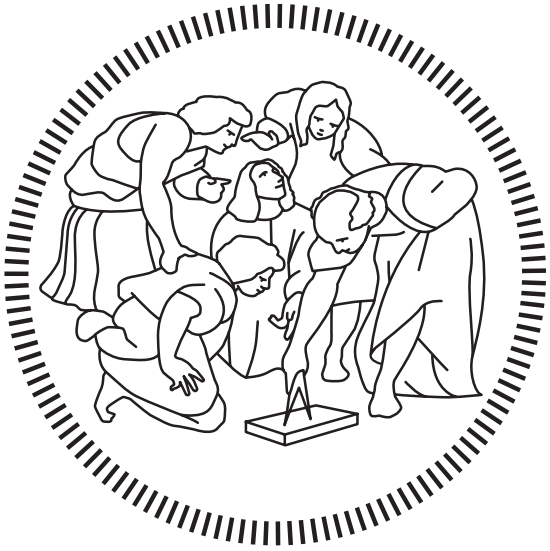
Politecnico di Milano

SCHOOL OF INDUSTRIAL AND INFORMATION ENGINEERING

Master of Science – Space Engineering



Reinforcement Learning based Model Predictive Control guidance for pinpoint landing on low gravity bodies

Supervisor

Prof. Mauro Massari

Candidate

Luca Maria Stefano Pascali – 920396

Academic Year 2019 – 2020

Acknowledgements

Throughout the development of this thesis I have received a great deal of support and assistance.

I would first like to thank my supervisor, Professor Mauro Massari, who has followed me and given me the possibility to work on new interesting topics.

I would also like to thank the Politecnico di Milano university itself for giving me the possibility to use the computational resources of the department that have been fundamental for the development of the thesis.

In addition, I would like to thank all the people that have always supported and believed in me such as my family, colleagues and friends.

Abstract

The aim of the following thesis is to present a new kind of guidance that allows to obtain a pinpoint landing on low gravity bodies without atmosphere. After a preliminary analysis of different kind of guidance laws already present, a new method has been investigated. This method is obtained by combining the Model Predictive Control (MPC) guidance with Reinforcement Learning (RL) techniques. The main idea behind this method is to enhance the performances of MPC by computing the cost function minimization through machine learning processes in order to overcome the problem related to the computational time usually needed for this kind of operation. Moreover, the combination with the artificial intelligence enables to consider a larger variety of constraints by embedding them in the definition of the learning environment. In order to achieve these tasks, three kinds of Reinforcement Learning algorithms have been considered and applied. Finally the results presented at the end of the work confirm the possibility of the presented method to have an important impact for future missions if deepened and enhanced in the correct way.

Contents

Acknowledgements iii

Abstract v

Contents viii

List of Figures x

List of Tables xi

1 Introduction 1
- 1.1 State of the art . . . 2
  - 1.1.1 Guidance Laws . . . 3
- 1.2 Guidance choice . . . 12
- 1.3 Thesis organization . . . 13

2 Method Description 15
- 2.1 Dynamics . . . 16
  - 2.1.1 Equations of motion . . . 16
  - 2.1.2 Initial state considerations . . . 18
- 2.2 Machine Learning general features . . . 20

3 Reinforcement Learning 23
- 3.1 Q-learning . . . 24
- 3.2 Policy Gradient . . . 25
  - 3.2.1 Reinforce with Baseline . . . 27
  - 3.2.2 Actor Critic . . . 28
  - 3.2.3 Advantage Actor-Critic . . . 29
- 3.3 Environment Description . . . 30
- 3.4 Algorithm Implementation . . . 34
  - 3.4.1 Neural Network Construction . . . 35
  - 3.4.2 Learning Algorithm implementation . . . 38

4 Results 41
- 4.1 Test Case Description . . . 41

4.2 Introduction to the results 43
4.3 2D 45
4.3.1 Reinforce with Baseline 46
4.3.2 Actor-Critic 48
4.3.3 Advantage Actor-Critic 49
4.4 3D 53
4.4.1 Advantage Actor-Critic 54
4.4.2 A2C reduced horizontal error case 60

5 Conclusions **67**

Acronyms **69**

Bibliography **72**

List of Figures

Figure 1.1 MSL Curiosity powered descent controller 2
Figure 1.2 Hybrid Guidance Scheme 8

Figure 2.1 MPC+RL guidance architecture block scheme 15
Figure 2.2 Lander thrusters configuration in body centred reference frame 18
Figure 2.3 Lander initial state errors and orientation 19
Figure 2.4 Reference system orientation 19
Figure 2.5 Fully Connected Neural Network example with one hidden layer 21

Figure 3.1 Reinforcement Learning scheme: Interaction between an agent and its environment 23
Figure 3.2 Reinforce with Baseline scheme 27
Figure 3.3 Actor-Critic scheme 29
Figure 3.4 Distance Reward 33
Figure 3.5 J Discount 33
Figure 3.6 Logarithmic Probability Network Model 35
Figure 3.7 Actor Network Model 36
Figure 3.8 Entropy Network Model 37
Figure 3.9 Value Network Model 37

Figure 4.1 Philae landing scenario 42
Figure 4.2 Reinforce with Baseline case A, $-60\,\mathrm{m}$ to $-40\,\mathrm{m}$ sub-range 47
Figure 4.3 Reinforce with Baseline case B, $-60\,\mathrm{m}$ to $-40\,\mathrm{m}$ sub-range 47
Figure 4.4 Reinforce with Baseline case C, $-60\,\mathrm{m}$ to $-40\,\mathrm{m}$ sub-range 47
Figure 4.5 Actor-Critic $-60\,\mathrm{m}$ to $-40\,\mathrm{m}$ sub-range 48
Figure 4.6 Advantage Actor-Critic $-20\,\mathrm{m}$ to $0\,\mathrm{m}$ range 49
Figure 4.7 Advantage Actor-Critic $-40\,\mathrm{m}$ to $-20\,\mathrm{m}$ range 49
Figure 4.8 Advantage Actor-Critic $-60\,\mathrm{m}$ to $-40\,\mathrm{m}$ range 50
Figure 4.9 Advantage Actor-Critic $-80\,\mathrm{m}$ to $-60\,\mathrm{m}$ range 50
Figure 4.10 Advantage Actor-Critic $-100\,\mathrm{m}$ to $-80\,\mathrm{m}$ range 50
Figure 4.11 2D case test Trajectories 52
Figure 4.12 2D $-100\,\mathrm{m}$ to $-80\,\mathrm{m}$ range u_x and u_z 53

Figure 4.13 2D −20 m to 0 m range u_x and u_z 53
Figure 4.14 Advantage Actor-Critic −20 m to 0 m range 54
Figure 4.15 Advantage Actor-Critic −40 m to −20 m range 55
Figure 4.16 Advantage Actor-Critic −60 m to −40 m range 55
Figure 4.17 Advantage Actor-Critic −80 m to −60 m range 55
Figure 4.18 Advantage Actor-Critic −100 m to −80 m range 56
Figure 4.19 3D case test Trajectories 57
Figure 4.20 3D case test Landing Points 57
Figure 4.21 3D −100 m to −80 m range u_x, u_y and u_z 58
Figure 4.22 3D −20 m to 0 m range u_x and u_y 58
Figure 4.23 3D case test overall Landing Points on ground plane compared to the initial maximum horizontal position error . . . 59
Figure 4.24 3D case test overall Landing points compared to the initial maximum horizontal position error 60
Figure 4.25 Advantage Actor-Critic −20 m to 0 m range 61
Figure 4.26 Advantage Actor-Critic −40 m to −20 m range 61
Figure 4.27 Advantage Actor-Critic −60 m to −40 m range 61
Figure 4.28 Advantage Actor-Critic −80 m to −60 m range 62
Figure 4.29 Advantage Actor-Critic −100 m to −80 m range 62
Figure 4.30 3D case test Trajectories 63
Figure 4.31 3D case test Landing Points 64
Figure 4.32 3D −100 m to −80 m range u_x, u_y and u_z 65
Figure 4.33 3D −20 m to 0 m range u_x, u_y and u_z 66

List of Tables

Table 1.1 MSL entry features 3

Table 3.1 y_{pred} of $loss_{logp}$ and $loss_{value}$ 38
Table 3.2 y_{true} of $loss_{logp}$ and $loss_{value}$ 38

Table 4.1 Initial nominal lander conditions for the test case 41
Table 4.2 Thruster features for the learning process 42
Table 4.3 Electric thrusters features 43
Table 4.4 Main learning parameters values 44
Table 4.5 2D initial state conditions 45
Table 4.6 RB Case A, B and C terminal rewards for the -60 m to -40 m 46
Table 4.7 2D A2C terminal rewards for every sub-range 51
Table 4.8 2D test initial state conditions 51
Table 4.9 3D initial state conditions 54
Table 4.10 3D A2C terminal rewards for every sub-range 54
Table 4.11 3D test initial state conditions 56
Table 4.12 3D A2C terminal rewards for every sub-range 60
Table 4.13 3D test initial state conditions 63

Chapter 1

Introduction

The aim of this thesis is to identify a method in order to get optimal results in the field of the pinpoint landing problem on low gravity bodies without atmosphere. Indeed this is an issue of great importance considering the increasing interest of the last decades in planet exploration as well as the renewed interest towards the Moon for what concerns the building of infrastructures for easy access to the Lunar surface in the next years. For these reasons the landing system technology will need to progress to satisfy the demand for more stringent requirements and an higher landing accuracy is for sure one of them. In fact the pinpoint landing problem subsumes other issues such as the hazard avoidance problem or the possibility to enable a landing in peculiar regions such as the centre of a crater. Furthermore, in case of having a rover to be deployed, a landing position closer to a location of scientific interest reduces the risk of the rover malfunctioning before it reaches the desired site. Moreover, reducing the distance the rover is required to travel can relax the design requirements for the rover, with a potential reduction in rover mass. As a consequence, landing accuracy on the order of several meters is desirable as it would both reduce mission risk and extend the scope of feasible missions. In general there are two necessary capabilities to enable a lander to achieve a pinpoint landing. First, the navigation system must be capable of accurately estimating the lander's state during the powered descent phase. Second, the lander's guidance and control system must be capable of using these state estimates to achieve a pinpoint landing. Nowadays the state of the art is to have three separately optimized systems: navigation, guidance and control. In this work the focus is to find a guidance to enable pinpoint landing performances on low gravity bodies without atmosphere. At first several guidance laws were taken under consideration analysing the advantages and disadvantages of each of them together with the performances achieved in some real applications, considering also landing environments different from the target one. In this way it was possible to have a general insight of what is the state of the art in this field, which is fundamental to understand in which direction to

proceed in the development of a new or at least renewed technology.

1.1 State of the art

At first some real landing missions were considered in order to understand the state of the art of the landing precision together with the involved techniques. For what concerns successful missions, the 3-sigma ellipse of 19 km by 7 km for the Martian landing achieved by the Mars Space Laboratory (MSL) is for sure one of the best results ever obtained in this sense, in which the powered descent phase employs a guidance very similar to the one introduced by the Apollo mission [1]. This guidance was based on an iterative approach that computed on the ground a flyable reference trajectory in the form of a quartic polynomia and generated an acceleration command that targets the final condition of the trajectory. This trajectory, which was not optimized for minimal fuel usage, is designed so that it can be tracked thanks to six feedback control loops, taking position and attitude estimates from the navigation filter as showed by Figure 1.1 ([1], [2]). One of the main reasons that avoids the landing accuracy to be even

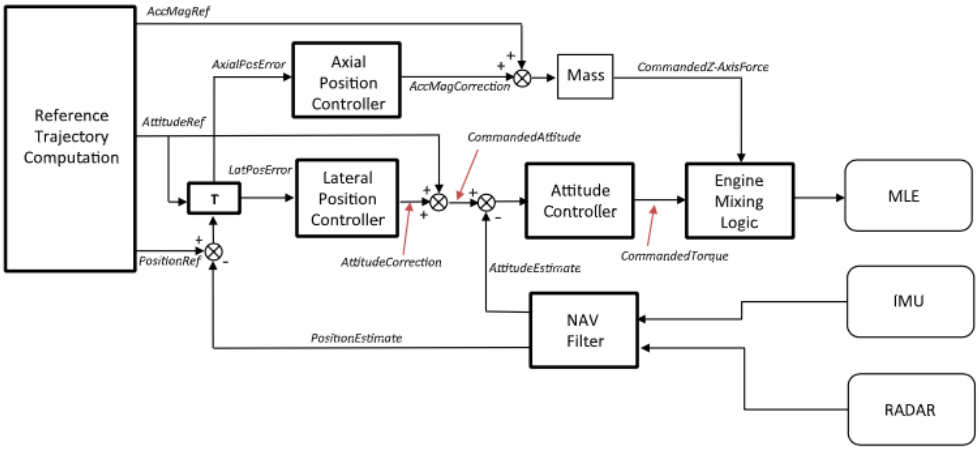


Figure 1.1. MSL Curiosity powered descent controller

better is that even in case of an optimal estimation of the lander state in the target-centered reference frame the guidance law is only capable of relatively small divert maneuvers. In any case it is still possible to appreciate the degree of accuracy obtained if the landing ellipse dimensions are compared with the initial conditions at the beginning of the Entry, Descent and Landing (EDL) phase reported in Table 1.1. Moreover, the recent events of the Mars 2020 mission, with the successful landing of the Perseverance rover inside the Jezero Crater on Mars of the last 18th February 2021, gives even more importance to what done by the MSL, since it was the base of this last mission. Practically speaking the Mars 2020 EDL design adds two great improvements: a strong

Mission	Entry velocity [km/s]	Entry flight path angle [deg]	Entry altitude [km]	Entry mass [kg]
MSL	6	-15.2	125	2800

Table 1.1. MSL entry features

reduction of the landing ellipse (7.7 km by 6.6 km) and the introduction of Terrain Relative Navigation, which is the ability to determine in the last minutes that the spacecraft is heading unluckily toward a known hazard, and to divert to a safe spot [3], [4]. Unfortunately there are still few information about the exact guidance applied.
Of course it is important to underline that the landing conditions of MSL are considerably different from the ones taken into consideration in this thesis due to the crucial importance of the Martian atmosphere and gravity for the above described mission. However, what presented is still an important insight in terms of both landing guidance and precision. In fact a similar approach could be taken into consideration even for the pinpoint landing problem case with low gravity and no atmosphere. In this sense it is important to consider that, as already said, the MSL guidance was based on the one used during the Apollo mission, whose landing body was the Moon, hence a body without atmosphere.
Another successful mission that can be taken into consideration is the Rosetta mission with the landing of Philae on the 67P/Churyumov–Gerasimenko comet in 2014. The environment where this landing took place is very similar to the target one of this work with the absence of atmosphere and the presence of a very low gravity. Nevertheless, the landing technique applied was a ballistic kind of descent without the involvement of any particular guidance law, resulting in a landing ellipse of around 100 m starting from an initial altitude between 1 km to 2 km with a pre-adjustable velocity between 0.05 m/s to 0.52 m/s [5].
Finally, in order to better understand what is the actual state of the art in terms of guidance laws, different approaches were analyzed and reported in the following section even if some of them have not been applied yet during real missions or are applied in other contexts such as missile guidance.

1.1.1 Guidance Laws

Before starting with the description of the analysed guidance laws, an important aspect to be considered is the difference between trajectory-tracking and trajectory-free guidance, since the following laws are of both types. The first kind of guidance consists in generating a trajectory in real time based on the lander's state at the beginning of the powered descent phase that is then passed to a control system that tracks it by determining which thrusters to fire and at

what thrust level. On the contrary, the second kind is a global guidance that maps the lander's state in the target-centered reference frame to a commanded acceleration in the inertial frame that then is passed to an actuator control system.

Zero-Effort-Miss/Zero-Effort-Velocity

The Zero-Effort-Miss/Zero-Effort-Velocity (ZEM-ZEV) is a trajectory-free guidance that outputs an acceleration that aims to minimize the overall energy of the system identified by the following cost function:

$$J = \frac{1}{2}\int_{t_0}^{t_f} \vec{a}^T \vec{a} dt \tag{1.1}$$

This method, that was at first applied to the missiles, has been developed and taken under consideration in the landing field in the last years. In order to obtain the desired result an Hamiltonian function is evaluated:

$$H = \frac{1}{2}\vec{a}^T\vec{a} + \vec{p_r}^T\vec{v} + \vec{p_v}^T(\vec{a} + \vec{g}) \tag{1.2}$$

where $\vec{p_r}$ and $\vec{p_v}$ are the co-states. The optimal acceleration is the one that maximises the Hamiltonian function and is described by the following equation:

$$\vec{a} = \frac{k_r}{t_{go}^2} Z\vec{E}M + \frac{k_v}{t_{go}} Z\vec{E}V \tag{1.3}$$

where t_{go} is the time-to-go (final time - current time), $Z\vec{E}M = \vec{r}_f - \vec{r}_{nc}$ and $Z\vec{E}V = \vec{v}_f - \vec{v}_{nc}$. $\vec{r}_{nc}$ and $\vec{v}_{nc}$ are the position and velocity computed through the integration of the equations of motion from the current time instant to the end of the mission with control action equal to zero. This feedback guidance is attractive because of its analytical simplicity and accuracy: guidance mechanization is straightforward and it can theoretically drive the spacecraft to a target location on the planetary surface both autonomously and with minimal guidance errors. It resulted to be finite time stable and robust to uncertainties (if proper sliding parameters are added) but with no capability of enforcing thrust or flight constraints as showed by Roberto Furfaro and Massari in [6].
Eventually this method represents a very good candidate as the basis for the development of a new guidance law. However, the presented drawbacks regarding thrust and flight constraints constitute important obstacles for the achievement of a pinpoint landing on low gravity bodies without atmosphere, where the thrusting conditions represent a crucial aspect.

Guidance for Fuel Optimal Large Divert

The Guidance for Fuel Optimal Large Divert (G-Fold) is an algorithm that is developed to compute, on board in real-time, global fuel optimal trajectories for large divert maneuvers necessary for planetary pinpoint or precision landing. This trajectory-tracking guidance is based on the lossless convexification approach that provides the solution of a general class of non-convex optimal control problems via convexification method. This method aims to minimize a specific function that describes the fuel consumption (Equation 1.4) while respecting several constraints, that could be both convex and non-convex as shown by Brhçet Açikmese et al. in [7].

$$max \quad m(t_f) \rightarrow min \int_0^{t_f} \alpha |\vec{T_c}| dt \tag{1.4}$$

subject to:

$$\vec{x}(t) \in \vec{X} \quad \forall t \in [0, t_f] \tag{1.5}$$

$$0 < \rho_1 \leq |\vec{T_c}(t)| \leq \rho_2, \;\; \vec{n}^T \vec{T}c(t) \geq |\vec{T_c}(t)| \cos\theta \tag{1.6}$$

$$m(0) = m_0 \tag{1.7}$$

$$\vec{r}(0) = \vec{r}_0, \;\; \dot{\vec{r}}(0) = \dot{\vec{r}}_0 \tag{1.8}$$

$$\vec{r}(t_f) = \vec{0}, \;\; \dot{\vec{r}}(t_f) = \vec{0} \tag{1.9}$$

Equation 1.5 reports the constraints regarding the state in general such as the glide slope constraint on the position vector and an upper bound constraint on the velocity vector magnitude, both convex. In Equation 1.6 are reported the thrust constraints in terms of upper boundary (convex), lower boundary (non-convex) and thrust direction (convex if $\theta \leq \frac{\pi}{2}$, non-convex when $\theta > \frac{\pi}{2}$). Then Equation 1.7, Equation 1.8 and Equation 1.9 represent the initial condition for the mass and the state and the final condition of the state. In order to solve the problem the lossless convexification method is applied to convert the optimal control problem into an equivalent problem with convex control constraints. This is achieved by using a particular convex relaxation of the control constraints. In the relaxed problem a slack variable is introduced (Γ) to lift the control space to a higher dimension (with one additional dimension) and relaxing the non-convex set of controls to a convex set as reported in Equation 1.10.

$$|\vec{T_c}(t)| \leq \Gamma(t), \;\; 0 < \rho_1 \leq \Gamma(t) \leq \rho_2, \;\; \vec{n}^T \vec{T}c(t) \geq \cos\theta \Gamma(t) \tag{1.10}$$

As J. M. Carson and Blackmore describe in [8] the optimal solution of the relaxed problem is also the optimal solution of the non relaxed one. So the next step is to discretize the optimal control in such a way to be able to use the Interior Point Methods (IPM) of convex optimization as Açikmese and S.R.Ploen show in [9]. IPM are powerful computational methods that establish guarantees of obtaining global optimal solutions of convex optimization problems to any accuracy in polynomial time, which means that as the size of the problem increases the computational complexity does not increase too much. Finally, thanks to this method the thrust vector over time is computed and as a consequence the overall trajectory is calculated as well.
Despite the good results obtained, the G-Fold presents some drawbacks. In fact the obtained result may not be the most fuel efficient trajectory and the method may not be suitable for missions where the cone constraint is incompatible with the mission goals, as it could be in the case of landing inside a crater. This could be a problem for the development of a more advanced version of the guidance that this work aims to develop, since in general the bodies with low gravity and without atmosphere are usually relatively small with a non-smooth surface, hence the presence of features like craters is probable.

