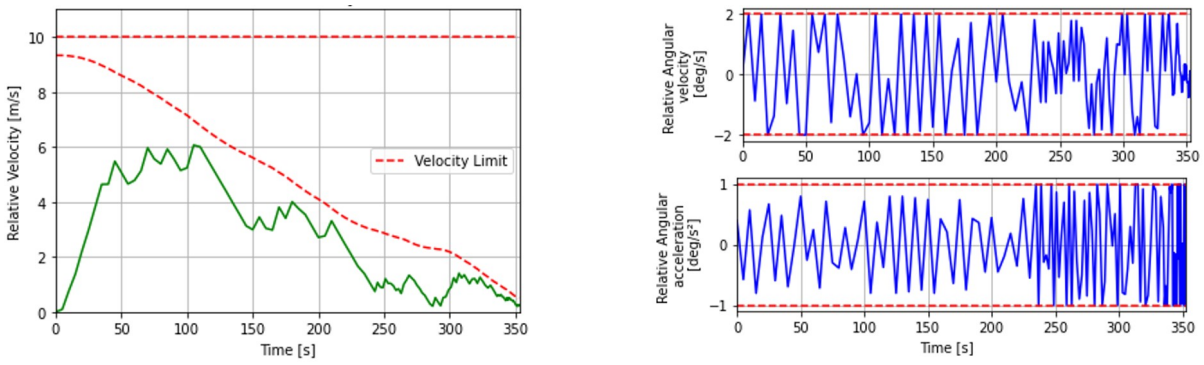


Figure 6.4: Case 1 - V-bar configuration: Relative velocity of the Chaser (left), and its angular velocity and acceleration (right).

Figure 6.4(left) displays two velocity limits: the fixed recoverable relative velocity limit, and the bounded relative velocity limit that decreases when the Chaser gets closer to the Target. The respect of the velocity constraints is due to both the reduction of the

action space and the reward logic. Therefore, the latter implies that they can be violated. Nonetheless, in this case the velocity constraints are respected.

To understand the learning behaviour of the Agent, the learning curve displayed hereafter must be considered. In this simulation, the Agent does not converge to a solution but instead finds one thanks to the variance of the learning process. In other words, the Agent finds an open-loop solution. To have a close-loop solution, the artificial neural networks should be fully trained which would imply that the learning curve converges to the solution.

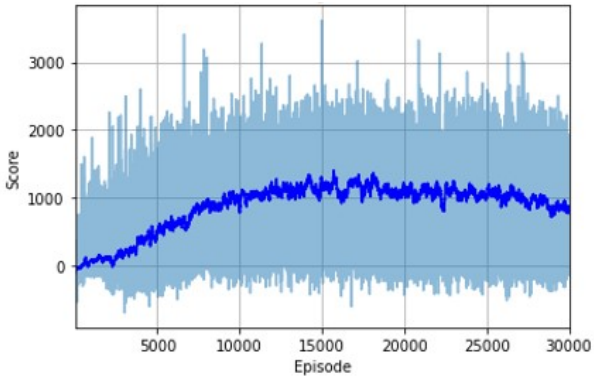


Figure 6.5: Case 1 - V-bar configuration: Learning behavior of the Agent (light blue), and its running average over 100 episodes (dark blue).

R-bar initial configuration

In this more complex configuration, the Agent has to understand that there is a docking cone by exploring its environment. Then, it has to learn how to dock by reaching and remaining inside the docking cone.
A solution trajectory is found after training the Agent for 30,000 episodes, which took about 3 hours.
The final state of the trajectory attests that the Chaser docks with the Target.

$$\|\mathbf{r}_{final}\| = 0.56\,m \;\; and \;\; |\theta_{N\,final}| = 4.9\,deg$$

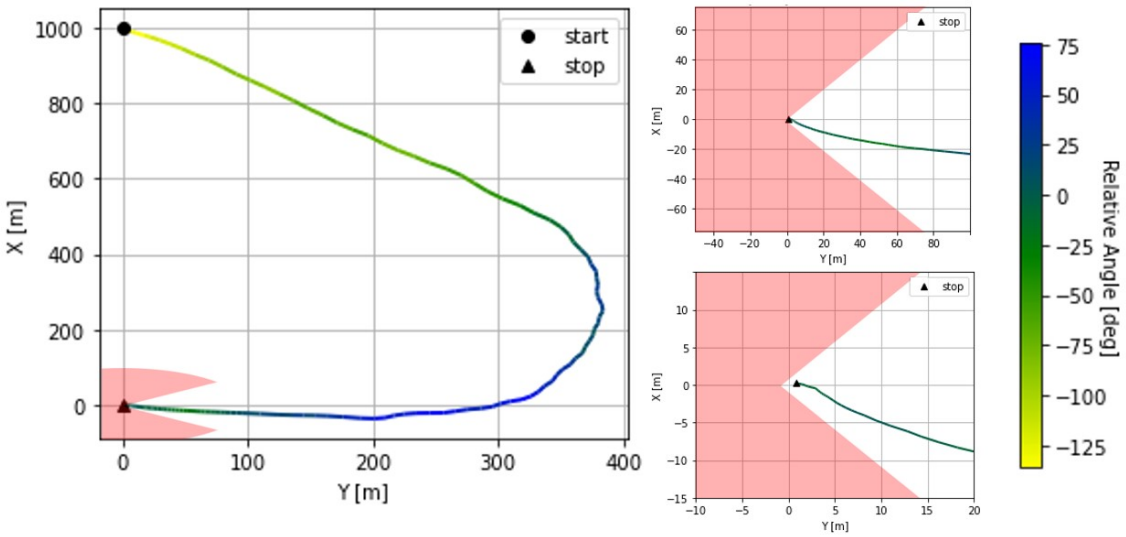


Figure 6.6: Case 1 - R-bar configuration: Full trajectory of the solution found (left), and trajectory zoomed on the Target (right). The docking cone is displayed in red.

The different constraints about the actuator physical limitation and safety are again respected as can be seen in figures 6.7 and 6.8.

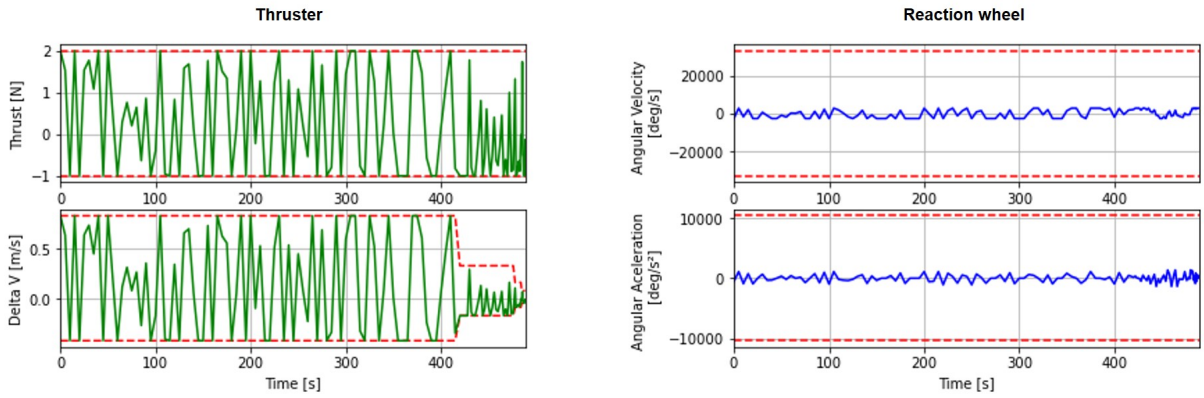


Figure 6.7: Case 1 - R-bar configuration: Behaviour of the Thruster in terms of thrust and commanded velocity impulse (left), and of the Reaction Wheel in terms of angular velocity and commanded angular acceleration (right).