Universal Powered Guidance

The Universal Powered Guidance (UPG) is a trajectory-tracking guidance that, similarly to G-Fold, tries to achieve a pinpoint landing minimizing the propellant consumption. However, the result is achieved in a very different way with respect to the previous method. In fact in this case a bang-bang magnitude thrust profile is considered and the whole method focuses on determining the number of switches and the optimal burn times of each sub-arc. In the case of a pinpoint landing, the performance index is just the propellant consumption, and the optimal thrust profile is the one that maximises the following Hamiltonian with dimensionless variables presented by Lu in [10]:

$$H = \vec{p_r}^T \vec{v} + \vec{p_v}^T \dot{\vec{v}} - p_m \frac{\vec{T}}{v_{ex}} \sqrt{\frac{R_0}{g_0}} - \frac{\vec{T}}{v_{ex}} \sqrt{\frac{R_0}{g_0}} \tag{1.11}$$

where $\vec{p}_r$, $\vec{p}_v$ and p_m are the co-states and the resulting optimal unit thrust direction vector is

$$\vec{u}_T = \frac{\vec{p}_v}{|\vec{p}_v|} \tag{1.12}$$

As demonstrated in [10] the optimal thrust magnitude T for the powered descent problems will be either on the upper bound T_{max} or lower bound T_{min}, that is, a bang–bang profile. Any intermediate thrust value in a finite interval is not optimal. Then, to actually define completely the thrust profile for a pinpoint

landing, the following conditions have to be considered:

$$\vec{r}(t_f) = \vec{r}^* \tag{1.13}$$

$$\vec{v}(t_f) = \vec{v}^* \tag{1.14}$$

$$H(t_f) = 0 \qquad (transversality\ \ condition) \tag{1.15}$$

These conditions constitute a root finding problem of a system of seven nonlinear algebraic equations for $\vec{z}$, with $\vec{z} = (\vec{p}_{v_0}^T,\ \vec{p}_{r_0}^T,\ t_f)^T$. Although a number of possible numerical methods exist for solving this system of equations, such as the Newton–Raphson method, the dogleg trust-region method by Powell is the most robust in offering reliable convergence for several problems [10]. However, some other ways to solve the problem in some peculiar cases have been successfully actuated. For instance, in the case of considering a 3D dynamics, a common solution is the one of considering at maximum two switches. In this way the correspondent time instants t_1 and t_2 can be found thanks to an outer loop minimization method. This method consists in treating t_1 as a tuning parameter to be fixed and then finding the optimal t_2 in order to minimize the performance index through a loop.
The advantages of UPG, especially if compared to G-Fold, are the simpler algorithm, the absence of the need of customization of an optimization solver and the flexibility of accommodating different problem formulations. On the other hand, this method presents some important drawbacks such as the difficulty of imposing constraints such as the glide slope, thrust direction and velocity direction. In addition, it also presents the problem of having no theoretically guaranteed convergence. As already said for the ZEM-ZEV guidance, the difficulty of imposing certain constraints such as the thrusting direction represents an important drawback for the considered pinpoint landing case.

Hybrid Guidance

The Hybrid Guidance (HG) is a method that combines different techniques to achieve a precise landing. The analyzed case of a hybrid system utilizes a global controller that implements an optimal guidance law augmented with a sliding mode (Optimal Sliding Guidance (OSG)) to bring the lander from an initial state to a predetermined reference trajectory and an Linear quadratic regulator (LQR) - based local controller to bring the lander to the desired point on the surface as described by Daniel R. Wibben and Sanfelice in [11]. The idea behind this guidance law is the utilization of a combination of a controller that works well globally with one that is more efficient near the desired target point, increasing

the overall flexibility and precision of the system. A switching logic between the two control laws is implemented in a hybrid controller to develop a more robust guidance law as shown in Figure 1.2. This control strategy is usually called "throw-and-catch". In fact the OSG 'throws' the lander from an initial state to a state near a pre-defined reference trajectory while the LQR 'catches' the lander and forces the system to track the reference trajectory to the target point. The

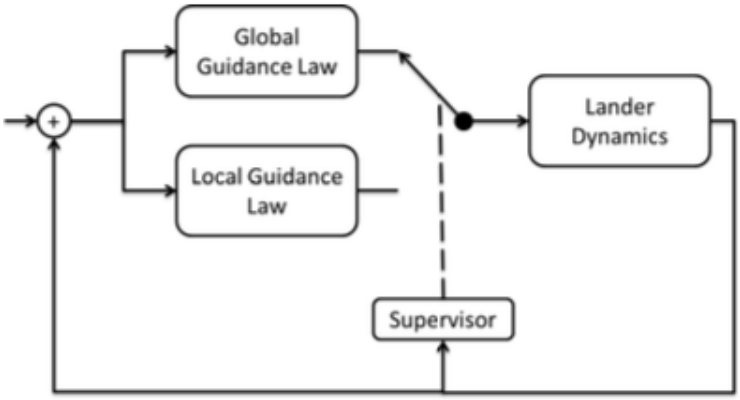


Figure 1.2. Hybrid Guidance Scheme

development of both guidance laws involves defining a reference trajectory as a target state for the global guidance law and as a reference to track for the local guidance law. For this reason it is possible to define the Hybrid Guidance as a trajectory-tracking guidance. The chosen reference trajectory is determined and found numerically by solving the minimum-fuel optimal landing problem, seen also in previous cases, via pseudo-spectral methods as described in [11].
Concerning the OSG, it is an enhancement of the Zero-Effort-Miss/Zero-Effort-Velocity, already described in section 1.1.1. At first the ZEM-ZEV guidance law is rewritten in terms of error state $\vec{\epsilon}_r$ and $\vec{\epsilon}_v$, then a sliding surface is defined as follows:

$$\vec{s} = \vec{\epsilon}_v + \lambda \vec{\epsilon}_r = \vec{0} \tag{1.16}$$

This surface goes to zero when both the state errors tend towards null values. For this reason the idea is to construct the guidance law in such a way that the system is always driven to the sliding surface. This can be done by taking the derivative of Equation 1.16 and introducing the optimal control given by the ZEM-ZEV guidance. In this way, after some steps described in [11], it is possible to obtain a new guidance law:

$$\vec{a}_c(\tau) = \frac{k_1}{t_{go}^2}\vec{\epsilon}_r + \frac{k_2}{t_{go}}\vec{\epsilon}_v - \vec{g} - \frac{\Phi}{t_{go}} sign(\vec{s}) \tag{1.17}$$

where Φ is a positive non-null constant. Daniel R. Wibben and Sanfelice show, through the use of Lyapunov's second method, that this guidance law is also globally stable [11].
Regarding the LQR part, the system dynamics must be linearized at first. In this way the dynamics can be written in the following way:

$$\Delta\dot{\vec{y}} = A\Delta\vec{y} + B\Delta\vec{u} \tag{1.18}$$

If the action is defined as a linear feedback controller ($\Delta\vec{u} = -k\Delta\vec{y}$), the system takes the following form:

$$\Delta\dot{\vec{y}} = (A - Bk)\Delta\vec{y} = A_c\Delta\vec{y} \tag{1.19}$$

where k is the gain of the feedback controller. At this point, the LQR approach can be introduced. It consists in finding the value of k that minimizes the following cost function:

$$J = \int_0^\infty (\Delta\vec{y}^T Q\Delta\vec{y} + \Delta\vec{u}^T R\Delta\vec{u})d\tau \tag{1.20}$$

subject to Equation 1.18 as a physical constraint, where Q and R are weighting matrices. The solution of this problem is expressed in Equation 1.21 where P is the matrix that satisfies the Riccati equation reported in [11].

$$k = R^{-1}B^T P \tag{1.21}$$

As described, this method represents a very attractive guidance solution thanks to its flexibility and precision that are aspects of fundamental importance for the considered landing case. However, the fact that two different kind of methods are combined together could also represent a drawback in case of undesired errors in one of the two involved optimization approaches. In fact a minor error in the development of one guidance could bring to a wrong application of the other one resulting in an overall globally higher error.

Generalized Proportional Navigation

The Generalized Proportional Navigation (GPN) is a trajectory-free guidance law mainly used in the field of missile guidance. However, if the particular case of a non-moving target is considered, it could have some interesting applications in the field of landing as well. The equations of motion are written in polar coordinates with the origin fixed at the interceptor, while the velocity vector is

considered as the target vector velocity relative to the interceptor.

$$\vec{v} = \vec{v}_r\vec{e}_r + \vec{v}_\theta\vec{e}_\theta = \dot{\vec{r}}\vec{e}_r + \vec{r}\dot{\theta}\vec{e}_\theta \tag{1.22}$$

The equations of motion are the following:

$$\frac{d\vec{v}}{dt} = \vec{a}_t - \vec{a}_m \tag{1.23}$$

where $\vec{a}_t$ is the target acceleration and $\vec{a}_m$ the interceptor one.
After defining ψ as the angle between the direction of the interceptor acceleration and the direction normal to the Line of Sight (LOS), the interceptor acceleration is considered to be proportional to the LOS angular rate as follows:

$$\vec{a}_m = k_r\dot{\theta}\vec{e}_r + k_\theta\dot{\theta}\vec{e}_\theta \tag{1.24}$$

where $k_r = \lambda\vec{v}_{\theta 0}\sin\psi$, $k_\theta = -\lambda\vec{v}_{r0}\cos\psi$ and λ is the navigation constant as Ciann-Dong Yang and Ghen describe in [12].
As said, this kind of solution could be used in the case of landing after imposing the target velocity and acceleration equal to zero. Despite being very interesting for its simplicity and efficiency, there are still several problems regarding especially some constraints such as the difficulty of granting low velocities once reached the landing point, which is actually an important problem for the kind of landing considered in this work. In fact the impact with a relatively high velocity on the surface of a low gravity body could result in the bouncing of the lander itself, compromising the final landing precision.

Reinforcement Learning derived Integrated Guidance and Control

The Reinforcement Learning derived Integrated Guidance and Control (IGCRL) is a trajectory-free guidance that maps the navigation system estimate of the lander state directly to commands specifying thrust levels for each engine by means of a global policy learned by the use of reinforcement learning methods as described by Brian Gaudet and Furfaro in [13].
The general problem is formulated in a similar manner with respect to the previously seen methods: the main objective is to find the thrust program and flight time that minimize a cost function that expresses the propellant consumption while respecting some given constraints such as the equations of motions, the glide slope and thrust constraints and while also following the boundary conditions regarding the initial and final states. The peculiarity of this approach is the way by which this result is obtained. In fact the RL framework is defined such that the agent (i.e., the lander) responds to a single reward signal which needs to be generally maximized during the search process. The

reward is a function that tells the system how good the action taken by the agent is and for this reason it is strictly related to all the constraints and the cost function that define the problem. Indeed the construction of the reward function is a fundamental aspect of the RL process, and it is dependent on the kind of problem that the system has to face.
Another fundamental aspect is the kind of learning process selected. In fact this selection is the one that determines how the policy is learnt depending on the rewards received after each agent's action. Finally, the learnt policy is the actual function that maps each state to the correspondent action needed time by time to bring the lander to the wanted landing point.
This kind of solution is very interesting for the application considered in this thesis since it gives very good results and allows to impose a great variety of constraints by simply modifying the reward function. This results in a great flexibility of the method that makes this approach really attractive for the pinpoint landing case considered in this thesis. However, there is the drawback of a lack of local or global stability guarantees.

Model Predictive Control

The MPC is a trajectory-free guidance that is able to give at each time step the needed control action to bring the state to the desired landing point. This method aims to handle different constraints and step by step calculation at the same time by means of a receding horizon optimization method reported by Ting Wang, Ma, and Liang in [14]. The algorithm considers the following general performance index to be minimized dependent on lander's state and control:

$$J = \int_0^{t_f} (\vec{x}^T Q \vec{x} + \vec{T}^T R \vec{T}) dt \tag{1.25}$$

subject to some convex constraints such as: glide slope, thrust direction, thrust magnitude and safe height. However, this is not the actual cost function that is considered. In fact the MPC considers a discretized system of equations of motion described by Equation 1.26 and exploits them to evaluate a model prediction within an arbitrary chosen finite horizon N ($i = 0, ..., N-1$) as shown by Lee and Mesbahi in [15].

$$\vec{x}_{t,i+1} = A_t \vec{x}_{t,i} + B_t \vec{u}_{t,i} + C_t \tag{1.26}$$

In Equation 1.26 t is the time instant at which the system is sampled and the matrices containing the system properties A_t, B_t and C_t (gravity term) are assumed to be constant during the time interval $[t, t+1]$. So at each considered time step, the system is discretized and the future states are predicted thanks

to Equation 1.26. Then the predicted states are used to build the following cost function:

$$C_t = (\sum_{i=0}^{N-1} q(\vec{x}_{t,i}) + r(\vec{u}_{t,i})) + p(\vec{x}_{t,N}) \tag{1.27}$$

where $q = \vec{x}_{t,i}^T Q \vec{x}_{t,i}$, $r = \vec{u}_{t,i}^T R \vec{u}_{t,i}$ and $p = \vec{x}_{t,N}^T Q \vec{x}_{t,N}$ (Q, R and P are the weighting matrices on state, control and final state). Finally the needed control sequence $U = [\vec{u}_{t,0}, ..., \vec{u}_{t,N}]$ that minimizes the cost function of Equation 1.27 is obtained thanks to a convex programming solver. At this point the first control action of the obtained sequence ($\vec{u}_{t,0}$) is applied to the system to obtain the state at the next time step where the whole process is repeated again until the end of the trajectory [15]. A final important aspect to be considered is to verify at each step that the finite horizon does not exceed the optimal time-to-go, that can be evaluated thanks to this equation:

$$\vec{g}^T \vec{g} t_{go}^4 - 4\vec{v}^T \vec{v} t_{go}^2 - 24\vec{r}^T \vec{v} t_{go} - 36\vec{r}^T \vec{r} = 0 \tag{1.28}$$

In the case that the finite horizon results higher with respect to the time-to-go, then it has to be reduced until it becomes smaller than the t_{go}.
This method is very interesting since it is able to compensate uncertainties in a very effective way and is capable to consider varying system model and constraints at each time step. In fact this feature is something that can be fundamental in order to reach a very high degree of precision, making this method really fit for the considered kind of landing. However, some drawbacks are present such as the difficulty in considering non-convex constraints and the time consuming optimization of the cost function.

1.2 Guidance choice

Having in mind all these different types of guidance laws, each with their advantages and disadvantages, the first step was to choose whether to have a trajectory-tracking or trajectory-free kind of guidance. On one hand, as it is possible to understand from the above analysis, the trajectory-tracking guidance laws result to be quite stable and reliable, as some of them have already been applied in real applications such as for the case of MSL, but in general with poor flexibility. On the other hand, trajectory-free guidance laws are more flexible, hence they are able to better counteract disturbances step by step thanks to their nature. However, they are not always able to respect particular constraints. For these reasons at the end the final choice was to select a trajectory-free guidance, considering the overall flexibility as a fundamental aspect especially thinking about the final aim of achieving an autonomous pinpoint landing in the consid-

ered kind of landing environment. In fact, having an high flexibility enables to counteract in a better way possible disturbances, errors or uncertainties that have a crucial impact on the final landing precision during the landing phase on a low gravity body without atmosphere.
Between all the presented guidance laws of this kind, the MPC was the selected one. This selection was done because, as already described, this method presents a very high degree of flexibility thanks to its nature of predicting and optimizing the control action and the consequent trajectory at each step with the possibility of tuning different parameters such as the finite horizon time and the discretization time step. In addition, in order to enhance its performances, a machine learning approach was introduced in order to be able to use any kind of constraints, including the non-convex ones, and especially in order to reduce the time needed for the optimization of the cost function at each step. The idea is in fact to let a neural network learning how to perform this optimization in such a way to overcome the problem of the relatively long time time of computation making this approach more attractive for real applications.
In order to develop and test this new guidance method, a similar situation in terms of environment and initial state of the Philae lander of the previously cited Rosetta mission was considered [5], [16]. This choice was done in order to have a relatively simple environment without atmosphere and with low gravity in order to focus as much as possible on the development of the method itself without introducing immediately too much complications that could lead to problems not directly related to the actual guidance method.

1.3 Thesis organization

This thesis is organized as follows. In chapter 2, the overall method is presented and described. In this chapter also the dynamics considered for the development of the algorithm and the relative equations of motion are presented. In chapter 3, the part relative to the machine learning is analysed in detail, together with the way selected for its implementation. In chapter 4, after the presentation of the environment used to test and train the guidance, the obtained results are presented and discussed. In chapter 5, the conclusions and future possible developments are reported.

Chapter 2

Method Description

As already anticipated in section 1.2 the final method analysed by this thesis is the application of MPC combined with the use of machine learning.
A general architecture of the method is presented in Figure 2.1. As it is possible

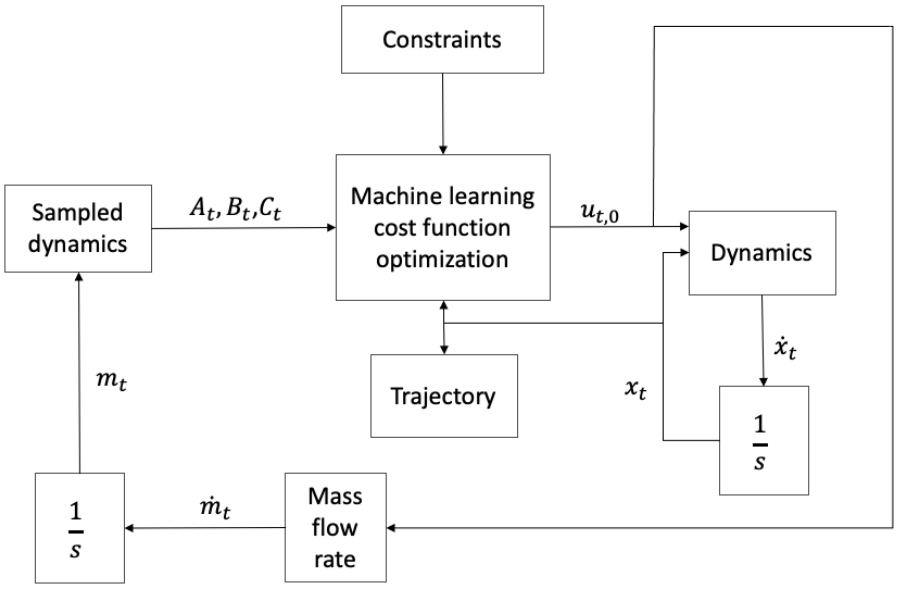


Figure 2.1. MPC+RL guidance architecture block scheme

to see this new guidance basically consists in all the passages of MPC already presented in section 1.1.1, with the addition of the cost function optimization obtained through machine learning techniques. The sampled dynamics block is where the dynamics is sampled at each time instant in order to get the matrix and vectors A_t, B_t and C_t that define the sampled dynamics equations of motion as it will be described later in subsection 2.1.1. These features, together with the current state and the constraints, are used by the machine learning process, defined by the machine learning cost function optimization block, to output the

needed control action $u_{t,0}$, which is the first action of the sequence that optimizes the MPC cost function determined by a prediction of the states inside a certain finite horizon defined by the user. This control action is then used to obtain the actual next state, thanks to the dynamics and integration blocks, and to define the mass at the following time instant, thanks to the mass flow rate definition identified by the homonym block and its integration. Finally the succession of all the states builds the overall landing trajectory expressed by the block of the same name. An important aspect to be underlined is that, in contrast to the classic MPC approach, in this case also the system constraints and boundary conditions are enforced thanks to a machine learning kind of approach. In this way the renewed obtained guidance maintains all the advantages of the original MPC with an enhanced ability in considering different kind of constraints and an important reduction in computational time regarding the cost function minimization that is needed to be performed at each time step in real time.

2.1 Dynamics

The dynamics taken in consideration for the development of the method is a 3D dynamics. This dynamics considers the lander as a point mass subjected to the low gravity of the considered landing body and to the thrust of each engine, with a variable mass in time depending on the chosen engine features. The presence of atmosphere is not considered.