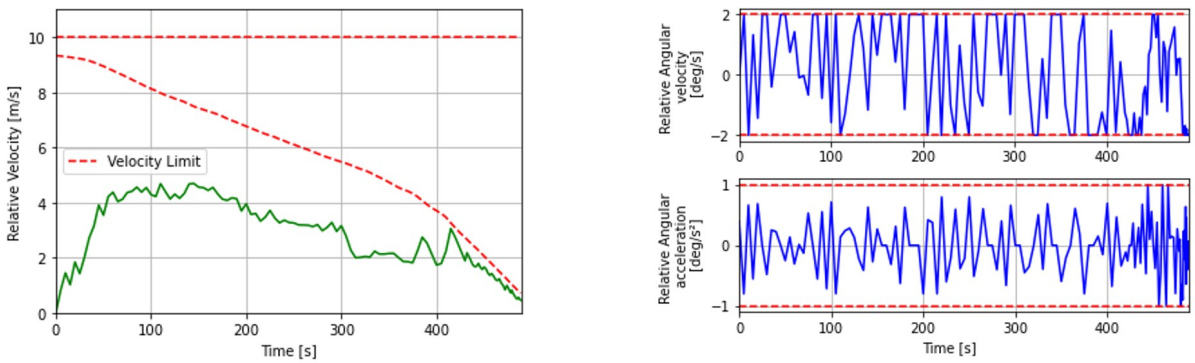


Figure 6.8: Case 1 - R-bar configuration: Relative velocity of the Chaser (left), and its angular velocity and acceleration (right).

In this more complex configuration, the Agent again finds a solution thanks to the variance of the learning process. By giving a closer look at the learning curve in figure 6.9, one can notice that only two episodes out of 30,000 are solutions. In the previous configuration, about ten episodes were solutions. This reflects the higher complexity of the R-bar configuration.

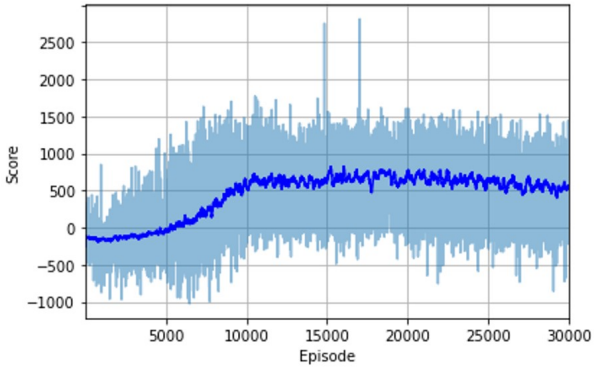


Figure 6.9: Case 1 - R-bar configuration: Learning behavior of the Agent (light blue), and its running average over 100 episodes (dark blue).

6.2 Case 2: Full ARPOD manoeuvre

The second and last test case aims to solve the full problem. To do so, the initial relative distance is fixed at 5 km, the Chaser is oriented toward the Target, and both the relative translation and angular velocity are null. By starting this far from the Target, the sequence of actions to dock is longer, and so the problem is harder to solve by the Agent.
In order not to increase too much the complexity, this case focuses on the V-bar initial configuration.

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

6.2.1 Reward Logic

The design of this reward logic follows the same ideas as the one in test case 1 (section 6.1.1). Only a few changes are made. For the sake of clarity, both shaping rewards and sparse rewards are recalled below by specifying the changes with respect to the previous case.

Shaping Rewards

While the shaping reward function dealing with the relative distance is the same, the one concerning the attitude is slightly different. Indeed, the linear term is not changed but an exponential term is added to encourage the Agent to keep an absolute attitude lower than 5 degrees.

$$R_t^{distance} = 10 + 10\,\frac{r_t}{r_0} + 5\,e^{1-0.01 r_t} \tag{6.3}$$

$$R_t^{attitude} = \begin{cases} 10\,\frac{5-|\theta_{N\,t}|}{180} + 4\,e^{-0.6\,|\theta_{N\,t}|} & if\ \ r_t \leq 100\,m \\ 0 & otherwise \end{cases} \tag{6.4}$$

where $\theta_{N\,t}$ is the attitude angle in degrees clipped between -180 and 180.

Finally, the shaping reward function is then the sum of the two previous functions.

$$R_t^{shaping} = R_t^{distance} + R_t^{attitude}$$

Sparse Rewards

With respect to the first case, only an extra bonus is given for the following reason.

- When the Chaser is in the proximity operation phase.

$$r_t \leq 1000\,m$$

Consequently, table 6.5 summarizes and quantifies the bonus and penalty logic of this test case.

Condition	Sparse reward
$r_t \leq \ \ 1000\,m$	$R_t^{sparse} = R_t^{sparse} + 10$
$r_t \leq \ \ 100\,m$	$R_t^{sparse} = R_t^{sparse} + 10$
$r_t \leq \ \ \ \ 5\,m$	$R_t^{sparse} = R_t^{sparse} + 500$
$r_t \leq 5\,m$ and $\lvert\theta_{N\,t}\rvert \leq 5\,deg$	$R_t^{sparse} = R_t^{sparse} + 500$
$r_t > r_{t-1}$	$R_t^{sparse} = -10$
$r_t > r_0 + 10\,m$	$R_t^{sparse} = -50$
$v_t > v_{max}$ or $v_t > v_{dock} + f_s\sqrt{\frac{F_{max}}{2m} r_t}$	$R_t^{sparse} = -10$
$r_t \leq 1000\,m$ and $(x_t - y_t > 1$ or $-x_t - y_t > 1)$	$R_t^{sparse} = 0$
$r_t \leq 1000\,m$ and $y_t < -1$	$R_t^{sparse} = -10$

Table 6.5: Case 2: Sparse reward logic for the attribution of bonuses and penalties

6.2.2 Hyperparameters

The set of hyperparmeters used in this second test case is as follows. Note that the Actor and Critic neural networks are unchanged with respect to the first case.

Name	Value
Horizon	64
mini-batch	16
epoch	4
γ	0.98
λ	0.65
ϵ	0.001
entropy coefficient	0.001 if $episode < 20000$ 0.0005 otherwise

Table 6.6: Case 2: Hyperparameters

Layer	Neurons	activation function
First hidden	130	ReLU
Second hidden	90	ReLU
Third hidden	60	ReLU
Output	2	tanh

Table 6.7: Case 2: Actor artificial neural network structure

Layer	Neurons	activation function
First hidden	130	ReLU
Second hidden	90	ReLU
Third hidden	60	ReLU
Output	1	linear

Table 6.8: Case 2: Critic artificial neural network structure

To avoid too long simulations, the maximum number of time steps per episode is fixed at 150.

6.2.3 Results

A solution trajectory is found after training the Agent for 30,000 episodes, which took about 5 hours. The longer simulation time is explained by the longer sequence of actions needed to reach the Target.
The final state of the trajectory attests that the Chaser docks with the Target.

$$\|\mathbf{r}_{final}\| = 0.98\,m \;\; and \;\; |\theta_{N\,final}| = 4.6\,deg$$