2.1.1 Equations of motion

This kind of dynamics is ruled by the following equations:

$$\dot{\vec{r}} = \vec{v} \tag{2.1}$$

$$m\dot{\vec{v}} = \vec{T} + m\vec{g} \tag{2.2}$$

$$\dot{m} = -\frac{|\vec{T}|}{I_s g_0} \tag{2.3}$$

where $\vec{r}$ and $\vec{v}$ are the position and velocity vectors, that constitute the state $\vec{x}(t) = [\vec{v}(t)^T, \vec{r}(t)^T]^T$, m is the mass of the lander, $\vec{T}$ is the overall thrusting force vector, $\vec{g}$ is the gravitational acceleration vector, I_s the engine specific impulse and g_0 the Earth's gravity absolute value. The constraints are considered on the

initial and final state as follows:

$$\vec{x}(0) = \vec{x}_0 \tag{2.4}$$

$$\vec{x}(t_f) = \vec{0} \tag{2.5}$$

stating that the wanted final velocity and position are null, considering the target landing position as the origin of the reference system at the beginning of the powered descent phase (target-centred reference system). An additional constraint is the one that considers the impossibility of having additional thrust once the propellant is totally consumed, expressed by the following statement:

$$if \quad m(t) \leq 0.5m_0 \rightarrow \vec{T} = \vec{0} \tag{2.6}$$

considering the assumption of having the propellant mass as half of the initial mass.
However, as already revealed in section 1.1.1, the dynamics needed for the MPC guidance is a sampled dynamics, so it is possible to discretize Equation 2.1, Equation 2.2 and Equation 2.3 with a sampling period of Δt as:

$$\vec{r}(t+1) = \vec{r}(t) + \Delta t\vec{v}(t+1) \tag{2.7}$$

$$m\vec{v}(t+1) = m\vec{v}(t) + \Delta t\vec{T}(t) + \Delta tm\vec{g} \tag{2.8}$$

$$m(t+1) = m(t) - \frac{|\vec{T}|}{I_s g_0}\Delta t \tag{2.9}$$

In this way, the initial set of non-linear equations can be rewritten as a set of linear equations:

$$M(t)\vec{x}(t+1) = A_1(t)\vec{x}(t) + B_1(t)\vec{u}(t) + C_1(t)\vec{g} \tag{2.10}$$

$(\vec{u}(t) = \vec{T}(t))$ with

$$A_1(t) = \begin{bmatrix} mI_3 & 0_{3x3} \\ 0_{3x3} & I_3 \end{bmatrix} \quad B_1(t) = \begin{bmatrix} \Delta tI_3 \\ 0_{3x3} \end{bmatrix} \tag{2.11}$$

$$C_1(t) = \begin{bmatrix} \Delta tmI_3 \\ 0_{3x3} \end{bmatrix} \quad M(t) = \begin{bmatrix} mI_3 & 0_{3x3} \\ -\Delta tI_3 & I_3 \end{bmatrix} \tag{2.12}$$

The linear set of equations of motion can be further simplified in this way:

$$\vec{x}(t+1) = A(t)\vec{x}(t) + B(t)\vec{u}(t) + C(t)\vec{g} \tag{2.13}$$

with

$$A(t) = \begin{bmatrix} I_3 & 0_{3x3} \\ \Delta t I_3 & I_3 \end{bmatrix} \quad B(t) = \begin{bmatrix} \frac{\Delta t}{m(t)} I_3 \\ \frac{\Delta t^2}{m(t)} I_3 \end{bmatrix} \quad C(t) = \begin{bmatrix} \Delta t I_3 \\ \Delta t^2 I_3 \end{bmatrix} \tag{2.14}$$

This set of linear equations is simpler to be treated and enables to do faster calculations, which is important especially considering that at each time step it is needed to perform a prediction up to the considered finite horizon.
Another important aspect regards the considered thrusters' configuration that gives the lander the overall $\vec{T}(t)$ that interacts with the dynamics. The considered configuration is the one reported in Figure 2.2. As it is possible to

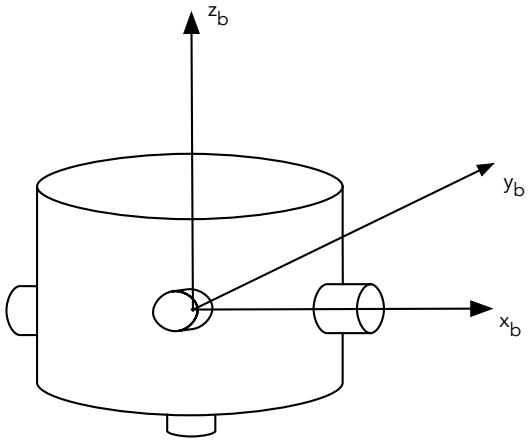
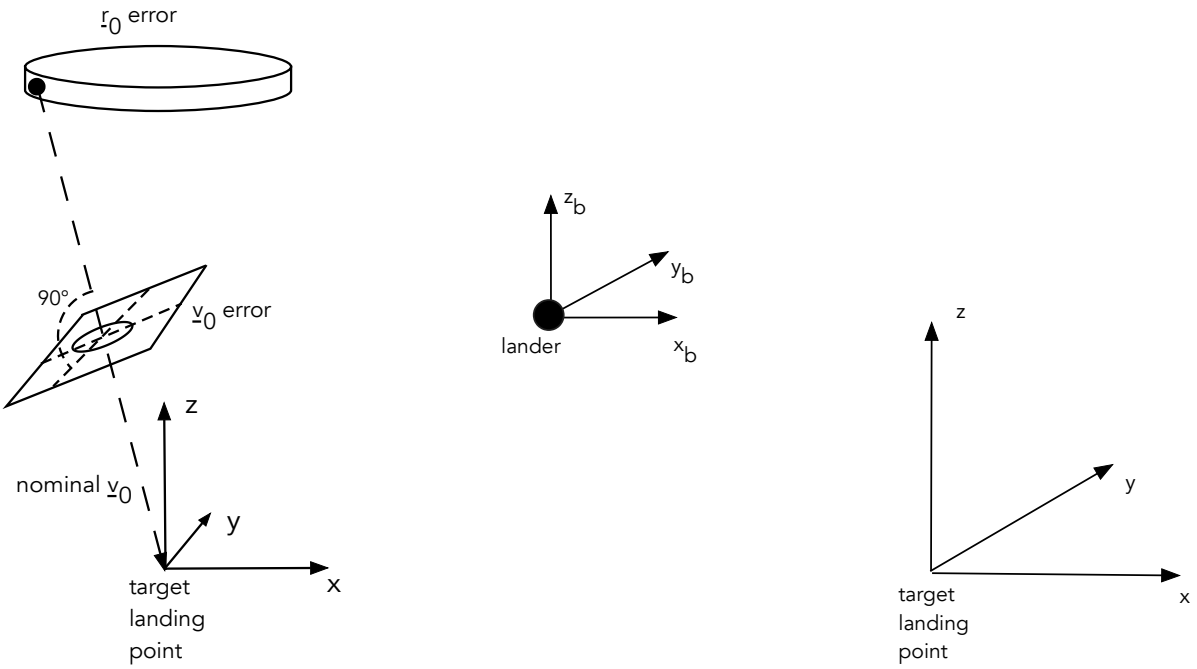


Figure 2.2. Lander thrusters configuration in body centred reference frame

see, the lander is considered to have 5 thrusters in such a way to be able to have thrust in any direction regarding the horizontal xy plane while for the vertical z direction it is only considered the possibility of having a positive thrust.

2.1.2 Initial state considerations

Another important aspect is the one relative to the initial state. In fact the errors on the initial state are considered as described by Figure 2.3a. The possible error on the initial horizontal position is considered in terms of radial distance from the landing point on the xy plane while the error on z direction is bounded between a maximum and a minimum value. Regarding the velocity, the nominal velocity is considered as always directed towards the target point. In this way it is possible to exploit the system symmetry and identify the x-axis



(a) Initial state errors

(b) Lander orientation

Figure 2.3. Lander initial state errors and orientation

of the reference system as always having the same direction of a general $\vec{a}$ vector that starts from the initial lander position and points the target landing point projection on the horizontal plane as showed in Figure 2.4.

In this way the initial horizontal position error can always be identified as along only the such described x-axis. Then, an uncertainty in terms of both direction and magnitude is applied to the initial velocity. Concerning the direction it is introduced an uncertainty in terms of the initial flight path angle γ_0 and initial heading angle β_0 while for the magnitude a maximum and minimum value are considered. This makes sense with the assumption of being in a similar environment of the Rosetta mission, where Philae is released from the orbiter directly into the powered descent phase. For this reason the assumption that errors on the initial position can be considered as an error in terms of radial distance for what concerns the horizontal position can be acceptable.

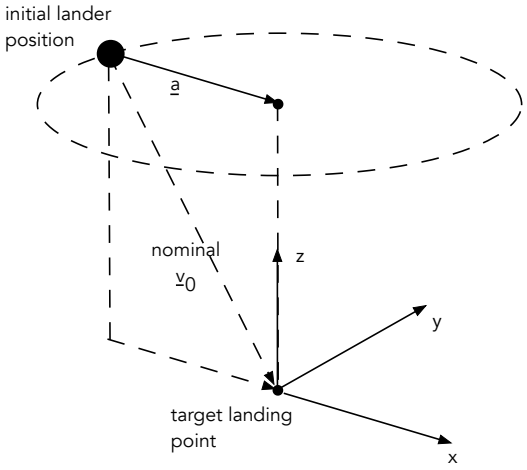


Figure 2.4. Reference system orientation

Furthermore, at the beginning of the powered descent phase, the lander is always considered to be with its x-axis aligned with the x-axis of the landing target-centred reference frame as showed in Figure 2.3b. In this way a couple of thrusters is always assumed to be oriented towards the landing position on the xy plane, letting the other couple of horizontal thrusters to work only in case of errors in the direction of the initial velocity vector.
At the beginning of each trajectory the radial direction becomes the x-axis previously discussed and for this reason, as just described, the couple of thrusters in y direction is used only in case that the initial velocity vector has an uncertainty on the β_0 angle.

2.2 Machine Learning general features

Concerning the machine learning part in general there are three kinds of tasks that can be considered [17]:

- Supervised Learning is the task of inferring a classification or regression from labeled training data.
- Unsupervised Learning is the task of drawing inferences from data sets consisting of input data without labeled responses.
- Reinforcement Learning (RL) is the task of learning how agents ought to take sequences of actions in an environment in order to maximize cumulative rewards.

In order to achieve the task needed by the guidance presented above, the RL approach results to be the best one. In fact this is the kind of machine learning technique that is used in many other different works concerning guidance laws such as the ones presented in [6] and [13]. The main idea behind RL is that an artificial agent may learn by interacting with its environment, similarly to a biological agent. Using the experience gathered, the artificial agent should be able to optimize some objectives given in the form of cumulative rewards. This approach applies in principle to any type of sequential decision-making problem relying on past experience. The environment may be stochastic, the agent may only observe partial information about the current state, the observations may be high-dimensional, the agent may freely gather experience in the environment or, on the contrary, the data may be constrained. So the overall idea in the case of the considered guidance law is to obtain an agent that learns what action it has to make, depending on the lander state, in order to minimize the MPC cost function while respecting all the given constraints and boundary conditions. The function that relates the observation given to the agent to the actual actions that

the agent takes is known as the policy. This policy can be any kind of function that depends on certain parameters that have to be learnt or, in general, it can be related to as a neural network.
A deep neural network is characterized by a succession of multiple processing layers. Each layer consists in a non-linear transformation and the sequence of these transformations leads to learning different levels of abstraction. An example is shown in Figure 2.5, where a simple neural network with one fully connected hidden layer is represented. The first layer is basically composed by the input values x in the form of a column vector of size n_x ($n_x \in N$). The values

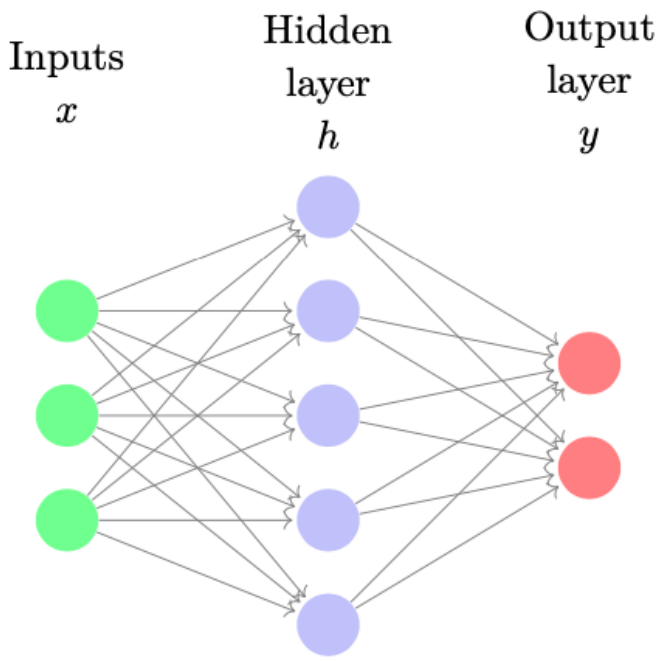


Figure 2.5. Fully Connected Neural Network example with one hidden layer

of the next hidden layer are a transformation of these values by a non-linear parametric function, which is a matrix multiplication by W_1 of size $n_h \times n_x$ ($n_h \in N$), plus a bias term of size n_h, followed by a non-linear transformation:

$$h = A(W_1 \cdot x + b_1) \tag{2.15}$$

where A is the activation function. This non-linear activation function is what makes the transformation at each layer non-linear, which ultimately provides the expressivity of the neural network. Then the output layer y can be expressed in a similar manner as the hidden one. It is important to highlight that in the presented example only fully connected feed-forward layers are considered, which means that each neuron of a layer is considered in the creation of each neuron of the following one through Equation 2.15. In general, other kinds of layers

can be considered such as convolutional or recurrent layers as described in [17]. However, these types of layers are better suited for other kinds of problems like images and sequential data. For this reason fully connected feed-forward layers are the only ones that have been considered throughout the development of the guidance algorithm.

Chapter 3

Reinforcement Learning

The general RL scheme is reported in Figure 3.1. [18]. Formally, the RL problem

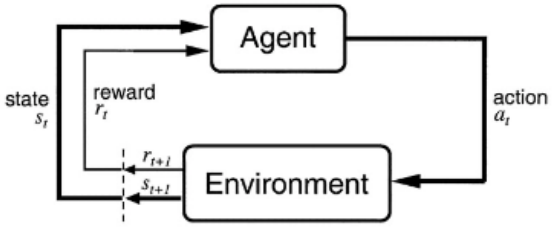


Figure 3.1. Reinforcement Learning scheme: Interaction between an agent and its environment

can be described as a Markov Decision Process (MDP). At time t, the agent is in state s_t and decides to perform an action a_t by obeying the policy $\pi(a_t|s_t)$. At the next time step, it arrives in the state s_{t+1} and obtains the reward r_{t+1}. The environment may be fully or partially observable. Future rewards are discounted by γ^k where $\gamma \in [0, 1]$ and k is the future time-step. Horizon H is the number of time-steps, T, needed to complete one episode from s_0 to s_T. The goal of the agent is to maximize the reward obtained on the long term by learning a policy π^* that will decide which action to take given a state:

$$\pi^* = argmax_\pi R_t \tag{3.1}$$

where $R_t = \sum_{k=0}^{T} \gamma^k r_{t+k}$ is known as the return. In general it is possible to distinguish between two kind of policies:

- stochastic policy π: determines the probability distribution $P(A)$ of performing an action.
- deterministic policy $\mu(s_t)$: is a discrete mapping from states to actions $(S \to A)$.

where S and A are respectively the state and action spaces. From Equation 3.1 it is possible to notice that future rewards have lower weights when compared to the immediate rewards since generally $\gamma^k < 1$. At the extremes, when $\gamma = 0$, only the immediate reward matters while when $\gamma = 1$ future rewards have the same weight as the immediate ones. In general the return can be interpreted as a measure of the value of a given state by following an arbitrary policy π:

$$V^{\pi}(s_t) = E_{\pi}[R_t|s_t] \tag{3.2}$$

where $E[x]$ is the expected value function. In other words, the value of a state s is the expected discounted return received if the agent starts in s and then follows its policy π. In this sense the goal of the agent is to learn the optimal policy that maximizes V^{π} for all states s:

$$\pi^* = argmax_{\pi} V^{\pi}(s) \tag{3.3}$$

The sequence of events leading to the terminal state can be modeled as the trajectory of the policy:

$$\tau = (s_0, a_0, r_1, ..., s_{T-1}, a_{T-1}, r_T, s_T) \tag{3.4}$$

If the MDP is episodic, when the agent reaches the terminal state, s_T, the state is reset to s_0. If T is finite, we have a finite horizon, otherwise the horizon is infinite.
In general it is possible to distinguish between two methods by which the agent is able to learn by interacting with the environment: Q-Learning and Policy Gradient.

3.1 Q-learning

Concerning the first one it is necessary to firstly define the Q-value function, that is the mathematical expectation of the return over all the trajectories starting from a state-action pair (s,a) defined by the policy π:

$$Q^{\pi}(s_t, a_t) = E_{\pi}[R_t|s_t, a_t] \tag{3.5}$$

The Q-value and value function are linked by the following relation:

$$V^{\pi}(s) = \sum_{a \in A} \pi(s, a) Q^{\pi}(s, a) \tag{3.6}$$

At this point, instead of finding the policy that maximizes the value for all states as done by Equation 3.3, the aim of Q-Learning is to find the action that maximizes the Q-value for all the states:

$$\pi^* = argmax_a Q(s, a) \tag{3.7}$$

In practice, the Q-value is learnt thanks to the following iterative algorithm known as the Bellman Equation [19]:

$$Q(s, a) = E_{s'}[r + \gamma max_{a'} Q(s', a')] \tag{3.8}$$

Q-Learning attempts to approximate the first-order expansion of the value expressed by Equation 3.2 as a function of both current state and action. Q-Learning is an off-policy RL algorithm. In fact it learns to improve the policy by not directly sampling experiences from that policy. In other words, the Q values are learned independently of the underlying policy being used by the agent. Once the Q-value function has converged, then the optimal policy is found thanks to Equation 3.7. It is important to notice that the agent must continually explore its environment while gradually taking advantage of what it has learned so far. Finding the right balance between Exploration and Exploitation is one of the main issues in RL. Generally, during the start of the learning process, the action is mainly random (exploration). As the learning progresses, the agent takes advantage of the Q value (exploitation). For instance, it could be that at the start 90% of the action is random and 10% from Q value function, while by the end of each episode the action becomes 10% random and 90% from Q value function.
Despite being very promising, the Q-Learning method presents some important limitations such as the fact that it is not able to deal with continuous action space environments and, as described above, it is not directly optimizing the policy along the learning process.

3.2 Policy Gradient

Policy gradient methods are in contrast applicable to environments with both discrete or continuous action spaces and are also directly optimizing the policy along the learning process. For these reasons at the end only policy gradient RL methods have been considered for the development of the guidance, giving particular attention to three types that are described in this section.
As for the Q-Learning, also for the Policy Gradient method the goal of the agent is to learn an optimal policy π^* that maximizes the return from all the states (Equation 3.1). However, now the aim is to directly learn the policy by a policy

parameterization $\pi(a_t|s_t) \rightarrow \pi(a_t|s_t, \theta)$, where θ indicates the parameters. In this way it is also possible to use a neural network to learn the policy. The learning process itself is done through an objective function $J(\theta)$ maximization, being $J(\theta)$ a performance measure with respect to the parameters θ. The function maximization is achieved by performing gradient ascent, which means that the gradient update is in the direction of the derivative of the function being optimized.
Concerning discrete action spaces the policy can be represented by the following equation:

$$\pi(a_i|s_t, \theta) = softmax(a_i) \quad for \quad a_i \in A \tag{3.9}$$

where a_i is the i-th action that can be the prediction of a neural network or a general function of the state. In this case $\pi(a_i|s_t, \theta)$ determines the probability of each a_i. Then the agent takes the action with the highest probability, $a_t = max_i \pi(a_i s_t, \theta)$.
For continuous action spaces, $\pi(a_i|s_t, \theta)$ samples an action from a probability distribution given the state. In general it is usually a Gaussian distribution whose mean μ and standard deviation σ are predicted by the policy network or are general functions of the states, each with their own parameters θ_μ and θ_σ. The predicted action is a sample from this Gaussian distribution (Equation 3.10). To ensure that no invalid prediction is generated, the action is clipped between its maximum and minimum values depending on the action space A.

$$\pi(a_i|s_t, \theta) = a_t \sim N(\mu(s_t), \sigma(s_t)) \tag{3.10}$$

Training a policy network is therefore a matter of optimizing the parameters $\theta = [\theta_\mu, \theta_\sigma]$.
Given a continuously differentiable policy function, the policy gradient can be computed as:

$$\nabla J(\theta) = E_\pi[\frac{\nabla_\theta \pi(a_i s_t, \theta)}{\pi(a_i s_t, \theta)} Q^\pi(s_t, a_t)] = E_\pi[\nabla_\theta ln \pi(a_i s_t, \theta) Q^\pi(s_t, a_t)] \tag{3.11}$$

where the property of the natural logarithm $\frac{\nabla x}{x} = \nabla lnx$ is exploited. As it can be noticed from Equation 3.11, the performance gradient is estimated from the target policy samples and it is proportional to the policy gradient. The policy gradient is scaled by the Q-value to encourage actions that positively contribute to the state value. The gradient is also inversely proportional to the action probability to penalize frequently occurring actions that do not contribute to the increase of performance measure.
In the following sections different policy gradient methods are presented, each of them with their own way of estimating the policy gradient.

3.2.1 Reinforce with Baseline

The Reinforce with Baseline (RB) method, based on the simpler Reinforce algorithm, is a method where the policy gradient is estimated in the following way:

$$\nabla J(\theta) = E_\pi[(R_t - B(s_t))\nabla_\theta ln\pi(a_t|s_t,\theta)] = E_\pi[R_t \nabla_\theta ln\pi(a_t|s_t,\theta)] \qquad (3.12)$$

where $\delta = R_t - B(s_t)$ is an unbiased sample of $Q^\pi(s_t, a_t)$ in the policy gradient theorem. $B(s_t)$ is a baseline that is subtracted to the return and requires its own network to be determined. In most of the cases, the value function is used as the baseline $B(s_t) = V(s_t)$. As shown by Figure 3.2, the parametrized policy and value functions can be modeled by neural networks.

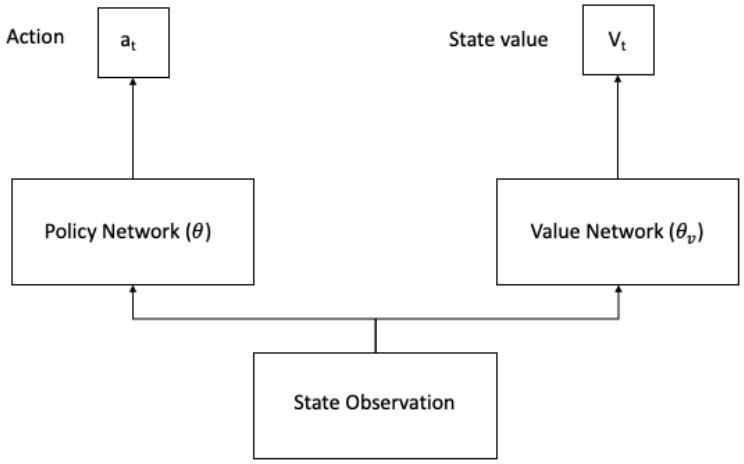


Figure 3.2. Reinforce with Baseline scheme

algorithm 1 reports the Reinforce with Baseline algorithm. Only experience samples are needed to optimally tune the parameters of the policy and value networks. The discount factor γ takes into consideration that rewards decrease in value as the number of steps increases. Both the gradients are discounted by γ^t and for this reason gradients taken at later steps have smaller contributions, since in general $\gamma \in [0, 1]$. The two learning rates α and α_v are scaling factors of the two gradient updates. The parameters are updated by performing gradient ascent using the discounted gradients and learning rates. This method requires that the agent completes an episode before processing the gradient updates and for this reason in general the gradient update is characterized by high variance. However, the presence of the baseline reduces this variance and does not affect the expectation of the performance gradient since it is not a function of the actions, as it is possible to see from Equation 3.12. If the return is overestimated,

the scaling factor δ is proportionally reduced by the value function resulting to a lower variance. This reduction of variance accelerates the learning process especially if compared to the already cited Reinforce algorithm, which is actually the very same algorithm but with the absence of the baseline function [20].