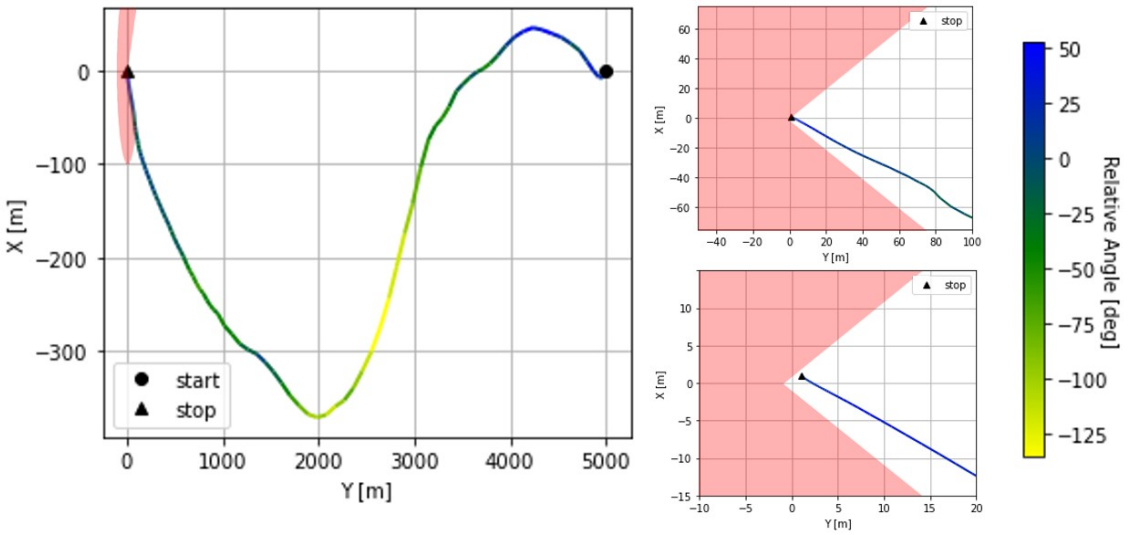


Figure 6.10: Case 2: Full trajectory of the solution found (left), and trajectory zoomed on the Target (right). The docking cone is displayed in red.

The different constraints about the actuator physical limitation and safety are still respected as can be seen in figures 6.11 and 6.12.

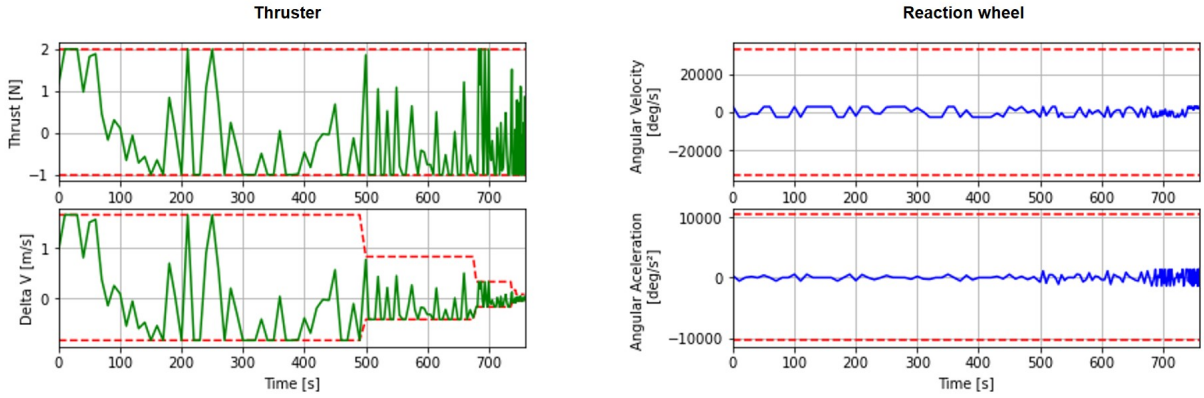


Figure 6.11: Case 2: Behaviour of the Thruster in terms of thrust and commanded velocity impulse (left), and of the Reaction Wheel in terms of angular velocity and commanded angular acceleration (right).

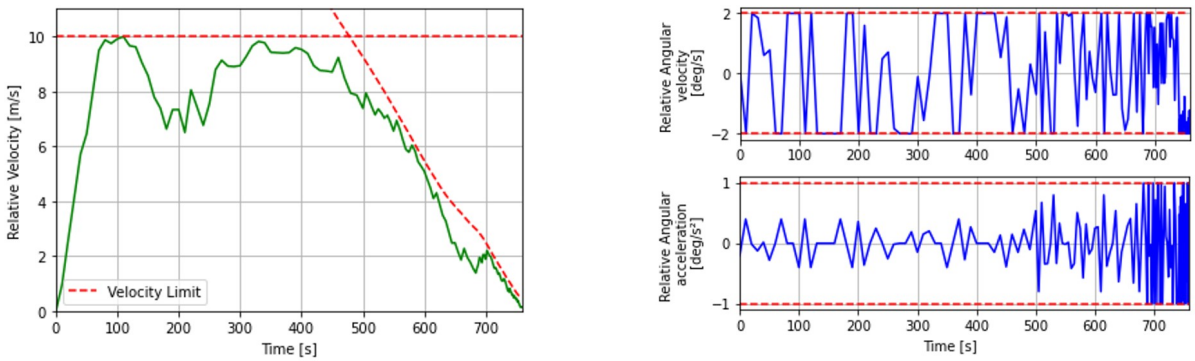


Figure 6.12: Case 2: Relative velocity of the Chaser (left), and its angular velocity and acceleration (right).

While in the previous cases the bounded recoverable velocity limit is dominant, in this test case starting with a bigger initial relative distance, both velocity constraints are impacting the result. Indeed, as it can be seen in Figure 6.12(left), for a relative distance higher than 1152.48 m the fixed recoverable relative velocity limit (*constraint 4*) is dominating the bounded relative velocity limit (*constraint 5*).

In this second case, the sequence of actions that has to be found is longer. As can been seen in the figures above, the Chaser still manages to dock with the Target by respecting all the constraints. However, by considering the learning behavior of the Agent represented in the figure below, it can be seen that it does not converge to a solution.

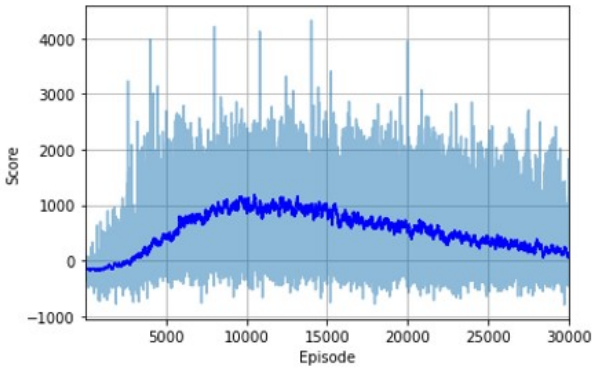


Figure 6.13: Case 2: Learning behavior of the Agent (light blue), and its running average over 100 episodes (dark blue).

6.3 Discussion

Throughout these two test cases, the behaviour of the Agent and the implementation of the problem can be put in perspective. Consequently, the promise of reinforcement learning methodology and its limitations can be discussed.

It can be observed that all the test cases presented previously respect the constraints imposed by the problem. This observation confirms the idea to model constraints by both restricting the action space and designing a proper reward logic. Indeed, the former implies that constraints cannot be violated and therefore are systematically respected. In this work, the reward function is used only to enforce the docking cone constraint and partially the velocity constraints. Both test cases prove that the Agent is able to understand and respect these two constraints. Consequently, reinforcement learning algorithms appear to be a promising technique to solve highly constrained problems.

However, none of the test cases converge toward the solution but instead find a solution thanks to the variance of the learning process. Hence, the solutions found are open-loop ones. It means that they provide a set of waypoints respecting all the constraints of this non-linear problem that could help fulfil requirements for increased autonomy in future spacecraft. Nonetheless, the motivation of using reinforcement learning algorithms is to generate a policy that is implementable as a feedback control law. Therefore, having fully trained networks that lead to successful learning convergence is a challenge that still needs to be addressed.
Furthermore, by carefully tuning the hyperparameters, as well as the reward function,

the previous limitation can be mitigated. However, the duration of the simulations makes this task very challenging. One may have noticed that the simple cases presented in this work already take a few hours to run with an average computer. For successful learning, the simulation time could go up to days with powerful computers. Consequently, the trial and error process necessary to properly tune the hyperparameters and reward function turns out to be very time-consuming.

Conclusion

The emerging field of reinforcement learning provides a potential route to solve some of the hardest non-linear problems in spacecraft dynamics. In this Master's thesis, the reinforcement learning algorithm known as Proximal Policy Optimization (PPO) is applied to under-actuated spacecraft guidance and control problems to better understand its applicability to such tasks. More specifically, this work focuses on the Autonomous Rendezvous, Proximity Operation and Docking (ARPOD) manoeuvre problem. This specific problem is very relevant to new space applications. However, ensuring mission safety is challenging. Hence, the objective and constraints of this work reflect the complexity necessary for a safe and efficient trajectory.