Algorithm 1: Reinforce with Baseline algorithm

Repeat:

Generate an episode $(s_0, a_0, r_1, ..., s_{T-1}, a_{T-1}, r_T, s_T)$ by following $\pi(a_t|s_t, \theta)$

for *steps t=0,...,T-1* **do**

- Compute Return, $R_t = \sum_{k=0}^{T} \gamma^k r_{t+k}$
- Subtract Baseline, $\delta = R_t - V(s_t, \theta_v)$
- Compute discounted value gradient, $\nabla V(\theta_v) = \gamma^t \delta \nabla_{\theta_v} V(s_t, \theta_v)$
- Perform gradient ascent, $\theta_v = \theta_v + \alpha_v \nabla V(\theta_v)$
- Compute discounted performance gradient, $\nabla J(\theta) = \gamma^t \delta \nabla_\theta ln\pi(a_t|s_t, \theta)$
- Perform gradient ascent, $\theta = \theta + \alpha \nabla J(\theta)$

end

3.2.2 Actor Critic

The Actor-Critic (AC) is a variation of the RB method where the policy and value networks showed by Figure 3.3, play the roles of actor and critic networks. The policy network is the actor that decides which action to take given the state while the value network acts as a critic which quantifies how good or bad the chosen action made by the actor is. This last network evaluates the state value by comparing it with the received reward and the discounted value of the observed next state as expressed by δ:

$$\delta = r_{t+1} + \gamma V(s_{t+1}, \theta_v) - V(s_t, \theta_v) \tag{3.13}$$

Since estimating distant future rewards is difficult, the used estimate is based only on the immediate future. This technique, known as bootstrapping, often accelerates learning and reduces variance. As it is possible to see from algorithm 2, at every step both networks are trained unlikely to what is done in the RB case, where the agent completes an episode before the training is performed. The value network is consulted twice. Firstly, during the value estimate of the current state and secondly for the value of the next state. Both values are used in the computation of gradients.

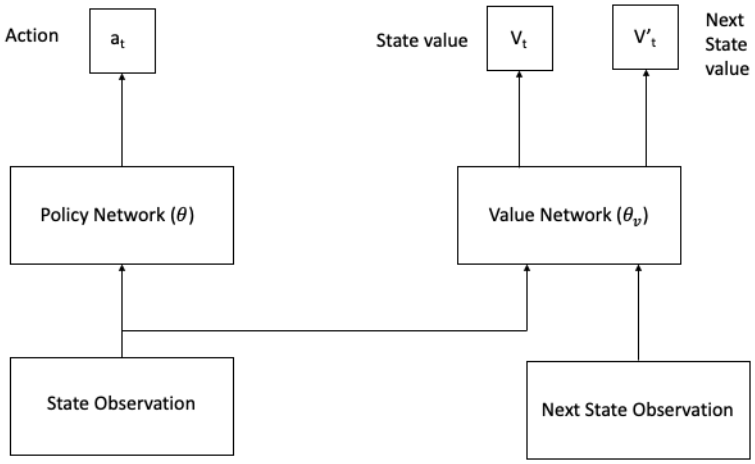


Figure 3.3. Actor-Critic scheme

Algorithm 2: Actor-Critic algorithm

Repeat:

for *steps t=0,...,T-1* **do**

Sample an action, $a \sim \pi(a|s, \theta)$

Execute the action and observe reward r and next state s'

Execute state value estimate, $\delta = r + \gamma V(s', \theta_v) - V(s, \theta_v)$

Compute discounted value gradient, $\nabla V(\theta_v) = \gamma^t \delta \nabla_{\theta_v} V(s_t, \theta_v)$

Perform gradient ascent, $\theta_v = \theta_v + \alpha_v \nabla V(\theta_v)$

Compute discounted performance gradient, $\nabla J(\theta) = \gamma^t \delta \nabla_\theta ln\pi(a_t|s_t, \theta)$

Perform gradient ascent, $\theta = \theta + \alpha \nabla J(\theta)$

Update state, $s = s'$

end

3.2.3 Advantage Actor-Critic

The Advantage Actor-Critic (A2C) method is a variation of the AC one where the value gradient is equal to the partial derivative of the mean squared error between the return, R_t , and the state value:

$$\nabla V(\theta_v) = \frac{\delta(R_t - V(s, \theta_v))^2}{\delta \theta_v} \tag{3.14}$$

where the quantity $(R_t - V(s, \theta_v))$ is called Advantage and as it tends to zero, the value network prediction gets more accurate. The corresponding network for A2C is similar to Figure 3.3 since only the method of gradient computation has changed.
As it is possible to see from algorithm 3, there are also other differences with respect to the AC procedure. In fact A2C, similarly to RB, is trained after one episode has been completed. However, the learning process starts from the last state and ends on the first one. Moreover, the A2C policy and value gradients are no longer discounted by γ^t and the gradient of the weighted entropy value of the policy function, $\beta \nabla_\theta H(\pi(a_t|s_t, \theta))$, where β is the entropy weight, is added to the gradient function [21].

Algorithm 3: Advantage Actor-Critic algorithm

Repeat:

Generate an episode $(s_0, a_0, r_1, ..., s_{T-1}, a_{T-1}, r_T, s_T)$ by following $\pi(a_t|s_t, \theta)$

$$R_t = \begin{bmatrix} 0 & if\ s_t\ is\ terminal \\ V(s_t, \theta) & for\ non-terminal\ s_t,\ bootstrap\ from\ last\ state \end{bmatrix}$$

for *steps t=T-1,...,0* **do**

Compute return, $R_t = r_t + \gamma R_t$

Compute value gradient, $\nabla V(\theta_v) = \frac{\delta(R_t - V(s,\theta_v))^2}{\delta \theta_v}$

Accumulate gradient, $\theta_v = \theta_v + \alpha_v \nabla V(\theta_v)$

Compute performance gradient,

$\nabla J(\theta) = \nabla_\theta ln \pi(a_t|s_t, \theta)(R_t - V(s, \theta_v)) + \beta \nabla_\theta H(\pi(a_t|s_t, \theta))$

Perform gradient ascent, $\theta = \theta + \alpha \nabla J(\theta)$

end

All the algorithms described above are implemented and tested, as it will be shown in chapter 4, in order to compare the different performances achieved by each of them and eventually chose the one that produces the best results.

3.3 Environment Description

As already said and showed by Figure 3.1, the environment is what interacts with the agent, taking as input the actions and giving as output the reward and the next state. For this reason building a good environment that well represents the dynamics of the system with a reward that correctly respects the objectives of the learning process is fundamental to obtain good results. In this section

the environment used for the development of the guidance is presented with its main features.
At first, some of the already working relatively simple environments, such as the continuous mountain car or the continuous cart-pole presented in [22], were analysed in order to have a better idea on how to proceed in the development of the applied environment. In this way it was possible to understand that one fundamental aspect, as also expressed in [23], is to have a reward designed in such a way that it leads the algorithm faster to more promising solutions by incorporating domain knowledge into Reinforcement Learning through reward shaping. In addition, it was possible to observe that the reward is in general bounded between a maximum and minimum value, usually between $[-1, 1]$, in order to avoid excessive large gradients during the learning process.
Another important aspect is the one relative to the terminal condition of an episode. In fact in the environment it is needed to define when to terminate an episode. In general there are three kinds of terminal conditions:

- Time limits, maximum number of iterations exceeded
- Positive terminal, the system reaches a final state that is inside the desired boundaries
- Negative terminal, the system reaches a final state that is outside the desired boundaries

A final observation can be done relatively to the meaning of the numerical value of the reward obtained step by step that can be usually distinguished between positive and negative rewards. In fact, in general, a positive reward encourages the system to keep going to accumulate more rewards and avoid terminals unless they yield a very high reward. On the contrary, a negative reward normally encourages the system to reach the terminal state as quick as possible in order to avoid accumulating penalties.
Having all these considerations in mind the environment was built as follows. The actions given by the agent are used to build the temporal history of each control action prediction $u_{t,x}(t)$, $u_{t,y}(t)$ and $u_{t,z}(t)$ thanks to a cubic function interpolation until the decided finite horizon instant N. This finite horizon is compared at every step to the time-to-go t_{go} and reduced if bigger than this parameter ($N \leq t_{go}$). The action history is then sampled by a Δt defined by the user in order to be compatible with the sampled equations of motion described in subsection 2.1.1 that are used to predict the future states until the end of the finite horizon. Then all the predicted states are grouped together in a single vector $\vec{X} = [\vec{x}_{t,0}^T, ..., \vec{x}_{t,N}^T]^T$. The same is done for the sampled control action $\vec{U} = [\vec{u}_{t,0}^T, ..., \vec{u}_{t,N}^T]^T$, where $\vec{u}_{t,i} = [u_{t,x}(i), u_{t,y}(i), u_{t,z}(i)]^T$. These vectors are then

used to build the cost function of the MPC process that has to be minimized:

$$J = w_x \vec{X}^T \vec{X} + w_u \vec{U}^T \vec{U} \tag{3.15}$$

where w_x and w_u are the weights for the states and control actions. In order to have comparable values of J along the trajectory, each component of the X and U vectors is normalized as follows:

$$a_{norm} = \frac{a - a_{min}}{a_{max} - a_{min}} \tag{3.16}$$

where a represents a general vector's component. Furthermore, to have a value of J independent from the size of the X and U vectors, each of them is divided by its length in the construction of the cost function:

$$J = w_x \frac{\vec{X}^T \vec{X}}{length(\vec{X})} + w_u \frac{\vec{U}^T \vec{U}}{length(\vec{U})} \tag{3.17}$$

At this point it is possible to formulate the reward. This is actually the most important part of the environment definition since it is what defines the quality of the actions applied by the agent, hence it deeply influences the whole learning process. The reward is chosen to be a shaped reward since it makes the learning process more efficient with respect to a reward where values are given in a discontinuous way. At first the idea was to construct a reward dependent only on the value of J. Since the objective of the learning process is to minimize the value of the cost function, the reward was built in order to give increasing rewards the more the value of J was reduced. Two possible reward functions were built in this sense:

- $$r = 1 - \frac{J}{J_{max}} \tag{3.18}$$

  where $r \in [0, 1]$

- $$r = k \tanh \frac{\Delta J}{\Delta J_{max}} \tag{3.19}$$

  where $\Delta J = J_{i-1} - J_i$, $\Delta J_{max} = J_{max} - J_i$ and $r \in [-k, k]$.

The maximum value of J is estimated by running a test episode and looking the values assumed by the cost function along the episode. In Equation 3.18, the reward gets closer to its maximum value the more the value of $J \to 0$, where 0 is the minimum achievable value of the cost function, since $J \geq 0$ by definition. In Equation 3.19 instead the reward depends on how much the cost function is reduced step by step along the trajectory of each episode, being $i = 1, ..., T$

where T is the last instant of the episode. So the main difference is that while Equation 3.18 rewards the reduction of J in a global way, Equation 3.19 does the same thing in a local manner along each instant of the episode. Unfortunately, once tested, both of these methods resulted to be not sufficiently satisfying since the final wanted landing conditions were hardly reached especially in terms of final horizontal landing position. For this reason it resulted necessary to introduce in the reward construction a part that was directly related to the position at each instant in order to lead the lander towards the wanted final state step by step. Eventually this idea resulted in the following reward construction:

$$r_{distance} = 1 - (\frac{d}{d_{max}})^{0.4} \tag{3.20}$$

$$r = J_{discount} r_{distance} \tag{3.21}$$

where $J_{discount} = 1 - \frac{J}{J_{max}}$, $d = |[r_x, r_y]^T|$ and d_{max} is in general the distance between the goal and the edge of the workspace on the horizontal plane. As it is possible to see, the reward is constituted by two main parts: one related to the horizontal position and one related to the cost function J. This two parts are built in such a way that at first a value of reward based only on the horizontal position ($\in [0, 1]$ if $d \in [0, d_{max}]$) is given with a magnitude that is higher the more the lander is close to the origin, as showed in Figure 3.4. Then this initial reward is discounted by a value $\in [0, 1]$ that is actually what already previously described by Equation 3.18 and whose shape is showed in Figure 3.5.

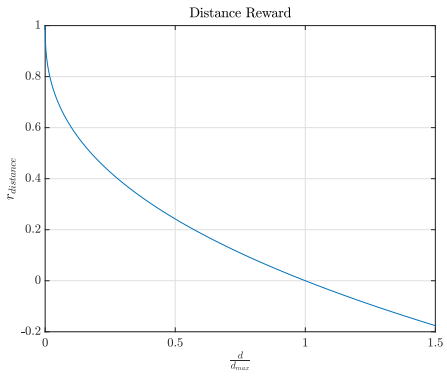


Figure 3.4. Distance Reward

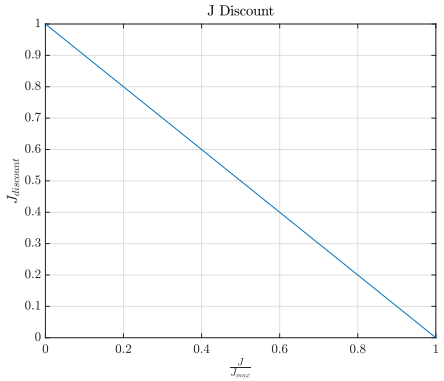


Figure 3.5. J Discount

This kind of reward is the one that actually worked better and brought some promising results as it will be showed in chapter 4. However, there is still another important aspect relative to the reward at the terminal states. In fact when the episode reaches its final condition the reward is given in a different manner with respect to what just described. The termination of an episode is defined by a boolean parameter called "Done" that is set to "True" once one of the following

conditions is reached:

- $v_z > 0$, the lander starts going away from the ground
- $r_z \leq 0$, the lander has reached ground

If at the end of the episode the lander has not reached ground or has violated some constraints, then the reward is set to a value that depends on which kind of constraint has been broken. On the contrary, if the lander has managed to land respecting all the imposed constraints, a positive final reward is given. Establishing the value of the final reward resulted to be actually part of the optimization process of the algorithm since the whole learning process resulted to be highly sensitive to these particular parameters.
A final important aspect relative to the environment is the one relative to its reset at the end of each episode. As said, for each step of the trajectory the cost function J is evaluated accordingly to the selected finite horizon and the reward is consequently evaluated. Then only the first predicted state is considered and set as the following state that is used by the agent to obtain the new action and repeat the whole process until the end of the episode. Once the episode is terminated the lander state and initial mass are resorted according to their uncertainty and error distribution, that concerning the state variables are the ones already described in subsection 2.1.2.
Having all these considerations in mind, in general it is possible to define three main functions that constitute the environment: the "initialization" function, the "step" function and the "reset" function. The first one, as its name says, has the role of initializing all the parameters that form the environment. The second one is the function that, given the current state and action as input, outputs the next state, the reward and the "Done" variable. The third function instead has the role of resetting the environment according to the considerations made above.

3.4 Algorithm Implementation

In this section the implementation of all the three algorithms presented in section 3.2 is described, since all of them have been tested for the development of the desired guidance. The software used is Keras, which is an Application Programming Interface (API) built on top of Tensorflow 2.0 in Python programming language.
The implementation was done by taking inspiration from the codes presented in [19], where the implementation of all the presented algorithms is developed for the continuous mountain car environment case. The general algorithm is structured as follows:

- Neural network construction
- Learning Algorithm implementation

3.4.1 Neural Network Construction

As already described in section 2.2 only fully connected feed-forward layers, implemented by using the *Dense* command in Keras, have been considered in the development of the applied neural networks. Since the continuous action space case has been considered, the policy network, as anticipated in section 3.2, has the role to estimate the $\mu(s_t)$ and $\sigma(s_t)$ needed to sample the action from a Gaussian distribution. As it is possible to see from Figure 3.6, the main network is formed by two hidden layers of 64 neurons each. The second hidden layer is then connected in parallel to the "mean" and "standard deviation" layers, each with a number of neurons equal to the wanted action dimension. This two layers are then connected to another layer that actually evaluates the action, called "action" layer. Then the "mean", "standard deviation" and "action" layers are connected to a final layer, the "logp" layer, that is used to evaluate the logarithmic probability of taking a certain action, which is fundamental for the definition of the loss needed by the learning process.

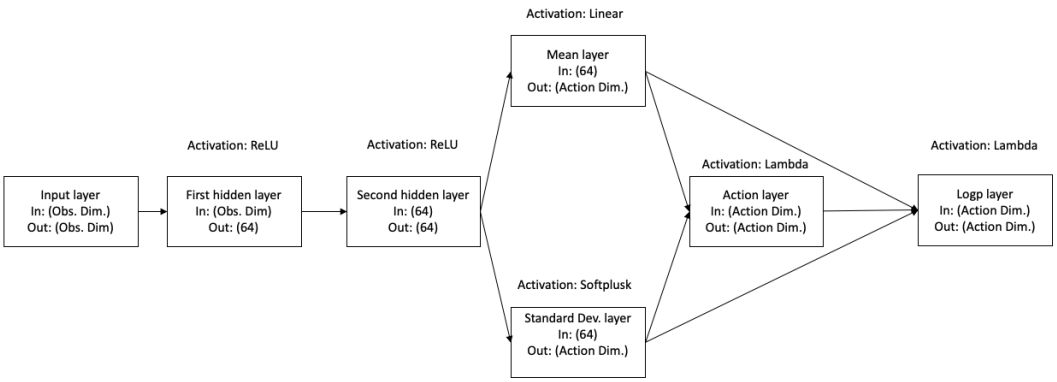


Figure 3.6. Logarithmic Probability Network Model

The activation function for the hidden layers is the "relu" (Rectified Linear Unit) one, which is typical for this kind of layers. It basically consists in taking the maximum value between 0 and the input ($a(x) = max(x, 0)$). The presence of these layers is very important since they are what in principle can actually allow the approximation of any kind of arbitrary function. In fact in the case of absence of this type of layers what is obtained as an output of the network is just a linear regression of the input variables. In general the hidden layers are two at maximum, with a number of neurons that has to be kept not too high

but nor too low depending on the considered problem. Usually this number is chosen to be a certain power of 2. The "mean" layer has a "linear" activation function ($a(x) = x$) while the "standard deviation" layer applies a "softplus" activation function which enables to have an output value always higher than zero ($a(x) = \ln(\exp x + 1)$). In order to avoid values too close to zero, which wouldn't have sense for the $\sigma(s_t)$ values, this activation function was modified as follows and called "softplusk": $softplusk(x) = softplus(x) + 10^{-10}$. The "action" layer applies a custom (Lambda) activation function that samples a Gaussian distribution $N(\mu(s_t), \sigma(s_t))$ using the values given by the "mean" and "standard deviation" layers. Then this action is clipped between its minimum and maximum values. The "logp" layer finally uses the outputs of the "mean", "standard deviation" and "action" layers to evaluate the logarithmic probability of the action using another custom activation function. This whole network just described actually is what builds the "logp" model that is what is actually trained during the learning process. In fact another model can be defined as the "actor" (or "policy") model, that is constituted by the very same network of the previous one with the absence of the last "logp" layer, as it is possible to see from Figure 3.7. Since these two models share the same parameters training the "logp" model means training also the "actor" one. However, as already said

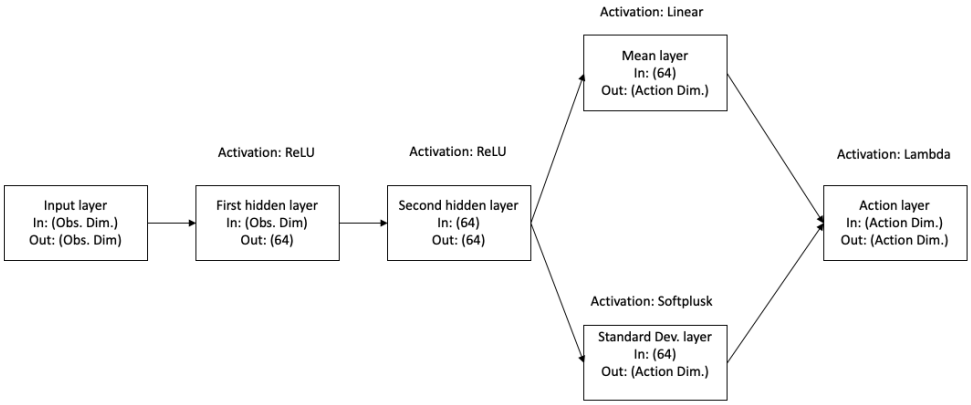


Figure 3.7. Actor Network Model

in section 3.2 there are also other networks that are needed for the learning process: the value network and the entropy network (used only by A2C). As it is possible to see from Figure 3.8, the entropy network is basically the same as the actor one with the only difference regarding the last layer, where a custom activation function is used to evaluate the entropy associated to each action. Concerning the value network, as shown by Figure 3.9, it is simply made by two hidden layers and a final layer with a "linear" activation function.

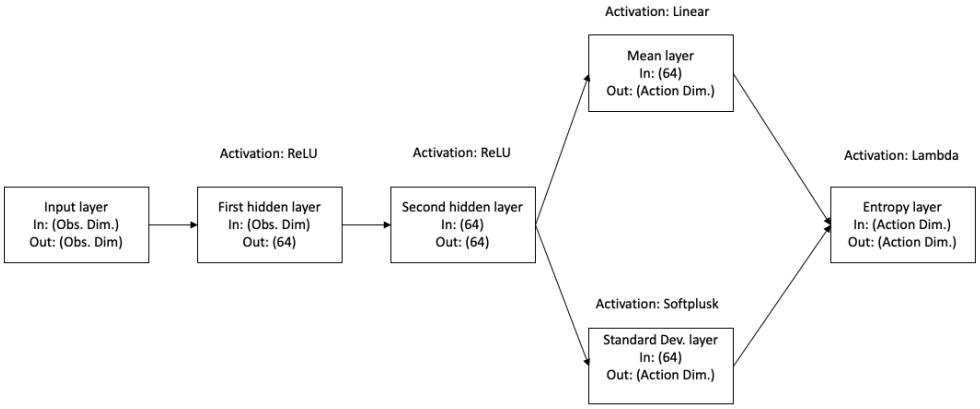


Figure 3.8. Entropy Network Model

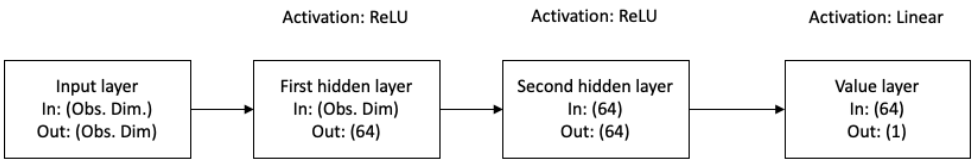


Figure 3.9. Value Network Model

A fundamental aspect to be considered while using these networks is the importance of scaling the inputs to avoid saturating the activation functions at each layer [13]. For this reason, in the development of the algorithm, the input of the network (the observation) was considered as the following one: $\vec{obs} = [v_x, v_y, v_z, t_{go}, r_z]^T$, where each component has been normalized according to the already described Equation 3.16. In addition, in order to have a more efficient learning process, it is also important to insure that the magnitude of the neural network outputs are are reasonably close to unity [13]. This is obtained thanks to the activation function of the "action" layer where the action values are clipped between $[-1, 1]$. Then this values are reinterpreted by the environment, where the value of 1 is associated to the maximum value of available thrust.
Once the networks are built, the next step is preparing the networks for the training. In the previously presented algorithms (section 3.2) it was performed an objective function maximization by gradient ascent in order to update the network's weights. Instead in Keras what it is usually done is to perform a loss function minimization by gradient descent. The loss function is simply the negative of the objective function being maximized while the gradient descent is

the negative of gradient ascent. In order to prepare the models to the learning it is needed to define the loss functions as well as the kind of optimizer used. In this sense for all the models involved the chosen optimizer was "Adam", which is a typical optimizer used in the field of Reinforcement Learning, since it is usually more robust with respect to the hyperparameters selection if compared to other methods. The hyperparameters are some of the parameters that influence the learning process like: the learning rate α, the number of hidden layers, the number of neurons for each hidden layer, the discount factor γ and the number of episodes.
Concerning the loss functions they are defined as follows:

- "logp" model:

$$loss_{logp} = -mean((y_{pred} * y_{true}) + (\beta * entropy)) \tag{3.22}$$

- "value" model

$$loss_{value} = -mean(y_{pred} * y_{true}) \tag{3.23}$$

where β is the entropy loss weight (usually equal to 0.9 for A2C, equal to 0 for RB and AC), y_{pred} is the common term of all the performance gradients reported in Table 3.1, directly evaluated thanks to the involved network, and y_{true} is a value that depends on the involved method as it possible to see from Table 3.2.

y_{pred} **of** $loss_{logp}$	y_{pred} **of** $loss_{value}$
$\nabla_\theta \ln \pi(a_t \mid s_t, \theta)$	$\nabla_{\theta_v} V(s_t, \theta_v)$

Table 3.1. y_{pred} of $loss_{logp}$ and $loss_{value}$

Algorithm	y_{true} **of** $loss_{logp}$	y_{true} **of** $loss_{value}$
Reinforce with Baseline	$\gamma^t \delta$	$\gamma^t \delta$
Actor-Critic	$\gamma^t \delta$	$\gamma^t \delta$
Advantage Actor-Critic	$(R_t - V(s, \theta_v))$	R_t

Table 3.2. y_{true} of $loss_{logp}$ and $loss_{value}$

3.4.2 Learning Algorithm implementation

With all the network models and loss functions in place, the last part is the training strategy, which is different for each algorithm. In general two train functions are used: "train by episode" and "train".
The first function, that is different for each algorithm, is applied only in the RB

and A2C cases. In fact these algorithms are the ones that need the end of the episode before starting the learning process. Once an episode is completed, the whole trajectory history (states, actions and rewards) is stored in the computer memory. Then this function has the role of analyzing the whole trajectory in order to obtain the y_{true} values for each step and call the "train" function where the parameters of both the "logp" and "value" models are actually updated thanks to the Keras fit function, that needs the current state and the y_{true} value as inputs. Since the AC algorithm directly trains step by step without waiting the end of the episode only the "train" function is needed in this case. Of course what makes the difference between the analysed methods is the evaluation of the y_{true} value and what states are used during the learning process. These features are selected according to the algorithms presented in section 3.2. In fact, for instance, a remarkable aspect is that the training strategy of A2C is different in the sense that it computes gradients from the last step to the first step. Hence, in the evaluation of y_{true} the return accumulates beginning from the last step reward or the last state value.
In the Keras implementation, all the mentioned routines are implemented as methods in the "PolicyAgent" class. The role of the "PolicyAgent" is to represent the agent implementing policy gradient methods including building and training the network models and predicting the action, logarithmic probability, entropy, and state value.
Finally a general main code is implemented. This code deals with all the presented functions of the "PolicyAgent" class and has the role of managing the whole learning process. At first the selected environment is loaded and initialized and the needed neural networks are created. Then a for loop is executed for the selected number of episodes. At the beginning of each episode the environment is restored by the "reset" function and a while loop is started until the end of the episode to evaluate all the trajectories. In the case of applying the AC algorithm, the weights are updated directly inside this loop by the "train" function. On the contrary, for the RB and A2C cases, the network weights are updated by the "train by episode" function once the while loop has terminated. Eventually at the end of the for loop the learning process is terminated.

Chapter 4

Results

4.1 Test Case Description

As already anticipated in section 1.2, the kind of environment selected to test and develop the guidance algorithm is inspired by the Philae lander of the Rosetta mission. Rosetta is a Cornerstone Mission of the Horizon 2000 ESA Programme, launched in 2004, that has reached its target, comet 67P/Churyumov–Gerasimenko, in 2014. Since the landing body is a comet, no atmospheric drag is present and the gravity force that acts on the system is very low. However, since asteroids and comets are not spherical, the gravitational field may not be assumed to be homogeneous and the rotation can include large nutation or may even be chaotic. The Lander, whose overall mass is around 98 kg, is separated from the main Orbiter with high accuracy and a pre-adjustable velocity between 0.05 m/s to 0.52 m/s at an altitude between 1 km to 2 km and descends ballistically to the surface as shown by Figure 4.1, with a small landing ellipse (< 100 m) [5].
Tanking this situation as a reference, the initial nominal landing conditions considered as a test case are the ones reported in Table 4.1. As said, this kind

m_0 $[kg]$	$\vec{r}_0$ $[m]$	$\vec{v}_0$ $[m/s]$	g $[m/s^2]$
100	$[0, 0, 1000]^T$	$[0, 0, -0.52]^T$	10^{-10}

Table 4.1. Initial nominal lander conditions for the test case

of situation was considered as one of the possible best environments that could represent the low gravity without atmosphere pinpoint landing case to start the development of a new kind of guidance. In this context the comet gravity acceleration was considered as constant, helping in keeping the dynamics as simple as possible. In addition to the original Philae case, where only one z-axis oriented thruster was involved, in the considered test case also horizontal thrusters were

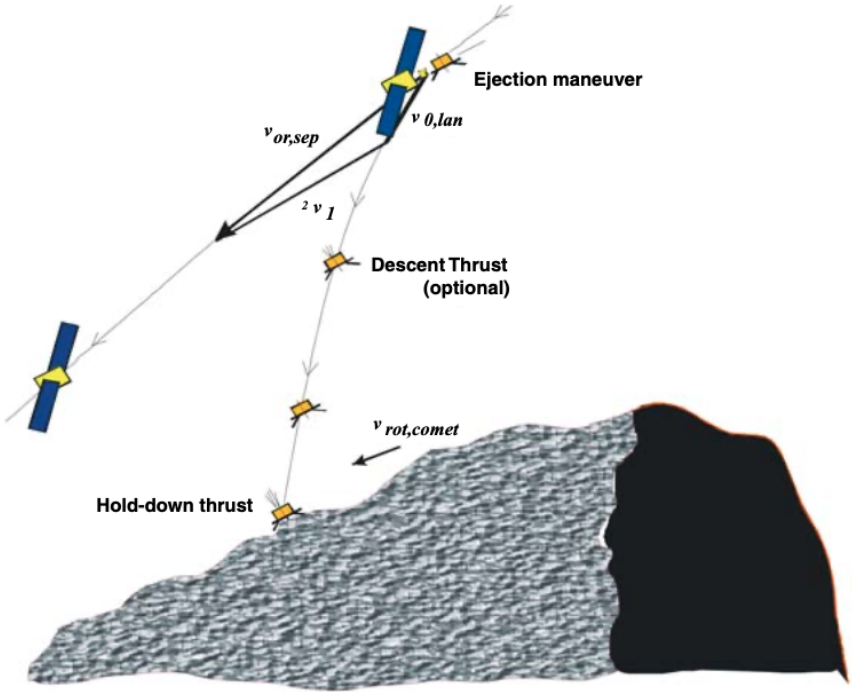


Figure 4.1. Philae landing scenario

applied, as explained in subsection 2.1.1, in order to counteract possible initial state errors and enable the possibility of having a landing ellipse considerably smaller with respect to the 100 m one of the Rosetta mission. In subsection 2.1.2 the initial state conditions were already presented without saying the considered numerical values. Being the selected nominal initial altitude of 1 km, an error of 100 m with respect to the horizontal position was considered. In this way it was considered a potential dispersion of 10 % with respect to the initial lander position in horizontal direction. Concerning the vertical position, a smaller error was initially considered as it will be showed in chapter 4. Having in mind this kind of situation and comparing the possible initial position with the initial nominal velocity, an electric kind of thrust was considered to be sufficient for the case taken under consideration, with the maximum and minimum thrust values and specific impulse reported in Table 4.2.

T_{max} $[mN]$	T_{min} $[mN]$	I_{sp} $[s]$
150	1	3000

Table 4.2. Thruster features for the learning process

Of course this range of thrust is something that can not be achieved by any kind of throttlable electric thruster at the moment. However, the aim of giving such an high range of values is to encourage the lander to explore different

possible levels of thrust along the learning process. In this way the possibility of finally stabilizing around an optimal range of thrust levels is increased. Finally, once obtained this range, the proper model of thruster can be selected. For instance, for the test case described above the engines reported in Table 4.3 were considered as possible options.

Thruster	**Thrust range** $[mN]$	I_{sp} $[s]$	**Mass** $[kg]$
QinetiQ T5 (Gridded Ion)	1 to 25	$>$ 3000	2.5
QinetiQ T6 (Gridded Ion)	20 to 230	$>$ 4200	8.3
RIT 10 EVO (Gridded Ion)	5 to 25	$\sim$ 3000	1.8
RIT 2X (Gridded Ion)	80 to 200	$\sim$ 3000	8.8
Arcjet	100 to 300	$\sim$ 1000	

Table 4.3. Electric thrusters features

A final important aspect is relative to the observation of the state space that is passed to the neural network. In fact, as described in subsection 3.4.1, this observation is represented by the following vector:

$$\vec{obs} = [v_x, v_y, v_z, t_{go}, r_z]^T \tag{4.1}$$

which is strictly connected to the initial state representation of the test case. In fact, since the initial horizontal position can be in any direction (subsection 2.1.2), instead of passing the components of the horizontal position vector (r_x and r_y), this information is passed thanks to the time-to-go obtained by Equation 4.2.

$$t_{go} = \frac{|\vec{r}|}{|\vec{v}|} \tag{4.2}$$

Eventually the absence of the lander translational coordinates results in a policy with good generalization in the policy's behavior that can extend to areas of the full state space not experienced during learning as also done by Brian Gaudet and Furfaro in [13].

4.2 Introduction to the results

In the following sections are reported the obtained results of the described guidance algorithm. As anticipated, all the three presented learning methods have been applied and tested. Moreover, two different cases have been analyzed concerning the dynamics. In fact, in order to identify the best learning method, at first a 2D case was applied where the only difference with respect to the 3D one previously described is the absence of velocity deviation along the y-axis (no

β_0 uncertainty), hence no thrust in this direction is needed leading to a simpler 2D dynamics. Then, once the best method was identified, the learning process in a 3D environment was executed.
Before starting with the analysis of the results obtained from each of the conducted tests, it is important to define some of the most important learning parameters (hyperparameters and environment parameters) that have been used for the learning process. These values, reported in Table 4.4, were identified after a long trial and error procedure and are the ones that are applied for all the following presented results. The chosen learning rate, that has the same value for

Learning parameters	
Learning Rates α and α_v	10^{-4}
RB and AC discount factor γ	0.99
A2C discount factor γ	0.95
Action Dimension	20
Hidden layers' neurons	64
Finite horizon time t_f $[s]$	30
Sampling time Δt $[s]$	3
d_{max} $[m]$	100
State weight w_x	1.5
Control action weight w_u	1
Horizontal position error boundary $[m]$	10
Horizontal velocity error boundary $[m/s]$	0.5
Vertical velocity error boundary $[m/s]$	1.5
Number of episodes	400 to 2000

Table 4.4. Main learning parameters values

both the policy and value networks, was found as a good compromise between learning time and weight updates. In fact, in general, the higher the learning rate is the faster the learning process is. However, this also implicates increasing values of gradient updates that could eventually cause very high training errors, hence a wrong learning process. Concerning the Δt and maximum horizontal position error accepted, these values come from a compromise between results and computational time. In fact, even considering the reported values for these parameters, the learning process takes approximately one minute for each episode, which means that considering 1000 episodes for instance the whole learning process takes approximately 17 h. In this context, in order to reduce the maximum horizontal position boundary, in general it would be needed a more refined trajectory integration, which means a reduction of the considered Δt, hence a consistent increment of the computational time. Anyway the considered Δt, especially if compared to the velocities involved, has an acceptable value as well as the horizontal boundary, which is the 10 % of the initial horizontal

position error, meaning that an overall reduction of the 90 % of the initial error is guaranteed.

4.3 2D

Concerning the above described 2D case, at first several learning processes of 1000 episodes each were tested using the initial conditions reported in Table 4.5, that reflects what previously explained in subsection 2.1.2 and section 4.1.
In this context, the first 10 action values were used to build the $u_x(t)$ while the other 10 were used for the $u_z(t)$. Unfortunately no results were obtained,

2D initial state conditions	
$\lvert\vec{v}_0\rvert$	$\pm 5\%$
x_0 $[m]$	-100 to 0
z_0 $[m]$	± 10
γ_0 $[deg]$	± 0.05

Table 4.5. 2D initial state conditions

probably due to the necessity of having a consistent higher number of episodes to manage a proper learning process. However, this would have meant also a large increase in the computational time that would have reached even several days, leading to a complicated situation to be dealt especially considering the fact that the applied method has the need of a large number of trial and error processes in order to optimize the algorithm as much as possible. For this reason, in order to achieve results while keeping a similar number of episodes it was decided to split the initial horizontal error range in five different sub-ranges of 20 m each (-100 m to -80 m, -80 m to -60 m, -60 m to -40 m, -40 m to -20 m, -20 m to 0 m). This subdivision is done with respect to the horizontal position error since it resulted to be the most influencing one. In this way five different networks are optimized for each sub-range and in the end they can be combined to obtain an overall policy that is able to deal with the whole range of error. By doing so it was possible to work at the same time on different networks and test the different methods in acceptable time frames. Eventually the main learning parameters were set to the ones reported in Table 4.4 and the only tuning parameter left was the reward given at the end of each episode, which resulted to be the most determinant one in order to obtain acceptable results. In fact, after having set these parameters, the training process resulted to start converging towards certain regions of solutions, making the lander to always land in a certain range of ground positions while always respecting the velocity boundaries. However, these positions were strongly dependent on the value of

final reward and so a trial and error process was needed in order to have these landing points inside the wanted boundary.

4.3.1 Reinforce with Baseline

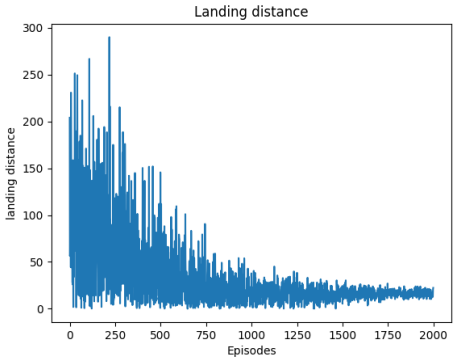
Using the RB learning algorithm some promising results for the $-60\,\mathrm{m}$ to $-40\,\mathrm{m}$ sub-range were obtained. In particular three different interesting results are described in this section considering the final rewards reported in Table 4.6, where the terminal conditions are the following:

- Positive Terminal: The final state respects all the imposed constraints
- Negative Terminal #1: The final state is outside the final horizontal position boundary but respects all the other constraints
- Negative Terminal #2: The final state does not respect the final velocity boundaries
- Negative Terminal #3: The episode ends without reaching ground

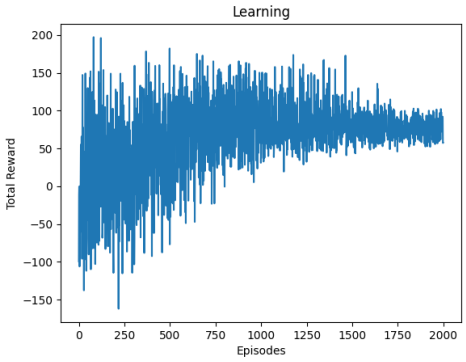
Case	**Positive Terminal reward**	**Negative Terminal #1 reward**	**Negative Terminal #2 reward**	**Negative Terminal #3 reward**
A	$50(1-(\frac{d}{10})^{0.4})$	$50(1-(\frac{d}{10})^{0.4})$	-1000	$-10r_z$
B	100	$-0.8d$	-1000	$-10r_z$
C	$100(1-(\frac{d}{100})^{0.6})$	$100(1-(\frac{d}{100})^{0.6})$	-1000	$-10r_z$

Table 4.6. RB Case A, B and C terminal rewards for the $-60\,\mathrm{m}$ to $-40\,\mathrm{m}$

The results obtained by these three cases are shown in Figure 4.2, Figure 4.3 and Figure 4.4. As it is possible to notice, there are different philosophies behind these three kinds of final rewards. Case A considers a reward that is shaped in the same way for both the Positive and Negative Terminal #1. In fact the idea is to have a continuous function that rewards the agent in a continuous manner depending on the final position, giving negative rewards for being out of the boundary, zero reward for being exactly on the boundary and positive rewards for being inside. On the contrary, in case B the reward is shaped only for the case of being outside the boundary, whereas a constant positive reward is given for being inside, with a value that is totally independent from how close to the target the final position is. Finally case C does something really similar to case A but with the only difference of giving positive rewards even in the case of a Negative Terminal #1 in order to encourage the learning process of the system in any case. By looking at the reported figures, it is possible to understand that

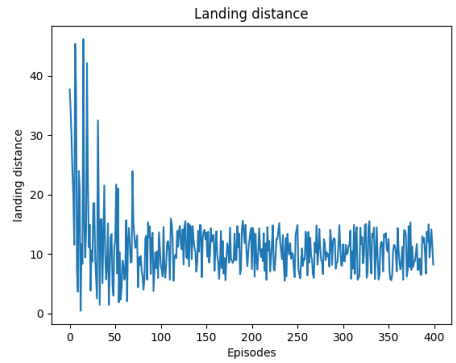


(a) Case A horizontal landing position along learning

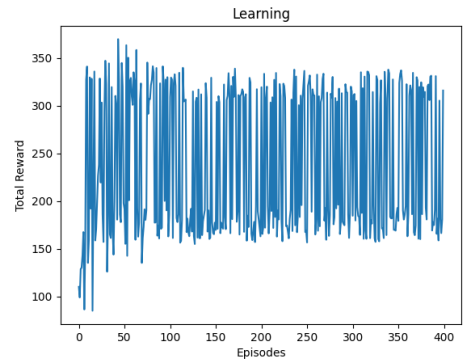


(b) Case A total reward trend along learning

Figure 4.2. Reinforce with Baseline case A, $-60\,\mathrm{m}$ to $-40\,\mathrm{m}$ sub-range

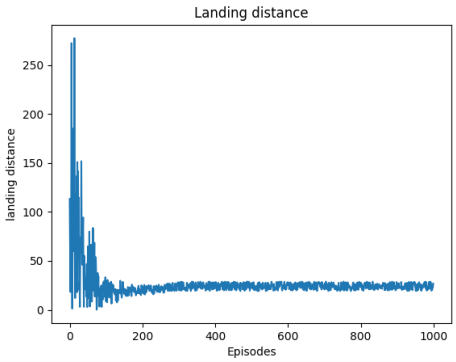


(a) Case B horizontal landing position along learning

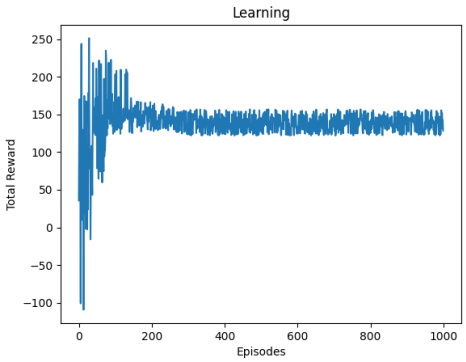


(b) Case B total reward trend along learning

Figure 4.3. Reinforce with Baseline case B, $-60\,\mathrm{m}$ to $-40\,\mathrm{m}$ sub-range



(a) Case C horizontal landing position along learning



(b) Case C total reward trend along learning

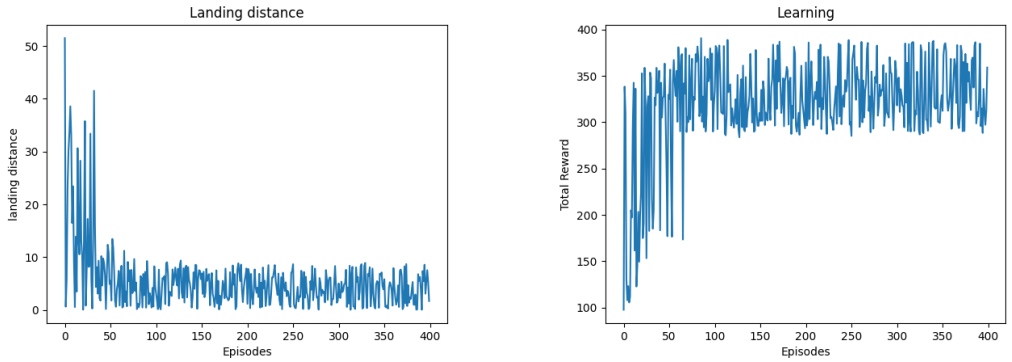
Figure 4.4. Reinforce with Baseline case C, $-60\,\mathrm{m}$ to $-40\,\mathrm{m}$ sub-range

the second option is the one that gives the best results. In fact, while in case A and C the landing position is stabilized slightly above the boundary of $10\,\mathrm{m}$,

in case B the landing positions result to be inside the wanted boundary in a very high number of cases. However, despite being the best result obtained over several tests with this method, this is still not an acceptable result since the number of landing points outside the boundary is considerably high.
Different tries were done also concerning the other sub-ranges. Unfortunately, none of them resulted in a completely successful learning, leading to similar results to the ones reported above.