To begin with, this work gives a clear insight into the implementation of the reinforcement learning framework for a highly constrained problem. While the reward function is at the core of such algorithms, designing an efficient function that can model various constraints is tricky. In this thesis, constraints are handled by a hybrid modelisation based on restricted action spaces, shaping rewards, and sparse rewards. The use of this hybrid implementation eases the learning process and offers a way to ensure the respect of constraints.
Furthermore, the dilemma of exploration vs. exploitation is a challenge in most applications of reinforcement learning. In this work, the variance enables the Agent to explore different control actions and better improve its maximization of rewards. Thereby, it manages to solve the problem on isolated episodes. Even if a successful learning convergence would provide a low-cost control solution well suited for spacecraft, the solutions found can be seen as open-loop ones. As such, they provide a set of waypoints respecting all the constraints of this non-linear problem that could help fulfil requirements for increased autonomy in future spacecraft.
Finally, a limitation of reinforcement learning techniques is highlighted. Indeed, long-range manoeuvres such as a full ARPOD problem requires a long sequence of actions. However, by increasing the sequence of actions the problem becomes harder to solve by the Agent since action consequences are postponed. Consequently, the learning process turns out to be longer and therefore the proper tuning of the reward function and the hyperparameters of the PPO algorithm is more time-consuming.
Nevertheless, for short-range manoeuvres, the PPO algorithm appears to be a promising technique to solve highly constrained problems in spacecraft dynamics.

Future Work

While the results of this research demonstrate the promise of a reinforcement learning approach for solving highly constrained problems in spacecraft dynamics, more work will be required to address its present limitations. The absence of successful learning convergence is a significant challenge that needs to be addressed. Indeed, the optimality and robustness of the reinforcement learning solution could only be studied with a fully trained Agent. While Agent behaviour relies on the design of the reward function and the tuning of the hyperparameters, the relatively long duration of the learning process makes these tasks challenging.
A possible research focus lies in the improvement of the reinforcement learning algorithm. The clipping variant of the PPO algorithm is used in this work. Since this version introduced an additional hyperparameter, the KL divergence version would be worth investing to shortcut the tuning of the clipping parameter.
In addition, giving a closer look at the internal status of the artificial neural networks would enable their structure to be refined. In that case, a TensorFlow implementation of the algorithm would be necessary.

Finally, while this work already considers a large set of constraints, additional concerns on fuel consumption or mission time are worth being addressed. While the current implementation teaches the Agent how to safely dock, it does not consider the duration of the manoeuvre. Therefore, this richer problem would lead to a more realistic solution at the expense of a trickier reward function design.

Bibliography

[1] A. Flores-Abad et al. "A Review of Space Robotics Technologies for On-orbit Servicing". In: *Progress in Aerospace Sciences* 68 (2014), pp. 1–26.

[2] K.G. Symonds et al. "Operational Reality of Collision Avoidance Manoeuvres". In: *SpaceOps 2014 Conference* (2014), p. 1746.

[3] D. Woffinden and D. Geller. "Navigating the road to autonomous orbital rendezvous". In: *Journal of Spacecraft and Rockets* 44.4 (2007), pp. 898–909.

[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. 1976. Chap. 3, pp. 173–242.

[5] B.C. Hacker and J.M. Grimwood. *On the Shoulders of Titans: A History of Project Gemini*. Ed. by NASA SP-4203. 1977, pp. 1–16.

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

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

[8] I. Kawano et al. "Result of Autonomous Rendezvous Docking Experiment of Engineering Test Satellite-VII". In: *Journal of Spacecraft and Rockets* 38.1 (2001), pp. 105–111.

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

[10] S. McCamish, M. Romano, and X. Yun. "Autonomous distributed control algorithm for multiple spacecraft in close proximity operations". In: *AIAA Guidance, Navigation and Control Conference and Exhibit, Hilton Head, SC* (2007).

[11] R. Bevilacqua, T. Lehmann, and M. Romano. "Development and Experimentation of LQR/APF Guidance and Control for Autonomous Proximity Maneuvers of Multiple Spacecraft". In: *Acta Astronautica* 68 (2011), pp. 1260–1275.

[12] 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* 22.12 (2012), pp. 1398–1427.

[13] A. Weiss et al. "Model Predictive Control of Three Dimensional Spacecraft Relative Motion". In: *American Control Conference, Montréal* (2012), pp. 173–178.

[14] I. Garcia and J.P. How. "Trajectory Optimization for Satellite Reconfiguration Maneuvers with Position and Attitude Constraints". In: *American Control Conference, Portland, OR* (2005).

[15] V. Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[16] D. Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489.

[17] D. Silver et al. "Mastering the game of Go without human knowledge". In: *Nature* 550.7676 (2017), pp. 354–359.