4.3.2 Actor-Critic

Concerning the AC learning algorithm, also in this case some promising results for the -60 m to -40 m sub-range were obtained considering different final rewards, where only the rewards for the Positive Terminal and Negative Terminal #1 were changed with respect to the previous case, keeping the others identical to the ones of Table 4.6. However, a successful learning was actually obtained only by using the very same final rewards of the RB case B, as shown by Figure 4.5. In fact it is possible to notice that in this case acceptable results are obtained, since the final landing horizontal position always falls below the considered 10 m boundary and the overall total reward of each episode is stabilized around high positive values. The reason why this method produced better results with respect to the previous one is probably due to the fact that by learning at each step the agent is able to understand what are the best actions to take directly throughout the episode. For this reason, if the rewards are correctly shaped, the agent is able to produce better trajectories.
Unfortunately, despite the high number of tests, it resulted very difficult to obtain similar results for all the other sub-ranges.



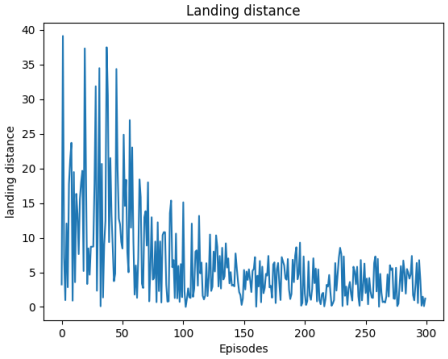
(a) Actor-Critic horizontal landing position along learning

(b) Actor-Critic total reward trend along learning

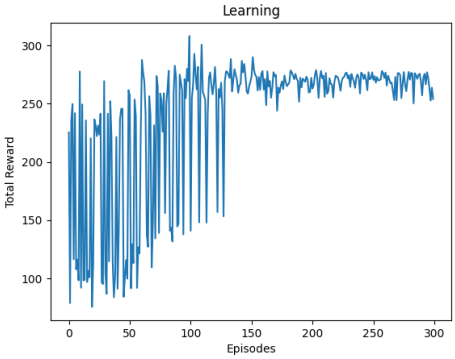
Figure 4.5. Actor-Critic -60 m to -40 m sub-range

4.3.3 Advantage Actor-Critic

Finally with the A2C method it was possible to obtain acceptable results for all the considered sub-ranges as showed by Figure 4.6, Figure 4.7, Figure 4.8, Figure 4.9 and Figure 4.10. In order to obtain these results the final rewards reported in Table 4.7, obtained after a large number of trial and error tests, were used. As for the previous case, the reward for Negative Terminal #2 and Negative Terminal #3 were kept the same of the RB case.

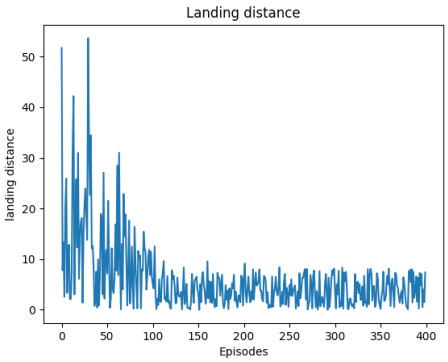


(a) −20 m to 0 m range horizontal landing position along learning

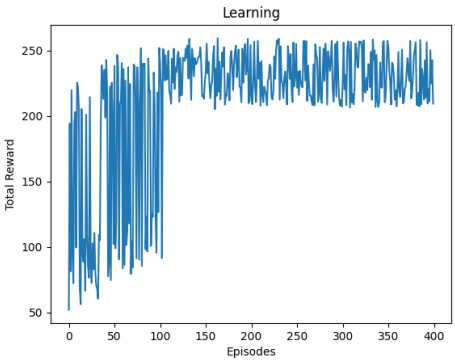


(b) −20 m to 0 m range total reward trend along learning

Figure 4.6. Advantage Actor-Critic −20 m to 0 m range

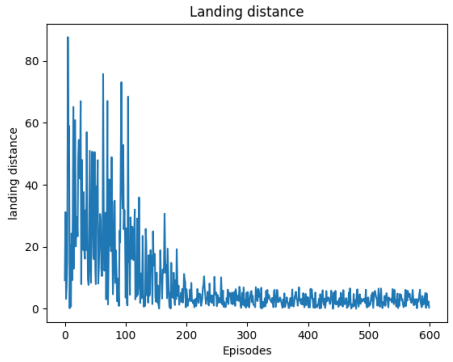


(a) −40 m to −20 m range horizontal landing position along learning

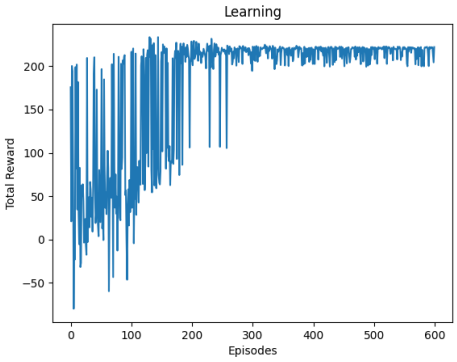


(b) −40 m to −20 m range total reward trend along learning

Figure 4.7. Advantage Actor-Critic −40 m to −20 m range

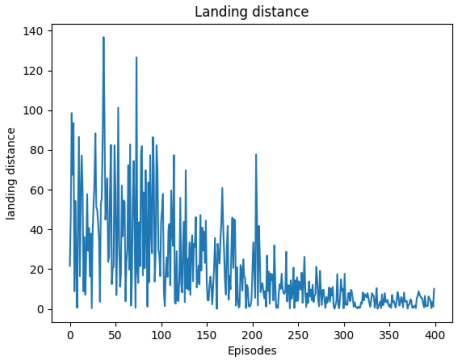


(a) −60 m to −40 m range horizontal landing position along learning

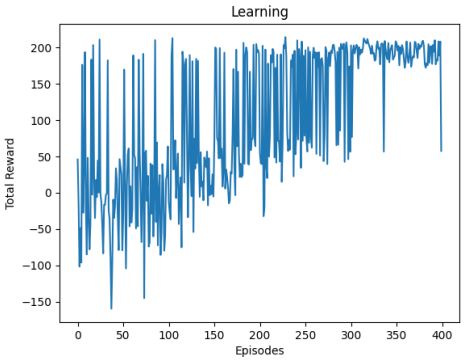


(b) −60 m to −40 m range total reward trend along learning

Figure 4.8. Advantage Actor-Critic −60 m to −40 m range

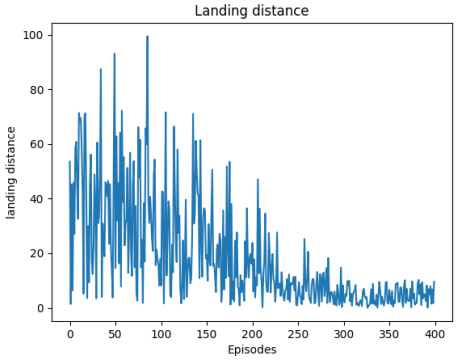


(a) −80 m to −60 m range horizontal landing position along learning

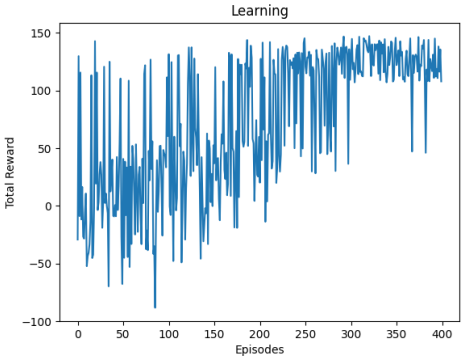


(b) −80 m to −60 m range total reward trend along learning

Figure 4.9. Advantage Actor-Critic −80 m to −60 m range



(a) −100 m to −80 m range horizontal landing position along learning



(b) −100 m to −80 m range total reward trend along learning

Figure 4.10. Advantage Actor-Critic −100 m to −80 m range

As it is possible to see, the learning process resulted successful in all the presented

Range	Positive Terminal reward	Negative Terminal #1 reward
$-100\,\text{m}$ to $-80\,\text{m}$	100	$-d$
$-80\,\text{m}$ to $-60\,\text{m}$	100	$-1.3d$
$-60\,\text{m}$ to $-40\,\text{m}$	100	$-1.3d$
$-40\,\text{m}$ to $-20\,\text{m}$	100	-10
$-20\,\text{m}$ to $0\,\text{m}$	100	-10

Table 4.7. 2D A2C terminal rewards for every sub-range

cases, leading to a total reward that is stabilized around high values and a final landing distance that is always inside the considered boundary of 10 m. The fact that this last algorithm is the one that produced the best results is probably due to its nature. In fact, since this algorithm starts the learning process from the last state, very high importance is given to this state and its related reward. Moreover, another important aspect could be the fact that, as reported in subsection 3.2.3, the A2C algorithm shares the very same network model of the AC one, which is the first one that produced very promising results. As shown in Table 4.7, the final rewards used are of the same kind of the previously presented RB case B, which resulted to be the best one also for the other two tested algorithms as described above. In this sense the analysis done with those algorithms resulted to be fundamental to identify a kind of final reward shape that could produce acceptable results. Indeed the rewards here reported present a fixed positive reward for the case of a Positive Terminal and a negative reward proportional to the landing distance from the target point in case of Negative Terminal #1. An exception is done with respect to the $-40\,\text{m}$ to $-20\,\text{m}$ and $-20\,\text{m}$ to $0\,\text{m}$ sub-ranges where a fixed negative reward for Negative Terminal #1 resulted to be sufficient.

Network test

In order to test the above presented working results, the initial conditions described in Table 4.5 were changed by increasing some values and introducing an uncertainty in terms of initial mass magnitude as showed by Table 4.8.

2D test initial state conditions	
$\lvert\vec{v}_0\rvert$	$\pm 10\%$
$x_0\ [m]$	-100 to 0
$z_0\ [m]$	± 50
$\gamma_0\ [deg]$	± 0.1
m_0	$\pm 5\%$

Table 4.8. 2D test initial state conditions

Starting from these values an overall number of 500 trajectories was evaluated using the combination of all the trained networks presented above. As it is possible to see from Figure 4.11, all the evaluated trajectories manage to land inside the boundary with the only exception of few of them that in any case still manage to land very close to the 10 m boundary. Concerning the velocity constraints, they are always respected. An important aspect that has been noticed throughout the test sessions of the trained network is the great sensitivity with respect to the value of γ_0 which resulted to be the most influencing parameter in terms of final landing precision.

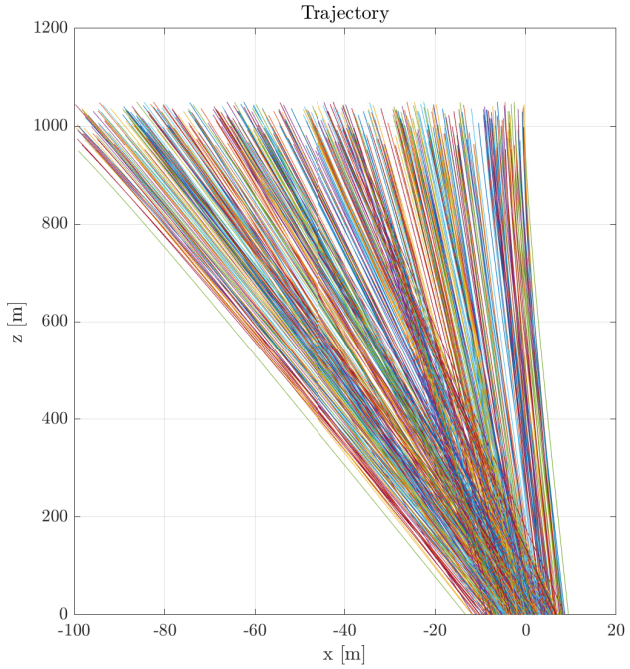
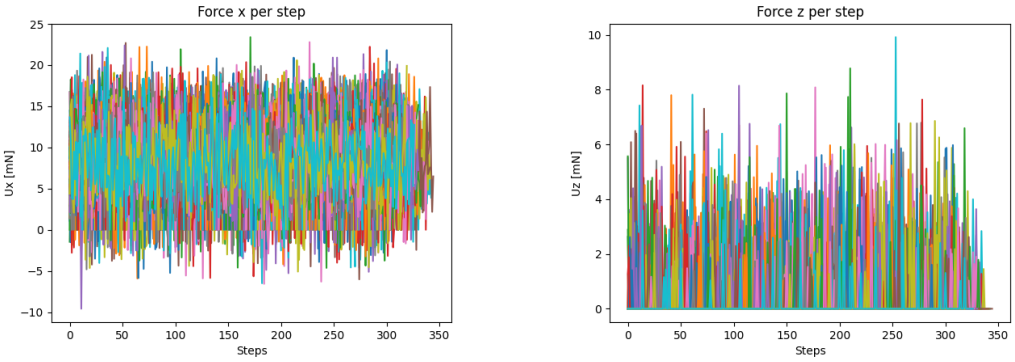


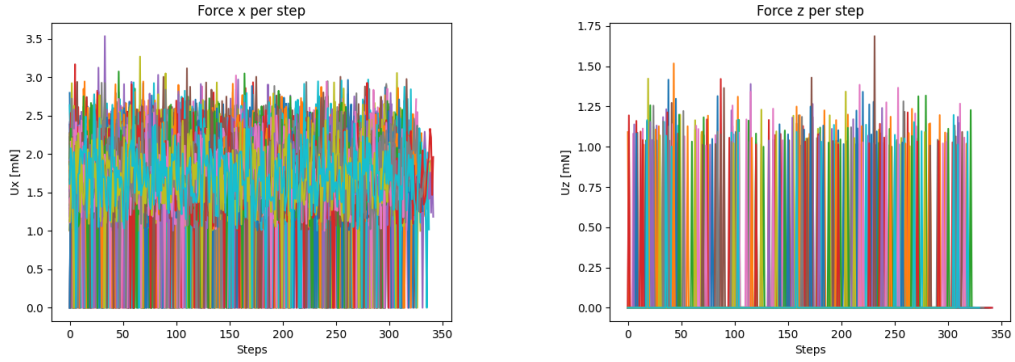
Figure 4.11. 2D case test Trajectories

Concerning the amount of thrust that each thruster has to employ, the trend of both u_x and u_z for the extreme sub-ranges are reported in Figure 4.12 and Figure 4.13. As it is possible to see the values of thrust never exceed 25 m N, confirming that the choice of an electric kind of thrust is correct. Moreover, by referring to Table 4.3, it is possible to see that choosing the *QinetiQ T*5 engine could be a good option for the analysed case. As expected, the required thrust in x direction is higher with respect to the one in z direction and the closer we start to the wanted horizontal position, the lower the required magnitude of thrust is, confirming the quality of the obtained results.



(a) $-100\,\mathrm{m}$ to $-80\,\mathrm{m}$ range u_x (b) $-100\,\mathrm{m}$ to $-80\,\mathrm{m}$ range u_z

Figure 4.12. 2D $-100\,\mathrm{m}$ to $-80\,\mathrm{m}$ range u_x and u_z



(a) $-20\,\mathrm{m}$ to $0\,\mathrm{m}$ range u_x (b) $-20\,\mathrm{m}$ to $0\,\mathrm{m}$ range u_z

Figure 4.13. 2D $-20\,\mathrm{m}$ to $0\,\mathrm{m}$ range u_x and u_z

4.4 3D

Since the best results for the 2D case were obtained by using the A2C method, this was the only method considered for the learning process of the 3D case. As previously done, also in this case the horizontal position initial error was divided in five sub-ranges for the same reasons explained in section 4.3. All the learning processes were developed using the initial conditions reported in Table 4.9, where the uncertainties of the initial flight path angle (γ_0) and heading angle (β_0) are typical values that can be found in other landing missions, as it is possible to see in [24]. In this case the action dimension was kept the same of the 2D case. The first 10 values were used to build an $u_{xy}(t)$ kind of control while the last 10 values were used to build $u_z(t)$. The $u_{xy}(t)$ is the overall control action magnitude on the horizontal xy plane directed towards the origin of the system. In this context, the consequent $u_x(t)$ and $u_y(t)$ are the projections of $u_{xy}(t)$ over the x and y-axis. As said for other previous decisions, also in this case this kind of choice related to the number of actions was done in order to

3D initial state conditions	
$\lvert\vec{v}_0\rvert$	$\pm 5\%$
$x_0\ [m]$	-100 to 0
$z_0\ [m]$	± 10
$\gamma_0\ [deg]$	± 0.1
$\beta_0\ [deg]$	± 0.5

Table 4.9. 3D initial state conditions

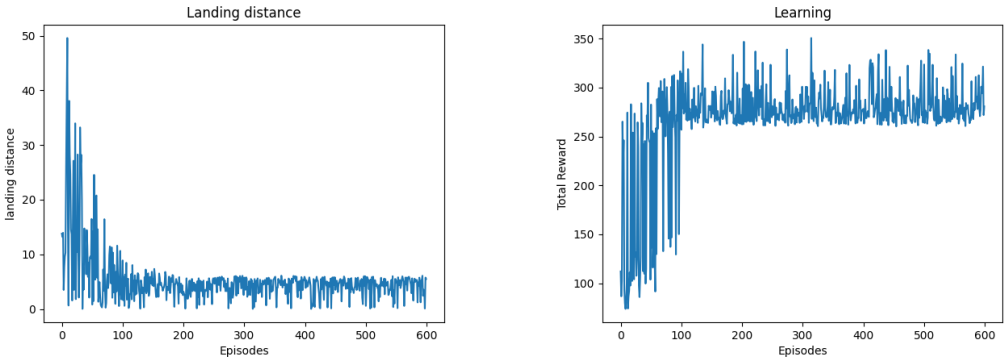
keep the computational time as low as possible and be able to conduct as many tests as possible.

4.4.1 Advantage Actor-Critic

Also in this case a successful learning was obtained for all the considered sub-ranges as showed by Figure 4.14, Figure 4.15, Figure 4.16, Figure 4.17 and Figure 4.18, where the considered values of the final rewards, obtained after a large number of tries, are reported in Table 4.10.

Range	Positive Terminal reward	Negative Terminal #1 reward
-100 m to -80 m	100	$-1.55d$
-80 m to -60 m	100	$-1.8d$
-60 m to -40 m	100	$-1.3d$
-40 m to -20 m	100	-19
-20 m to 0 m	100	-20

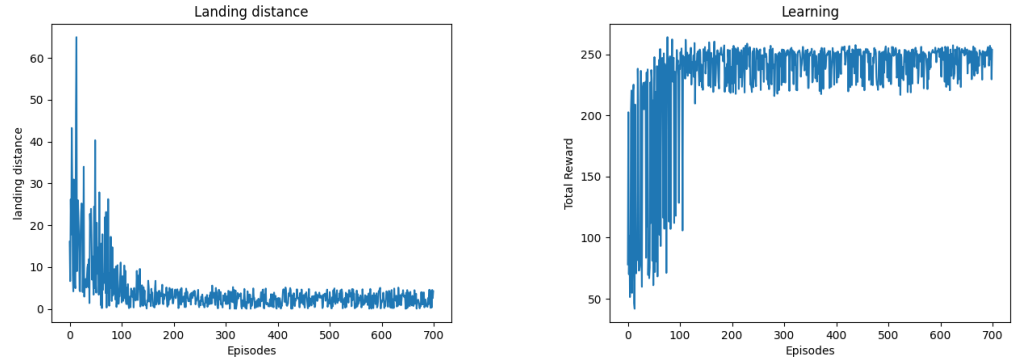
Table 4.10. 3D A2C terminal rewards for every sub-range



(a) -20 m to 0 m range horizontal landing position along learning

(b) -20 m to 0 m range total reward trend along learning

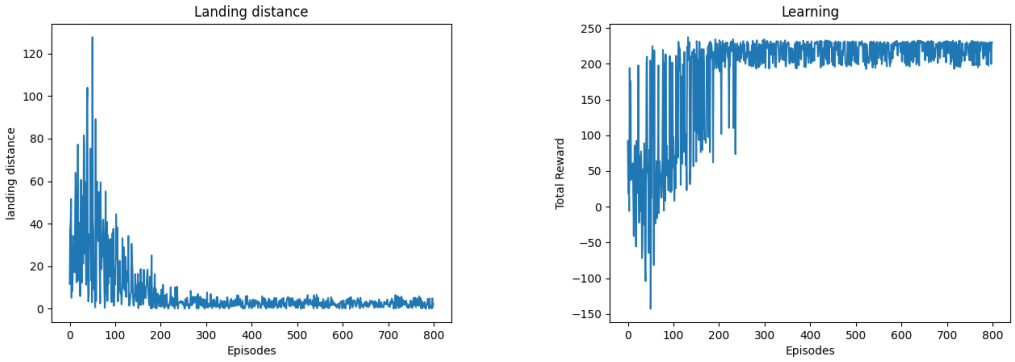
Figure 4.14. Advantage Actor-Critic -20 m to 0 m range



(a) −40 m to −20 m range horizontal landing position along learning

(b) −40 m to −20 m range total reward trend along learning

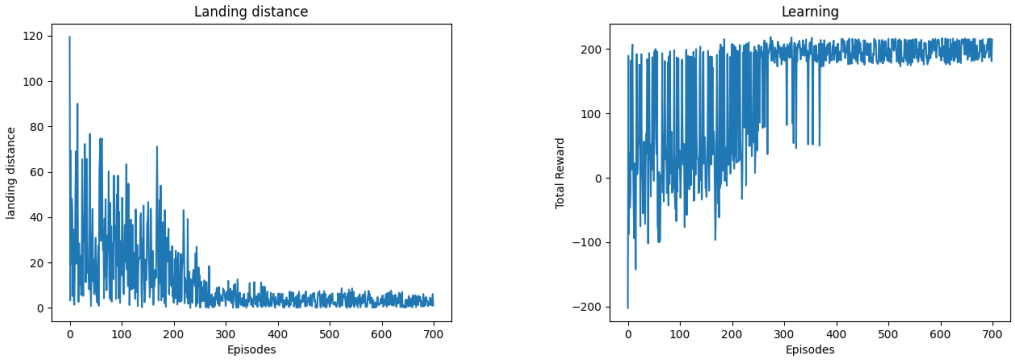
Figure 4.15. Advantage Actor-Critic −40 m to −20 m range



(a) −60 m to −40 m range horizontal landing position along learning

(b) −60 m to −40 m range total reward trend along learning

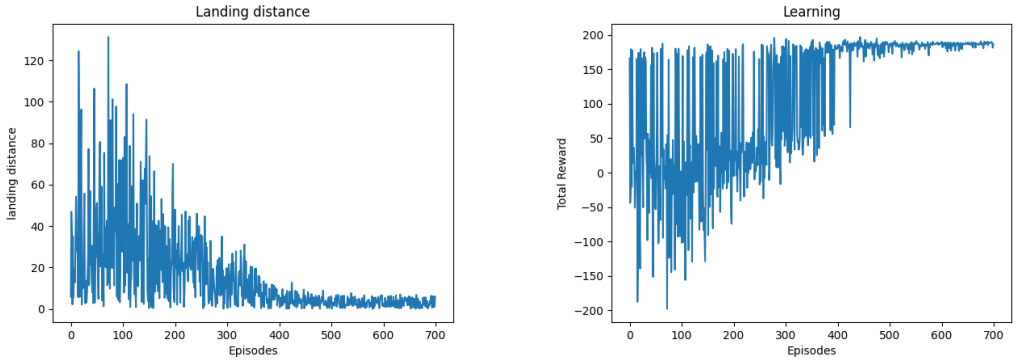
Figure 4.16. Advantage Actor-Critic −60 m to −40 m range



(a) −80 m to −60 m range horizontal landing position along learning

(b) −80 m to −60 m range total reward trend along learning

Figure 4.17. Advantage Actor-Critic −80 m to −60 m range



(a) −100 m to −80 m range horizontal landing position along learning

(b) −100 m to −80 m range total reward trend along learning

Figure 4.18. Advantage Actor-Critic −100 m to −80 m range

As it is possible to notice, in all the reported cases, the total reward is stabilized around very high values and the final horizontal landing position is always below 10 m, demonstrating the success of the learning process. Concerning the final reward shapes, the considerations done in subsection 4.3.3 can be confirmed.

Network test

As previously done, also in this case the trained networks were tested by increasing the initial conditions values to the ones reported in Table 4.11.

3D test initial state conditions	
$\lvert\vec{v}_0\rvert$	$\pm 10\%$
x_0 $[m]$	-100 to 0
z_0 $[m]$	± 50
γ_0 $[deg]$	± 0.5
β_0 $[deg]$	-0.5 to 0
m_0	$\pm 5\%$

Table 4.11. 3D test initial state conditions

It can be noticed that the heading angle uncertainty has been considered only in the range of negative values in order to exploit at maximum the symmetry of the system, since in the case of having positive β_0 angles the situation would be symmetrical in terms of applied u_y. An important difference with respect to what done for the 2D case is the increased uncertainty in terms of γ_0 that the system is able to overcome. This is probably due to the fact that, contrarily to what done for the 2D case, in this case the networks were trained by immediately considering the presence of an higher uncertainty for this value. Also in this case an overall number of 500 trajectories was evaluated and as it is possible to

see from Figure 4.19 and Figure 4.20, the combination of the obtained networks is able to satisfy the requirements. In fact only 9 trajectories out of 500 landed outside the position boundary, but still really close to the desired 10 m distance from the origin of the reference system (the desired landing point).

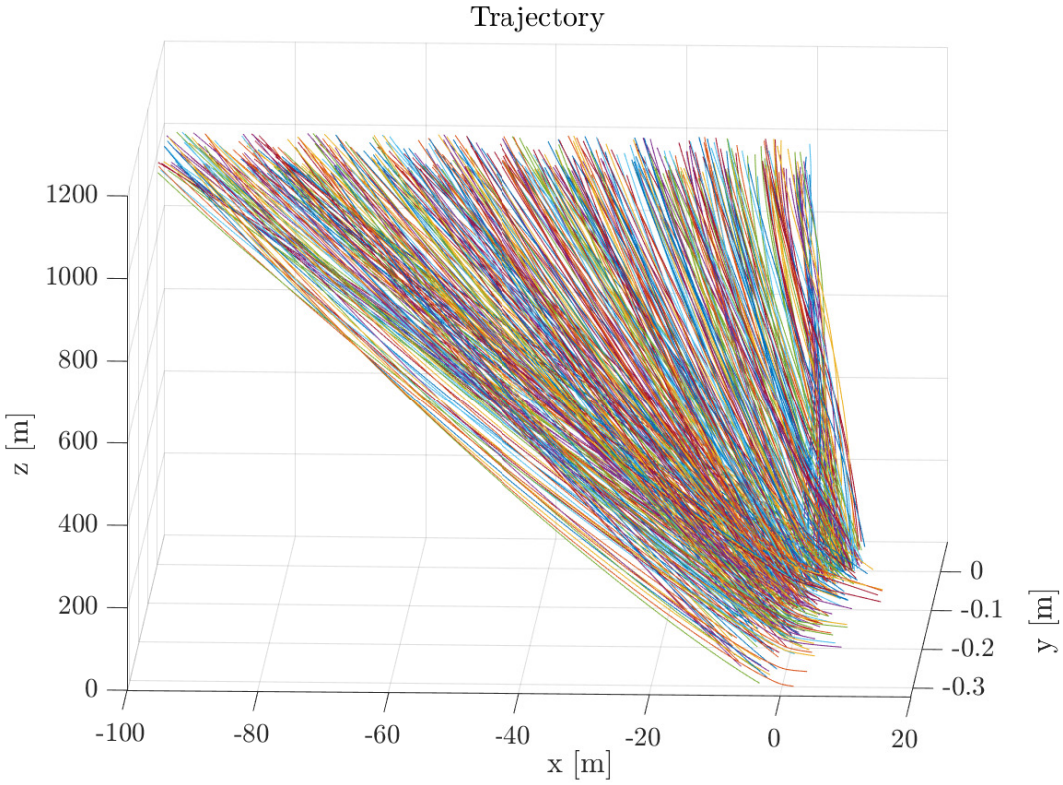


Figure 4.19. 3D case test Trajectories

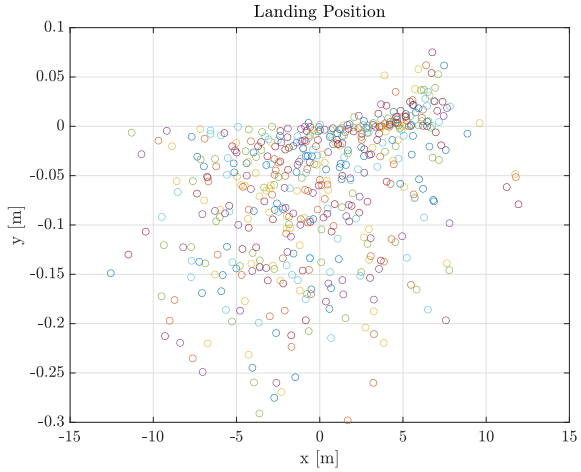
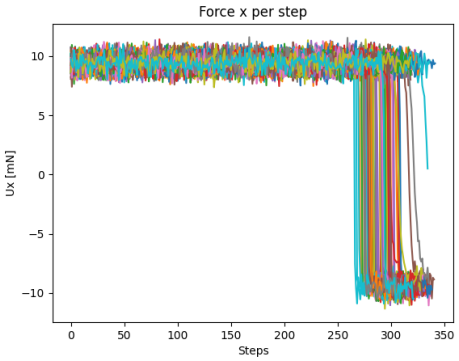


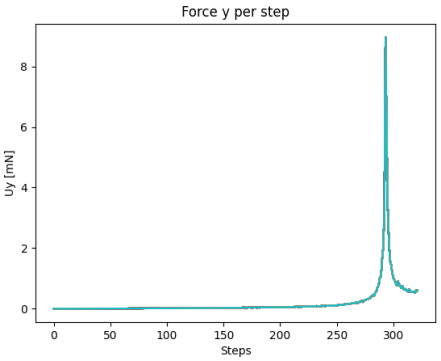
Figure 4.20. 3D case test Landing Points

Concerning the amount of thrust that each thruster has to employ, the trend of u_x, u_y and u_z for the extreme sub-ranges are reported in Figure 4.21 and

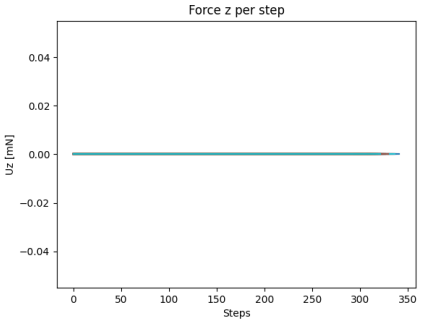
Figure 4.22. In Figure 4.22, the u_z trend is not reported since it is equal to the one of Figure 4.21.



(a) 3D case test u_x for the $-100\,\text{m}$ to $-80\,\text{m}$ sub-range

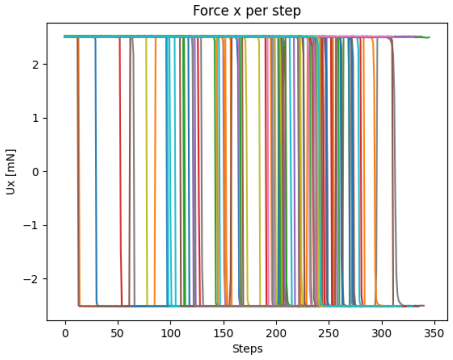


(b) 3D case test u_y for the $-100\,\text{m}$ to $-80\,\text{m}$ sub-range

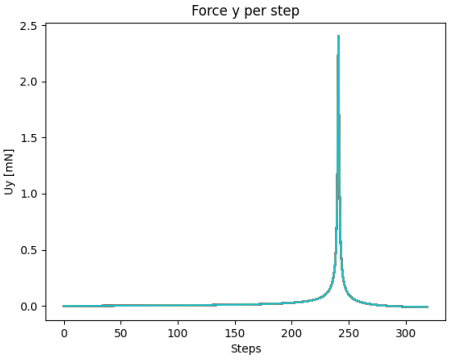


(c) 3D case test u_z for the $-100\,\text{m}$ to $-80\,\text{m}$ sub-range

Figure 4.21. 3D $-100\,\text{m}$ to $-80\,\text{m}$ range u_x, u_y and u_z



(a) 3D case test u_x for the $-20\,\text{m}$ to $0\,\text{m}$ sub-range



(b) 3D case test u_y for the $-20\,\text{m}$ to $0\,\text{m}$ sub-range

Figure 4.22. 3D $-20\,\text{m}$ to $0\,\text{m}$ range u_x and u_y

In this case the overall magnitude of thrust needed is even lower with respect to

the 2D case, symptom of an even better learning process. In fact the maximum value of thrust is around 10 m N. As a consequence, also in this case the *QinetiQ T*5 engine could result in a good choice for the considered case. A peculiarity is the complete absence of vertical thrust needed this time. However, this makes sense thinking about the involved values of initial velocity, gravity acceleration and vertical velocity boundary. Another interesting aspect is the trend of u_x. In fact, as it is possible to notice, the applied thrust initially accelerates the lander towards the landing position and then, once it is close to the target, a negative thrust is applied in order to slow down the system. Concerning u_y, it is applied really close to ground in order to reduce the deviation along the y direction before landing.
Eventually, by exploiting the system symmetry, it is possible to obtain a complete 3D graph of the considered situation in Figure 4.23 and Figure 4.24, from which it is possible to appreciate even more the obtained results.

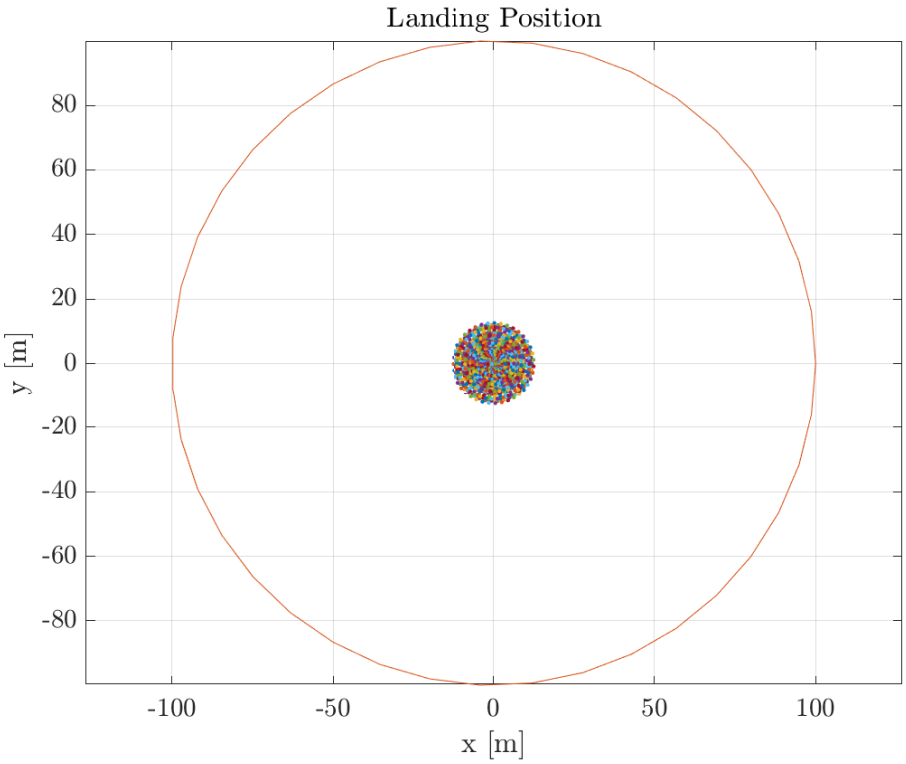


Figure 4.23. 3D case test overall Landing Points on ground plane compared to the initial maximum horizontal position error

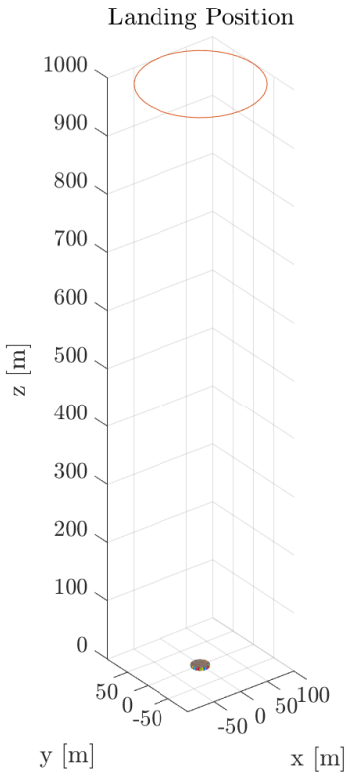


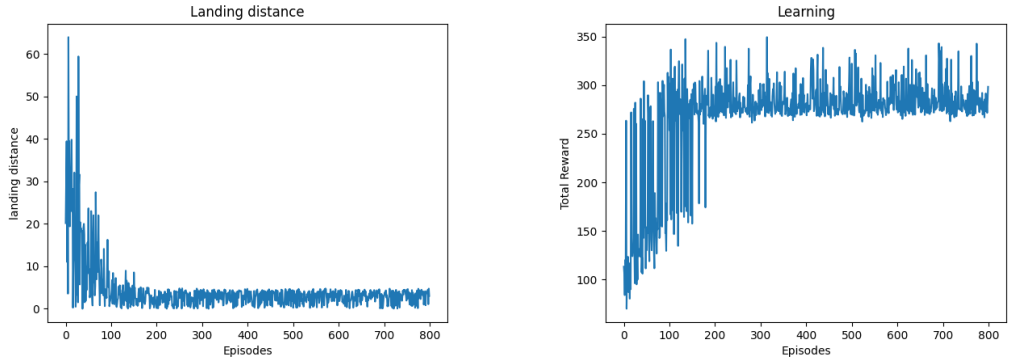
Figure 4.24. 3D case test overall Landing points compared to the initial maximum horizontal position error

4.4.2 A2C reduced horizontal error case

Despite what said in section 4.2 relatively to the possibility of reducing the horizontal error boundary, a final try was done in any case to understand if it was possible to manage to reduce the accepted horizontal position error at landing from 10 m to 5 m while keeping the same Δt to maintain a computational time similar to the one of the previous learning processes. Surprisingly, acceptable results were obtained also in this case for all the considered sub-ranges as shown in Figure 4.25, Figure 4.26, Figure 4.27, Figure 4.28 and Figure 4.29, by using the final rewards reported in Table 4.12.