[18] D.M. Chan and A. Agha-mohammadi. "Autonomous Imaging and Mapping of Small Bodies using Deep Reinforcement Learning". In: *IEEE Aerospace Conference, Big Sky, MT* (2019).

[19] A. Scorsoglio et al. "Actor-Critic Reinforcement Learning Approach to Relative Motion Guidance in Near-Rectilinear Orbit". In: *29 AAS/AIAA Space Flight Mechanics Meeting, Ka'anapali, HI* (2019).

[20] B. Gaudet, R. Linares, and R. Furfaro. "Deep Reinforcement Learning for Six Degree-of-Freedom Planetary Powered Descent and Landing". In: *Advances in Space Research* 65.7 (2020), pp. 1723–1741.

[21] J. Broida and R. Linares. "Spacecraft Rendezvous Guidance in Cluttered Environments Via Reinforcement Learning". In: *29 AAS/AIAA Space Flight Mechanics Meeting, Ka'anapali, HI* (2019), pp. 1–15.

[22] C.E. Oestreich, R. Linaresy, and R. Gondhalekar. "Autonomous Six-Degree-of-Freedom Spacecraft Docking Maneuvres via Reinforcement Learning". In: *AAS/AIAA Astrodynamics Specialist Virtual Lake Tahoe Conference* (2020).

[23] K. Hovell and S. Ulrich. "On Deep Reinforcement Learning for Spacecraft Guidance". In: *AIAA SciTech 2020 Forum, Orlando, FL* (2020).

[24] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. 2nd ed. The MIT Press Cambridge, 2018.

[25] H. van Hasselt. *Reinforcement Learning in Continuous State and Action Spaces*. Ed. by Springer Berlin Heidelberg. 2013, pp. 207–251.

[26] J. Schulman et al. *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. ICLR 2016, 2018.

[27] I. 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* 42.6 (2012).

[28] J. Schulman et al. "Trust Region Policy Optimization". In: *The Journal of Machine Learning Research* 37 (2015).

[29] J. Schulman et al. "Proximal Policy Optimization Algorithms". In: *OpenAI* (2017).

[30] V. Mnih et al. "Asynchronous Methods for Deep Reinforcement Learning". In: *The Journal of Machine Learning Research* 48 (2016).

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

[32] R.S. Sutton. "Two problems with backpropagation and other steepest-descent learning procedures for networks". In: (1986).

[33] N. Qian. “On the momentum term in gradient descent learning algorithms”. In: *Neural Networks : The Official Journal of the International Neural Network Society* 12 (1999), pp. 145–151.

[34] J. Duchi, E. Hazan, and Y. Singer. “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2121–2159.

[35] T. Tieleman and G. Hinton. “Lecture 6.5-RMSprop: Divide the gradient by a running average of its recent magnitude”. In: *COURSERA: Neural networks for machine learning* (2012).

[36] D.P. Kingma and J.L. Ba. “Adam: a Method for Stochastic Optimization”. In: *International Conference on Learning Representations* (2015).

[37] C. Jewison and R.S. Erwin. “A Spacecraft Benchmark Problem for Hybrid Control and Estimation”. In: *IEEE 55th Conference on Decision and Control (CDC), Las Vegas, NV* (2016), pp. 3300–3305.

[38] G.W. Hill. “Researches in the Lunar Theory”. In: *American Journal of Mathematics* 1.1 (1878), pp. 5–26.

[39] H.D. Curtis. *Orbital Mechanics for Engineering Students*. Ed. by Butterworth-Heinemann. 3rd ed. 2014.

[40] C.D. Petersen et al. “Challenge Problem: Assured Satellite Proximity Operations”. In: *AAS/AIAA Space Flight Mechanics Meeting* (2021).

[41] Y. LeCun et al. “Efficient BackProp”. In: *Neural Network: tricks of the trade* (1998).

[42] A.Y. Ng. “Shaping and Policy Search in Reinforcement Learning”. PhD thesis. University of California, Berkeley, 2003.

[43] M. Minsky. “Steps Toward Artificial Intelligence”. In: *Investigative Reporters and Editors* (1961).