Range	**Positive Terminal reward**	**Negative Terminal #1 reward**
-100 m to -80 m	50	$-0.9d$
-80 m to -60 m	100	$-1.8d$
-60 m to -40 m	100	$-0.8d$
-40 m to -20 m	100	-16
-20 m to 0 m	100	-10.5

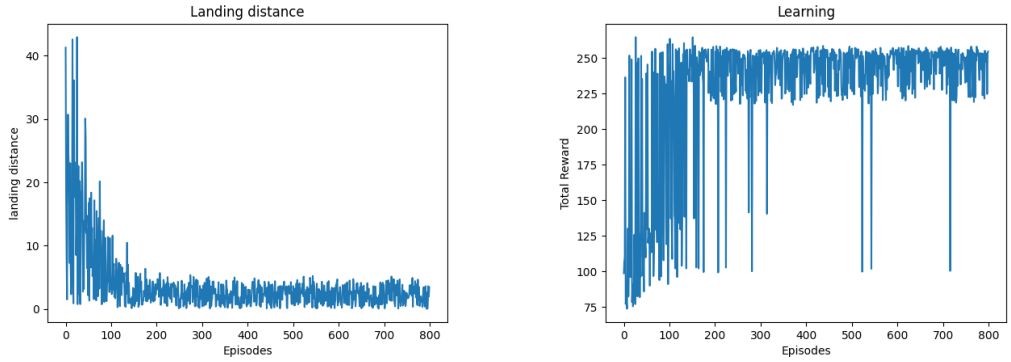
Table 4.12. 3D A2C terminal rewards for every sub-range



(a) −20 m to 0 m range horizontal landing position along learning

(b) −20 m to 0 m range total reward trend along learning

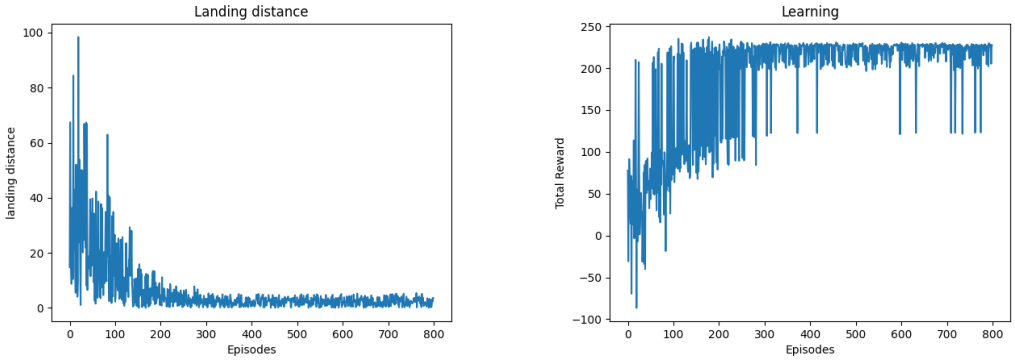
Figure 4.25. Advantage Actor-Critic −20 m to 0 m range



(a) −40 m to −20 m range horizontal landing position along learning

(b) −40 m to −20 m range total reward trend along learning

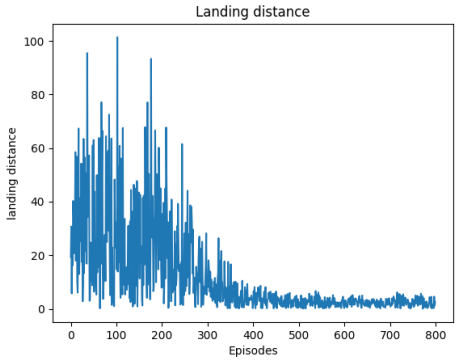
Figure 4.26. Advantage Actor-Critic −40 m to −20 m range



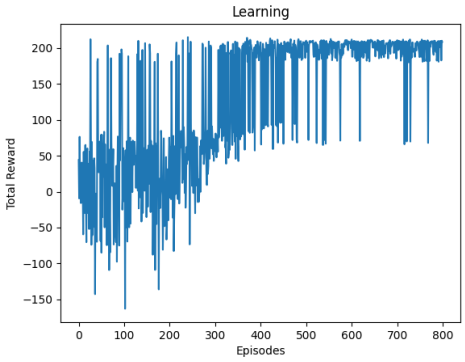
(a) −60 m to −40 m range horizontal landing position along learning

(b) −60 m to −40 m range total reward trend along learning

Figure 4.27. Advantage Actor-Critic −60 m to −40 m range

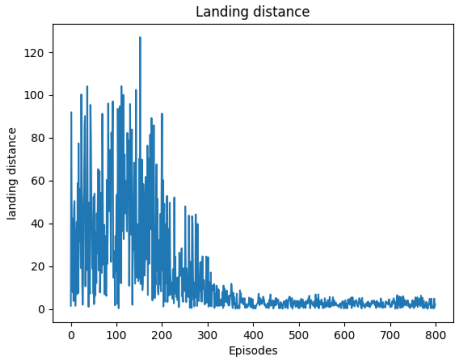


(a) −80 m to −60 m range horizontal landing position along learning

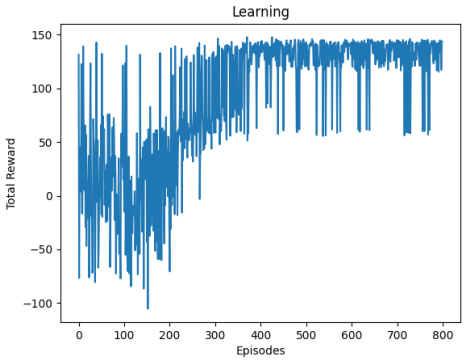


(b) −80 m to −60 m range total reward trend along learning

Figure 4.28. Advantage Actor-Critic −80 m to −60 m range



(a) −100 m to −80 m range horizontal landing position along learning



(b) −100 m to −80 m range total reward trend along learning

Figure 4.29. Advantage Actor-Critic −100 m to −80 m range

Unfortunately, as it is possible to see from the total reward trend figures, the learning processes are not as optimal as the ones with the 10 m error boundary. In fact, once that the total reward is stabilized around high values there are still few cases that do not respect the imposed final position boundary, causing a drop in terms of total reward for those particular episodes. Moreover, finding the values reported in Table 4.12 in order to achieve the reported results required even a greater number of trial and error processes with respect to the previous cases. For these reasons it is possible to confirm that a reduction of Δt together with an increase in the number of actions that the networks have to produce are needed to obtain more optimal results. Indeed in this way a more refined control action wold be produced leading to more precise results that could let the lander to always land even in smaller horizontal boundaries. Eventually this would enhance the overall performances but sensibly increase the computational learning time at the same time. However, the obtained results just reported are

the best ones obtained during the whole work in terms of final landing precision since the produced networks are able to achieve a 5 m landing precision, halving the previously accepted error.

Network test

Finally also for this last case the trained networks were tested by increasing the initial conditions values to the values reported in Table 4.13. As previously done, a total number of 500 trajectories was evaluated as shown in Figure 4.30 and Figure 4.31.

3D test initial state conditions	
$\lvert\vec{v}_0\rvert$	$\pm 10\%$
$x_0\ [m]$	-100 to 0
$z_0\ [m]$	± 50
$\gamma_0\ [deg]$	± 0.1
$\beta_0\ [deg]$	-0.5 to 0
m_0	$\pm 5\%$

Table 4.13. 3D test initial state conditions

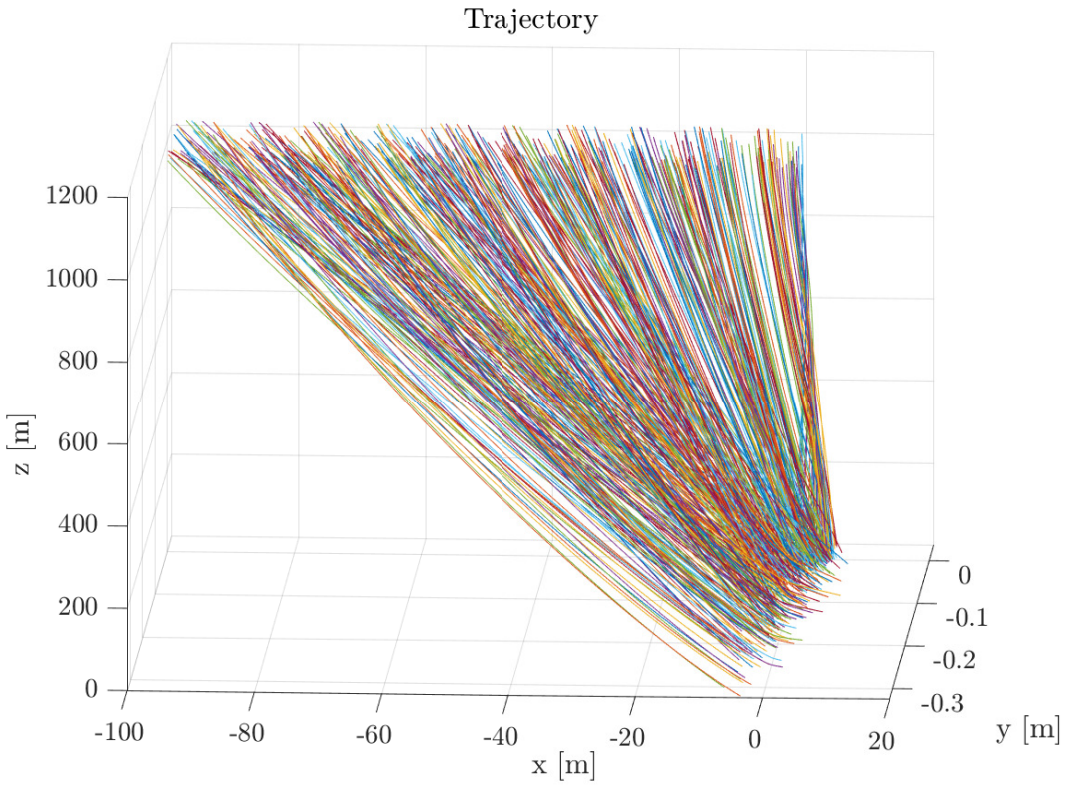


Figure 4.30. 3D case test Trajectories

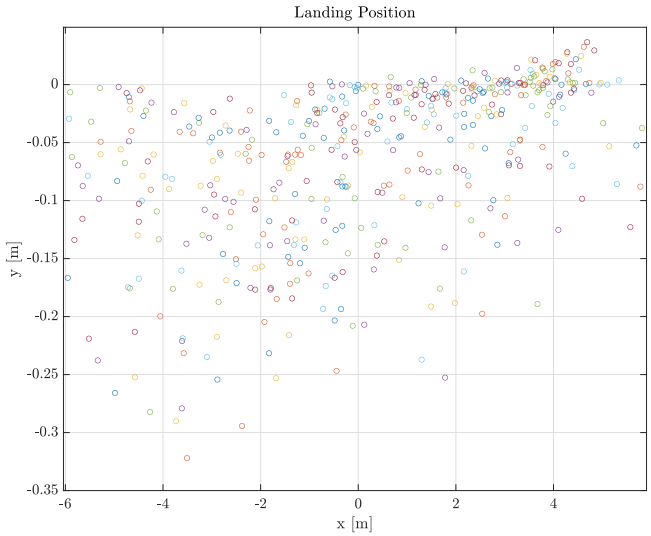
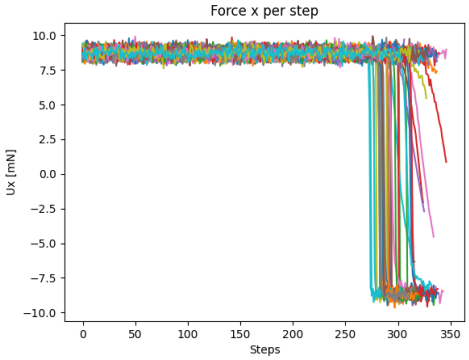


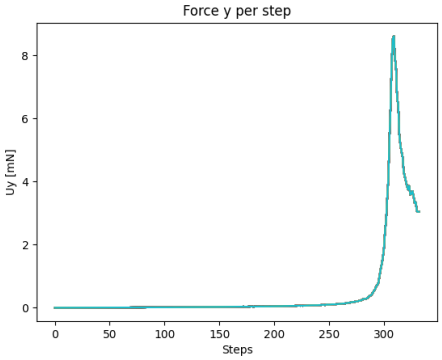
Figure 4.31. 3D case test Landing Points

The overall networks combination is able to satisfy the requirements in most of the cases, leading to landing points inside the 5 m error boundary excluding a certain number of episodes that still manage to land close to the considered boundary. However, as it is possible to see, the number of points outside the target boundary is relatively high, confirming the non completely optimal result of the learning process. Moreover, these results are obtained by keeping the β_0 and γ_0 values in the same uncertainty ranges used during the training process, since an increase in these values, especially concerning γ_0, would cause landing points even in the range of 10 m. This proves the fact that the overall learning process was more successful for the previous 3D case, since those networks resulted to be able to withstand higher uncertainties by keeping almost unchanged the overall precision with respect to the considered error boundary. These considerations highlight even more the need of a reduced Δt and an increased number of action to achieve similar results for a reduced error boundary case. At the same time, the results here reported demonstrate the feasibility of performing a learning process able to achieve very important results in terms of landing accuracy.
Concerning the amount of thrust that each thruster has to employ, the trend of u_x,u_y and u_z for the extreme sub-ranges are reported in Figure 4.32 and Figure 4.33. In this case the overall magnitude of thrust needed is even lower with respect to the previous case. In fact the maximum value of thrust never exceeds 10 m N. As a consequence, also in this case the *QinetiQ T*5 engine could result as a good engine choice. Concerning the trends of u_x and u_y, they are very similar to the ones of the 10 m boundary case, hence similar considerations can be done. On the contrary, the u_z trend is different. In fact it is possible to notice the presence of a non-zero vertical thrust in this case. This is probably

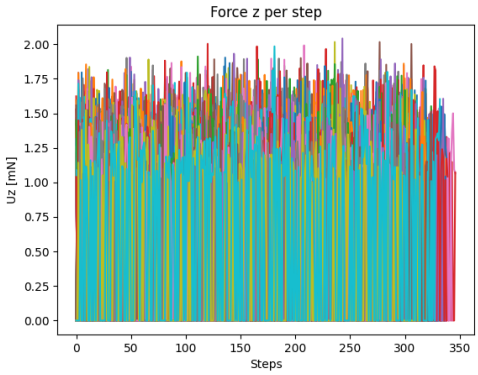
needed in order to slow down the vertical descent and let the horizontal thrust act for a longer time with respect to the 10 m boundary case. Eventually, by acting in this way, this thrust enables the lander to achieve the landing inside the 5 m position boundary and reduces the final vertical velocity at the same time.



(a) 3D case test u_x for the -100 m to -80 m sub-range

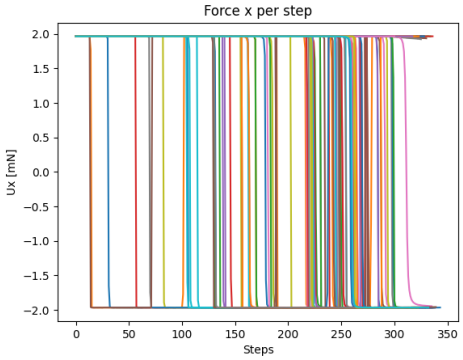


(b) 3D case test u_y for the -100 m to -80 m sub-range

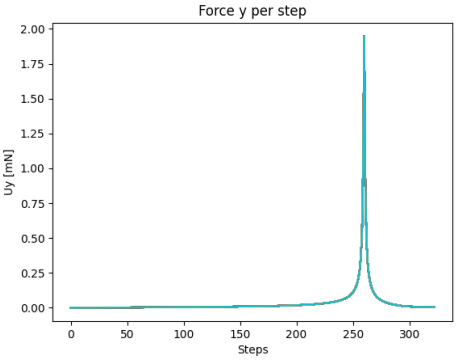


(c) 3D case test u_z for the -100 m to -80 m sub-range

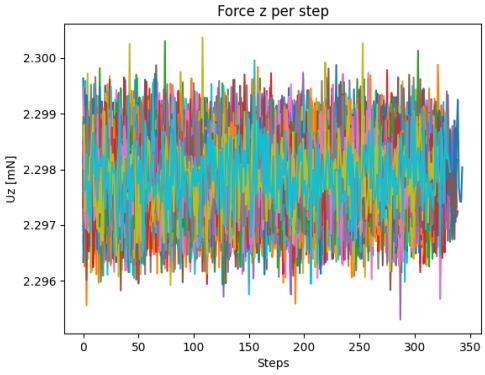
Figure 4.32. 3D -100 m to -80 m range u_x, u_y and u_z



(a) 3D case test u_x for the $-20\,\mathrm{m}$ to $0\,\mathrm{m}$ sub-range



(b) 3D case test u_y for the $-20\,\mathrm{m}$ to $0\,\mathrm{m}$ sub-range



(c) 3D case test u_z for the $-20\,\mathrm{m}$ to $0\,\mathrm{m}$ sub-range

Figure 4.33. 3D $-20\,\mathrm{m}$ to $0\,\mathrm{m}$ range u_x, u_y and u_z

Chapter 5

Conclusions

In conclusion it is possible to say that the new guidance developed in this work presents some interesting and promising results. In fact in the end it was obtained a system able to guide the lander inside the decided boundary of landing points counteracting some consistent errors achieving a pinpoint landing in low gravity and no atmosphere conditions. One of the main problems of this method is for sure the difficulty in setting the parameters needed for the learning process and the amount of computational time needed by this process itself, especially in the case of small Δt required. However, it is important to remember that all this training procedure does not have any particular time constraints since it is not something that is done while the system is at work in a real environment. In fact the training is done on ground and then, once the system is deployed, the obtained networks can be used by the lander in real time with practically very low computational time needed, which on the contrary is something that the original MPC guidance is not able to do, since at each time-step it needs the time to perform the cost function optimization. This aspect can be seen also under the view of the need of less powerful and power consuming computer systems for the application of this kind of guidance, since the employment of an already trained neural network is surely less demanding with respect to the real time computation of a cost function minimization process. Another important aspect is the flexibility offered by the proposed system. In fact in principle the same procedure presented in this work could be applied to a large variety of landing situations and constraints by a correct redefinition of the learning environment. Moreover, the machine learning field is in continuous evolution and it is strongly possible that in the foreseeable future the progresses in this field will lead to a great enhancement in the techniques applied for the development of the presented guidance algorithm.
Defining the presented guidance system and facing new topics such as the ones related to machine learning has been a challenging but exciting experience, which has required effort and time to be completed. Of course what proposed

is far from being something already applicable since several enhancements are still needed, but anyway it represents a method that could offer an important option in the pinpoint landing context for the future, as showed by the promising obtained results and the advantages described above. In this sense the initial goal of defining a technique that could be successfully applied for an autonomous pinpoint landing process on bodies with low gravity and no atmosphere has been achieved. As said, to proceed towards the definitive implementation of a method of this kind several steps need to be done, such as the implementation of a more complete environment with a 6D dynamics and a deeper optimization of the overall learning process, that could include the application of Reinforcement Learning algorithms that have not been taken in consideration during this work. Once these aspects will be overcome, I am confident that the presented approach could provide a significant contribution for the future space missions.

Acronyms

ZEM-ZEV	Zero-Effort-Miss/Zero-Effort-Velocity
MSL	Mars Space Laboratory
UPG	Universal Powered Guidance
OSG	Optimal Sliding Guidance
LQR	Linear quadratic regulator
GPN	Generalized Proportional Navigation
MPC	Model Predictive Control
IGCRL	Reinforcement Learning derived Integrated Guidance and Control
G-Fold	Guidance for Fuel Optimal Large Divert
RL	Reinforcement Learning
RB	Reinforce with Baseline
AC	Actor-Critic
A2C	Advantage Actor-Critic
IPM	Interior Point Methods
HG	Hybrid Guidance
LOS	Line of Sight
EDL	Entry, Descent and Landing
MDP	Markov Decision Process
API	Application Programming Interface

Bibliography

[1] A. M. SanMartin G. Singh and E. C. Wong. "Guidance and control design for powered descent and landing on Mars". In: *IEEE* (2007).

[2] A. M. SanMartin S. W. Lee and E. C. Wong. "The Development of the MSL Guidance, Navigation, and Control System for Entry, Descent, and Landing". In: (2013).

[3] Charles J. Byrne. *Travels with curiosity*. Springer, 2020.

[4] `https://mars.nasa.gov/resources/25491/perseverance-rover-landing-ellipse-in-jezero-crater/`. (Visited on 02/22/2021).

[5] Jens Biele Stephan Ulamec. "Surface elements and landing strategies for small bodies missions – Philae and beyond". In: *Science Direct* (June 2009).

[6] Richard Linares Roberto Furfaro Andrea Scorsoglio and Mauro Massari. "Adaptive Generalized ZEM-ZEV Feedback Guidance for Planetary Landing via a Deep Reinforcement Learning Approach". In: (Mar. 2020).

[7] MiMi Aung Brhçet Açikmese et al. "Flight Testing of Trajectories Computed by G-FOLD: Fuel Optimal Large Divert Guidance algorithm for Planetary Landing". In: (2013).

[8] B. Açikmese J. M. Carson and L. Blackmore. "Lossless Convexification of Powered-Descent Guid- ance with Non-Convex Thrust Bound and Pointing Constraints". In: (2011).

[9] B. Açikmese and S.R.Ploen. "Convex Programming Approach to Powered Descent Guidance for Mars Landing". In: *AIAA Journal of Guidance, Control and Dynamics* (2007).

[10] Ping Lu. "Propellant-Optimal Powered Descent Guidance". In: *Journal of Guidance, Control, and Dynamics* (2017).

[11] Roberto Furfaro Daniel R. Wibben and Ricardo G. Sanfelice. "Optimal Lunar Landing and Retargeting using a Hybrid Control Strategy". In: (2013).

[12] Fang-Bo Yeh Ciann-Dong Yang and Jen-Heng Ghen. "The Closed-Form Solution of Generalized Proportional Navigation". In: (2015).

[13] Richard Linares Brian Gaudet and Roberto Furfaro. "Deep Reinforcement Learning for Six Degree-of-Freedom Planetary Powered Descent and Landing". In: (Oct. 2018).

[14] Yao Zhang Ting Wang Yanning Guo, Guangfu Ma, and Zimu Liang. "Model Predictive Control Guidance for Constrained Mars Pinpoint Landing". In: (2016).

[15] Unsik Lee and Mehran Mesbahi. "Constrained Autonomous Precision Landing via Dual Quaternions and Model Predictive Control". In: *Journal of Guidance, Control, and Dynamics* (Sept. 2016).

[16] H. Boehnhardt J.-P. Bibring H. Rosenbauer et al. "The Rosetta Lander ("Philae") Investigations". In: *Science Direct* (Dec. 2006).

[17] Peter Henderson Vincent François-Lavet et al. "An Introduction to Deep Reinforcement Learning". In: *Foundations and Trends in Machine Learning* 11.3-4 (Dec. 2018).

[18] Julien Vitay. *Deep Reinforcement Learning.*

[19] Rowel Atienza. *Advanced Deep Learning with Keras.* Packt, 2018.

[20] Sutton and Barto. *Reinforcement Learning: An Introduction.*

[21] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* (2015).

[22] `https://gym.openai.com/envs/`. (Visited on 02/18/2021).

[23] Marek Grześ. "Reward Shaping in Episodic Reinforcement Learning". In: (2017).

[24] Peng Yuming Li Shuang. "Command generator tracker based direct model reference adaptive tracking guidance for Mars atmospheric entry". In: *Elsevier* (Aug. 2011).